# Topics Covered : Distributed Databases: Introduction to Distributed Database Systems, Distributed Database System Architecture;

***Dr. V.C.V. Rao***

***Office @Pune University Campus, Pune, India***

*CDAC is a R&D Institute of Dept. of Electronics & Information Technology (DeitY) , Ministry of Communications & Information Technology (MCIT); Govt. of India*

**NIT-Warangal:**

**Dept.  Of  Comp. Sc & Engg.**

**July – Nov 2023**

**Course Name (CS5305) :   Advanced DataBases**

**Course Name (CS5309) : Advanced DataBases**

# Advanced Data Bases (References & Source for Presentation

The Presentation is prepared based on Author's experience on various research projects on Distributed Computing – Distributed Data Bases (NoSQL) well as references given in this presentation.

**Source :**

*Text Books, Research Articles, Web Sites as indicated in many slides and References of this presentation*


1. M T Ozsu, Patrick Valduriez, Principles of Distributed Database Systems, Prentice Hall, 3rd Edition 2011

## Course Name : Advanced databases

**M.C.A 2nd Year, July 2023**

**Syllabus:**

**UNIT 1 :** Distributed Databases: Introduction to Distributed Database Systems, Distributed Database System Architecture;

**UNIT 2 :** Top-Down Approach, Distributed Database Design Issues, Fragmentation, Allocation, Database Integration, Bottom-up approach, Schema Matching, Schema Integration, Schema Mapping;

**UNIT 3:** Data and Access Control, View Management, Data Security; Query processing problem, Objectives of Query processing, Complexity of Relational Algebra Operations, Characterization of Query Processors, Layers of Query Processing;

**UNIT 4 :** Query Decomposition, Normalization, Analysis, Elimination of Redundancy and Rewriting;

**UNIT 5 :** Localization of Distributed Data, Reduction for primary Horizontal, Vertical, derived Fragmentation;

**UNIT 6** : Distributed Query Execution, Query Optimization, Join Ordering, Static& Dynamic Approach, Semi-joins, Hybrid Approach;

## Course Name : Advanced databases

**Syllabus:** **M.C.A 2nd Year, July 2023**

**UNIT 7 :** Taxonomy of Concurrency control Mechanisms, Lock-Based Concurrency Control, Timestamp-Based Concurrency Control, Optimistic Concurrency Control, Deadlock Management;

**UNIT 8 :** Heterogeneity issues Advanced Transaction Models, Distributed systems 2PC& 3PC protocols, Replication protocols, Replication and Failures, HotSpares;

**UNIT 9:** Parallel Databases: Introduction to Parallel Databases, Parallel Database System Architectures, Parallel Data Placement, Full Partitioning; Parallel Query Processing, Query Parallelism;

**UNIT 10 :** Parallel Query Optimization, Search Space, Cost Model, Search Strategy; Load Balancing.

**Text Books/ Reference Books / Online Resources:**

M T Ozsu, Patrick Valduriez, Principles of Distributed Database Systems, Prentice Hall, 1999.

S. Ceri and G. Pelaggati, Distributed Database System Principles and Systems, MGH, 1985.

# Course Name : Advanced databases

## Topics Covered in this Lecture

**UNIT 1 : Distributed Databases: Introduction to Distributed Database Systems, Distributed Database System Architecture;**

**UNIT 2 : Top-Down Approach, Distributed Database Design Issues, Fragmentation, Allocation, Database Integration, Bottom-up approach, Schema Matching, Schema Integration, Schema Mapping;**

# Chapter-02 & Chapter-03 of Reference Book No. 1

# Background

1. Two technological bases for distributed database technology**: database management** and **computer networks**

2. Overview of the concepts in these two fields that **are more important from** the perspective of distributed database technology.

# Overview of Relational DBMS

1. Define the terminology and framework used

2. Most of the distributed database technology has been developed using the relational mode

❖ Relational Database Concepts
   - Normalisation
   - Relational Data Languages

❖ Review of Computer Networks

# Relational Database Concepts

A database is a structured collection of data related to some real-life phenomena that we are trying to model. A relational database is one where the database structure is in the form of tables. Formally, a relation **R** defined over **n** sets $D_1, D_2, : : : , D_n$ (not necessarily distinct) is a set of n-tuples (or simply tuples)

$$\langle d_1, d_2, : : : ; d_n \rangle$$

such that

$$d_1 \in D_1, d_2 \in D_2, : , d_n \in D_n.$$

# Relational Database Concepts

Example : Use a database that models an engineering company. The entities to be modelled are the employees (**EMP**) and projects (**PROJ**).

For each employee, keep track of the

- employee number (**ENO**),
- name (**ENAME**),
- title in the company (**TITLE**),
- Salary (**SAL**),
- identification number of the project(s) the employee is working on (**PNO**),
- responsibility within the project (**RESP**), and duration of the assignment to the project (**DUR**) in months.
- Similarly, for each project, to store the project number (**PNO**), the project name (**PNAME**), & the project budget (**BUDGET**).

# Relational Database Concepts

## Sample Database Scheme

EMP

| ENO | ENAME | TITLE | SAL | PNO | RESP | DUR |
|---|---|---|---|---|---|---|

PROJ

| PNO | PNAME | BUDGET |
|---|---|---|

The ***relation schemas*** for this database can be defined as follows:

**EMP(<u>ENO</u>, ENAME, TITLE, SAL, <u>PNO</u>, RESP, DUR)**

**PROJ(**PNO**,PNAME, BUDGET)**

In relation scheme **EMP,** there are seven attributes: **ENO, ENAME, TITLE, SAL, PNO, RESP, DUR.** The values of **ENO** come from the domain of all valid employee numbers, say $D_1$, the values of **ENAME** come from the domain of all valid names, say $D_2$, and so on. Note that each attribute of each relation does not have to come from a distinct domain.

# Relational Database Concepts

- Various attributes within a relation or from a number of relations may be defined over the same domain

- The ***key*** of a relation scheme is the minimum non-empty subset of its attributes such that the values of the attributes comprising the ***key*** uniquely identify each tuple of the relation.

- The attributes that make up ***key*** are called prime attributes. The
- superset of a ***key*** is usually called a ***superkey***. Thus in our example the ***key*** of ***PROJ*** is ***PNO***, and that of ***EMP*** is the set (***ENO, PNO***).

# Relational Database Concepts

- Each relation has at least one ***key.*** Sometimes, there may be more than ***one possibility*** for the key.

- In such cases, each alternative is considered ***a candidate key***, and one of the candidate keys is chosen as the ***primary key***, which we denote by ***<u>underlining.</u>***

- The number of attributes of a relation defines its ***degre***e, whereas the number of tuples of the relation defines its ***cardinality***.

- In tabular form, the example database consists of two tables, as shown in Figure. The columns of the tables correspond to the attributes of the relations; if there were any information entered as the rows, they would correspond to the tuples

# Relational Database Concepts

- In tabular form, the example database consists of two tables, as shown in Figure.

    - The columns of the tables correspond to the attributes of the relations;

- if there were any information entered as the rows, they would correspond to the tuples.

- The empty table, showing the structure of the table, corresponds to the relation schema; when the table is filled with rows, it corresponds to a relation instance.

# Relational Database Concepts

EMP

| ENO | ENAME | TITLE | SAL | PNO | RESP | DUR |
|---|---|---|---|---|---|---|
| E1 | J. Doe | Elect. Eng. | 40000 | P1 | Manager | 12 |
| E2 | M. Smith | Analyst | 34000 | P1 | Analyst | 24 |
| E2 | M. Smith | Analyst | 34000 | P2 | Analyst | 6 |
| E3 | A. Lee | Mech. Eng. | 27000 | P3 | Consultant | 10 |
| E3 | A. Lee | Mech. Eng. | 27000 | P4 | Engineer | 48 |
| E4 | J. Miller | Programmer | 24000 | P2 | Programmer | 18 |
| E5 | B. Casey | Syst. Anal. | 34000 | P2 | Manager | 24 |
| E6 | L. Chu | Elect. Eng. | 40000 | P4 | Manager | 48 |
| E7 | R. Davis | Mech. Eng. | 27000 | P3 | Engineer | 36 |
| E8 | J. Jones | Syst. Anal. | 34000 | P3 | Manager | 40 |

PROJ

| PNO | PNAME | BUDGET |
|---|---|---|
| P1 | Instrumentation | 150000 |
| P2 | Database Develop. | 135000 |
| P3 | CAD/CAM | 250000 |
| P4 | Maintenance | 310000 |

Figure. Sample Database Instance

# Overview of Relational DBMS - Normalization

The aim of normalization is to eliminate various anomalies (or undesirable aspects) of a relation in order to obtain “better” relations. The following four problems might exist in a relation scheme:

- Repetition anomaly
- Update anomaly
- Insertion anomaly
- Deletion anomaly

Normalization transforms arbitrary relation schemes into ones without these problems. A relation with one or more of the above mentioned anomalies is split into two or more relations of a higher normal form.

# Overview of Relational DBMS-Relational Data Languages

Data manipulation languages developed for the relational model (commonly called ***query languages***) fall into two fundamental groups: ***relational algebra languages*** and ***relational calculus languages***. The difference between them is based on how the user query is formulated.

- The relational algebra is procedural in that the user is expected to specify, using certain high-level operators, how the result is to be obtained.

- The relational calculus, on the other hand, is non-procedural; the user only specifies the relationships that should hold in the result.

# Relational Data Languages - Relational Algebra

## Relational Algebra

- Relational algebra consists of a set of operators that operate on relations.

- Each operator takes one or two relations as operands and produces a result relation, which, in turn, may be an operand to another operator.

- These operations permit the querying and updating of a relational database.

# Relational Data Languages - Relational Algebra

## Normalized Relations

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 |
|---|---|---|
| P1 | Instrumentation | 150000 |
| P2 | Database Develop. | 135000 |
| P3 | CAD/CAM | 250000 |
| P4 | Maintenance | 310000 |

PAY

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

# Relational Data Languages - Relational Algebra

## Normalized Relations

There are five fundamental relational algebra operators and five others that can be defined in terms of these. The fundamental operators are
**selection, projection, union, set difference, and Cartesian product**.

The first two of these operators are unary operators, and the last three are binary operators.

The additional operators that can be defined in terms of these fundamental operators are

**intersection, θ - join, natural join, semi-join and division**.

# Relational Data Languages - Relational Algebra

## Unary Relational Operations

SELECT (symbol: σ)
PROJECT (symbol: π)
RENAME (symbol: ρ)

## Relational Algebra Operations From Set Theory

UNION (υ)
INTERSECTION ( ∩ ),
DIFFERENCE (-)
CARTESIAN PRODUCT ( x )

## Binary Relational Operations

JOIN
DIVISION

## Relational Calculus

In Relational calculus-based languages, instead of specifying how to obtain the result, one specifies what the result is by stating the relationship that is supposed to hold for the result.

Relational calculus languages fall into two groups: tuple relational calculus and domain relational calculus. The difference between the two is in terms of the primitive variable used in specifying the queries.

# Chapter-03

## Part-01

# Distributed Database Design

1. Top-Down Design Process

2. Distribution Design Issues

3. Fragmentation

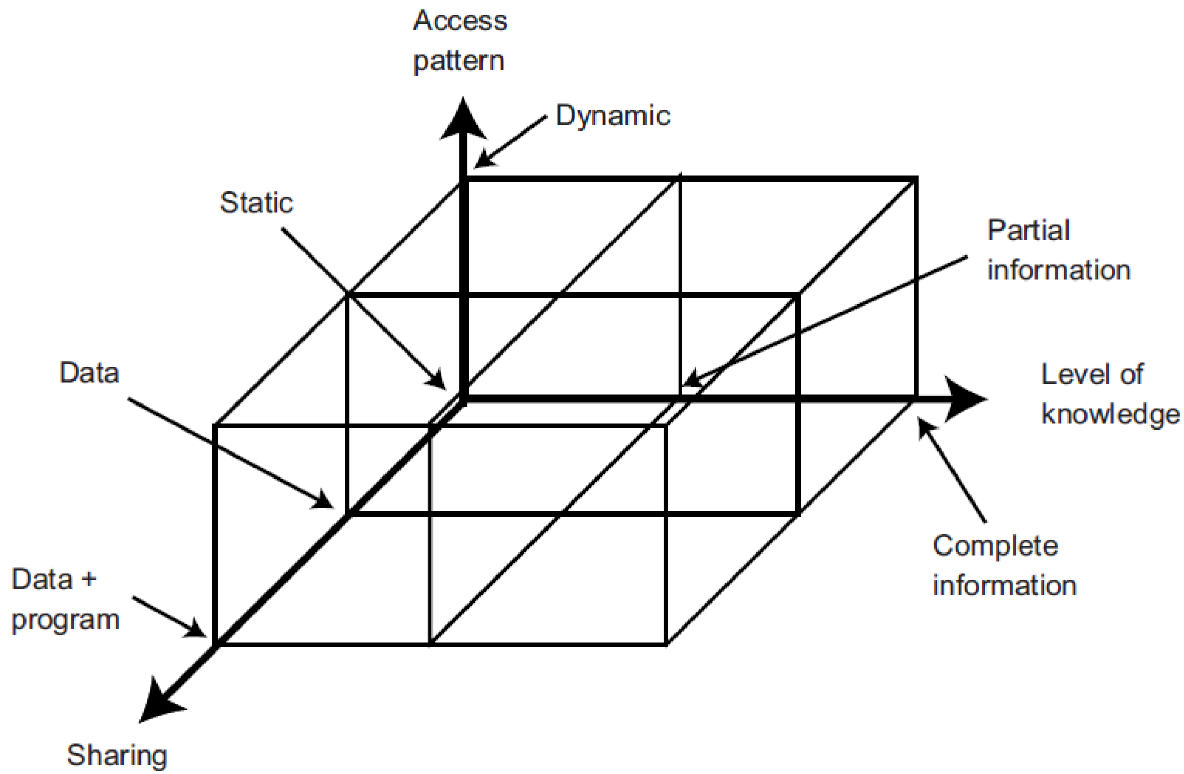
4. Allocation

5. Data base Integration

# Distributed Database Design

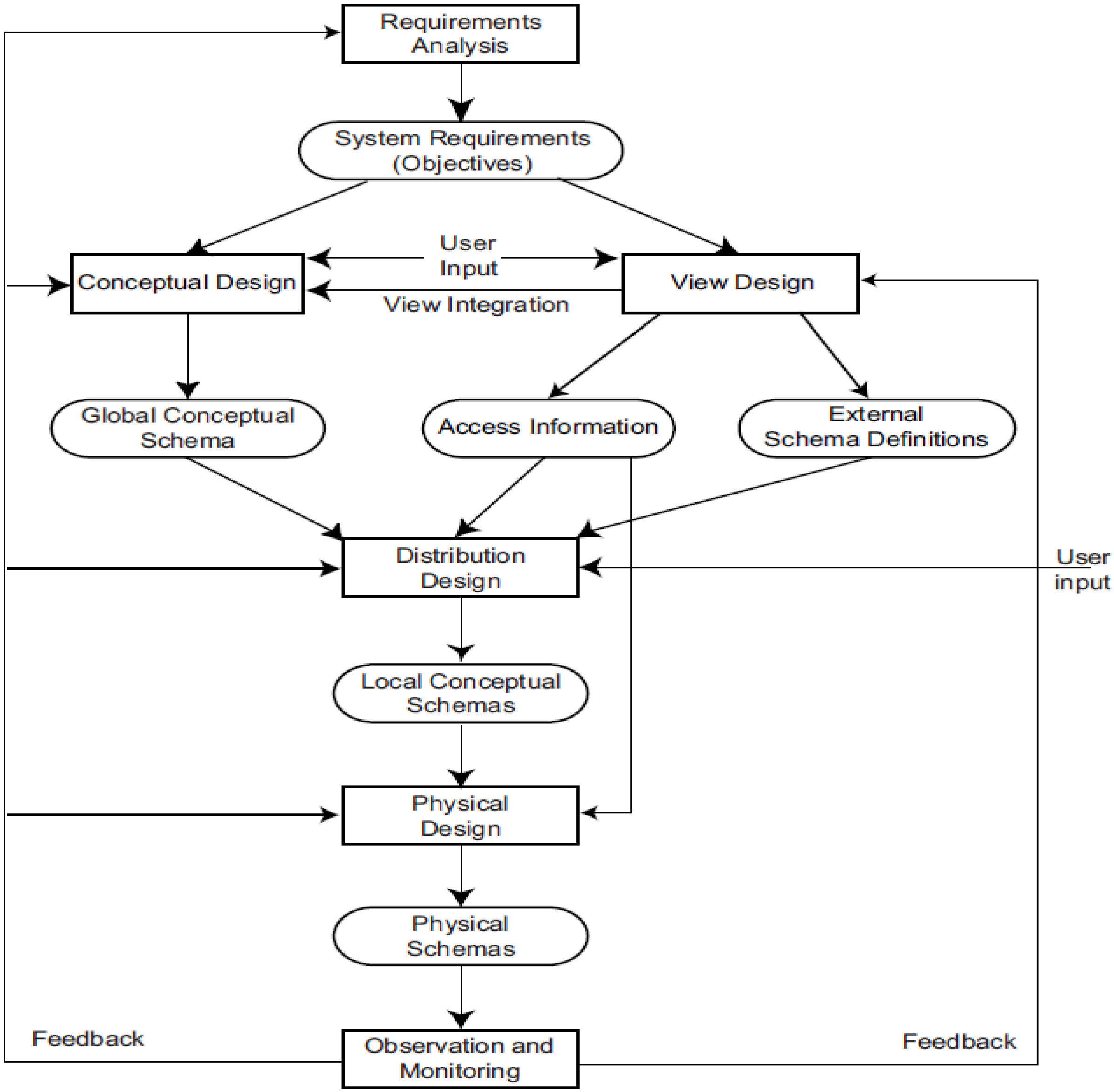
The organization of distributed systems can be investigated along three orthogonal dimensions (Figure)

1. Level of sharing
2. Behavior of access patterns
3. Level of knowledge on access pattern behaviour



# Top Down Design Process

The organization of distributed systems can be investigated along three orthogonal dimensions (Figure)



# Top Down Design Process

1. The conceptual design, on the other hand, is the process by which the enterprise is examined to determine entity types and relationships among these entities.

2. One can possibly divide this process into **two related activity groups**

- **entity analysis** and **functional analysis (**This is all part of **Centralized Data Base Design**)

# Top Down Design Process

1. The **global conceptual schema (GCS)** and access pattern information collected as a result of view design are inputs to the distribution design step.

2. The objective at this stage is to design the **local conceptual schemas (LCSs)** by distributing the entities over the sites of the distributed system

3. Rather than distributing relations, it is quite common to divide them into subrelations, called **fragments**, which are then distributed. Thus, the distribution design activity consists of two steps: **fragmentation** and **allocation.**

# Distributed Design Issues

**Questions should be addresses**

1. Why fragment at all?
2. How should we fragment?
3. How much should we fragment?
4. Is there any way to test the correctness of decomposition?
5. How should we allocate?
6. What is the necessary information for fragmentation and allocation?

# Reasons for fragmentation

**Questions should be addresses**

With respect to fragmentation, the important issue is the appropriate unit of distribution. A relation is not a suitable unit, for a number of reasons.

- **First**, application views are usually subsets of relations. (consider subsets of relations as distribution units.)
  **Second**, if the applications that have views defined on a given relation reside at different sites, two alternatives can be followed (Replication at one site or all sites
- **Finally,** the decomposition of a relation into fragments, each being treated as a unit, permits a number of transactions to execute concurrently

# Reasons for fragmentation

**Fragmentation raises difficulties (Problem 1 & Problem 2)**

Problem 1 :

- If the applications have conflicting requirements that prevent decomposition of the relation into mutually exclusive fragments
- Performance Degradation - Example necessary to retrieve data from two fragments and then take their join, which is costly
- Minimizing distributed joins is a fundamental fragmentation issue.

# Reasons for fragmentation

**Problem 2 :**

- The <u>**second problem**</u> is related to semantic data control, specifically to integrity checking.

- As a result of fragmentation, attributes participating in a dependency may  be decomposed into different fragments that might be allocated to different sites.

# Fragmentation Alternatives

**Problem 2 :**

- Relation instances are essentially tables, so the issue is one of finding alternative ways of dividing a table into smaller ones. There are clearly two alternatives for this: dividing it **horizontally** or dividing it vertically.

we use a modified version of the relational database scheme developed .

We have added to the **PROJ** relation a new attribute (**LOC**) that indicates the place of each project.

# Fragmentation Alternatives

**Example 1 : We have added to the PROJ relation a new attribute (LOC) that indicates the place of each project.**

- Figure (A) depicts the database instance we will use. Figure (B) shows the **PROJ** relation of Figure (A) divided horizontally into two relations. Sub-relation

  - **PROJ1** contains information about projects whose budgets are less than \$200,000, whereas

  - **PROJ2** stores information about projects with larger budgets.

# Fragmentation Alternatives

**Example 1 : We have added to the PROJ relation a new attribute (LOC) that indicates the place of each project.**

- Figure (A) depicts the database instance we will use. Figure (B) shows the **PROJ** relation of Figure (A) divided horizontally into two relations. Sub-relation

  - **PROJ1** contains information about projects whose budgets are less than $200,000, whereas
  - **PROJ2** stores information about projects with larger budgets.

Note that in the case of horizontal fragmentation, the “item” typically refers to a tuple, while in the case of vertical fragmentation, it refers to an attribute.

# Fragmentation Alternatives

**Example 1 : We have added to the PROJ relation a new attribute (LOC) that indicates the place of each project.**

$PROJ_1$

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

$PROJ_2$

| PNO | PNAME | BUDGET | LOC |
|---|---|---|---|
| P3 | CAD/CAM | 255000 | New York |
| P4 | Maintenance | 310000 | Paris |

Fig. (B) Example of Horizontal Partitioning

# Fragmentation Alternatives

**Example 3.2. Figure (C) shows the PROJ relation of Figure (A) partitioned vertically into two subrelations PROJ1** and **PROJ2.**

- **PROJ1** contains only the information about project budgets, whereas
- **PROJ2** contains project names and locations. It is important

Note that in the case of horizontal fragmentation, the “item” typically refers to a tuple, while in the case of vertical fragmentation, it refers to an attribute.

# Fragmentation Alternatives

**Example 3.2. Figure (C) shows the PROJ relation of Figure (A) partitioned vertically into two subrelations PROJ1** and **PROJ2.**

$PROJ_1$

| 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 |

Figure C. Example of Vertical Partitioning

In general, the applications need to be characterized with respect to a number of parameters

# Fragmentation Alternatives

- **Degree of Fragmentation (**Performance of query execution **)**

- **Correctness Rules of Fragmentation (**

  - Completeness,
  - Reconstruction, &
  - Disjoint Ness

# Allocation Alternatives

- Assuming that the database is fragmented properly, one has to decide on the allocation of the fragments to various sites on the network.
- When data are allocated, it may either be replicated or maintained as a single copy.
  - The reasons for replication are reliability and efficiency of read-only queries.
  - The decision regarding replication is a trade-off that depends on the ratio of the read-only queries to the update queries. This decision affects almost all of the distributed DBMS algorithms and control functions.

# Allocation Alternatives

- A non-replicated database (commonly called a partitioned database) contains fragments that are allocated to sites, and there is only one copy of any fragment on the network.
- In case of replication, either the database exists in its entirety at each site (**fully replicated database**), or fragments are distributed to the sites in such a way that copies of a fragment may reside in multiple sites (**partially replicated database**).

**Note :** In the latter the number of copies of a fragment may be an input to the allocation algorithm or a decision variable whose value is determined by the algorithm.

# Allocation Alternatives

## Comparison of Replication Alternatives

| | Full replication | Partial replication | Partitioning |
|---|---|---|---|
| QUERY PROCESSING | Easy | Same difficulty | |
| DIRECTORY MANAGEMENT | Easy or nonexistent | Same difficulty | |
| CONCURRENCY CONTROL | Moderate | Difficult | Easy |
| RELIABILITY | Very high | High | Low |
| REALITY | Possible application | Realistic | Possible application |

# Fragmentation

❖ **Fragmentation Strategies & Algorithms**

**Horizontal Fragmentation**

Partitions a relation along its tuples.

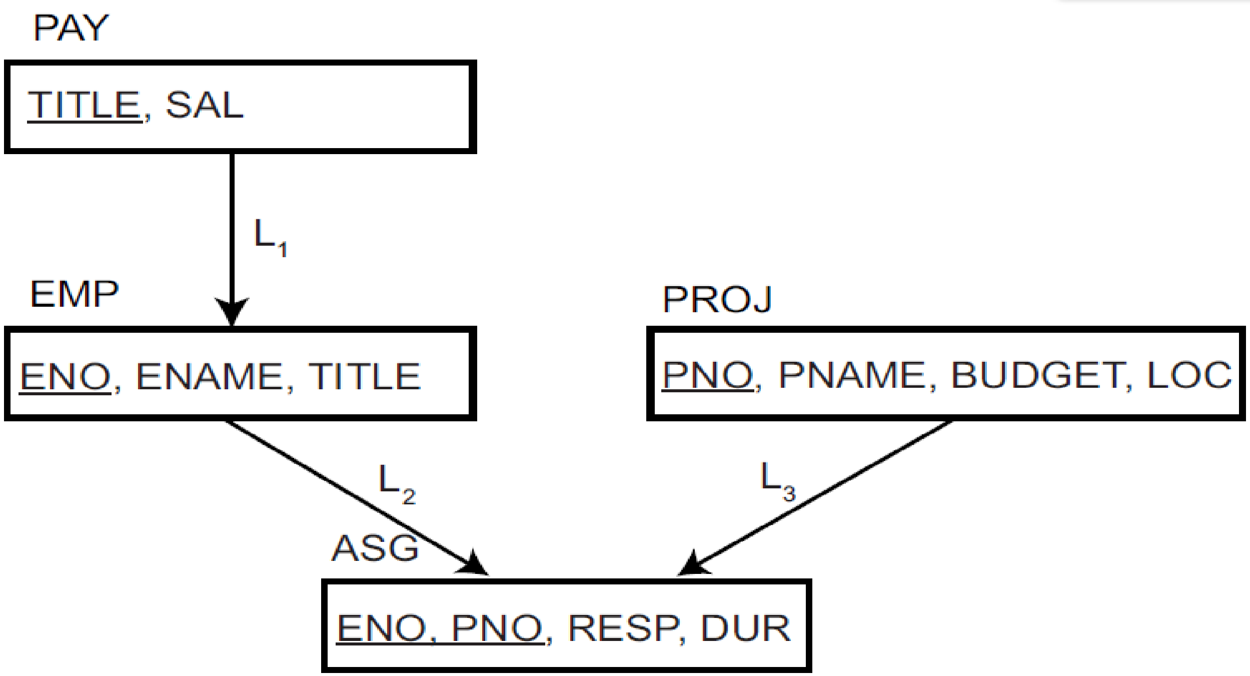
- Primary horizontal fragmentation &
- Derived horizontal fragmentation

❖ Information Requirements of Horizontal Fragmentation

- Database Information
- Application Information.

# Fragmentation

❖ **Fragmentation Strategies & Algorithms**




Expression of Relationships Among Relations Using Links

# Primary Horizontal Fragmentation

A **primary horizontal fragmentation** is defined by a selection operation on the owner relations of a database schema.

Therefore, given relation R, its horizontal fragments are given by

$$\mathbf{R_i = \sigma_{Fi}(R); 1 \leq i \leq w}$$

Where $\mathbf{F_i}$ is the selection formula used to obtain fragment $R_i$ (also called the fragmentation predicate). Algorithms are required

The decomposition of relation PROJ into horizontal fragments $PROJ_1$ and $PROJ_2$ in Example (Earlier ) is defined as follows$_1$:

$PROJ_1 = \sigma_{BUDGET\ \ 200000}\ (PROJ)$

$PROJ_2 = \sigma_{BUDGET > 200000}\ (PROJ)$

# Primary Horizontal Fragmentation

$PROJ_1$

| 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 |

Primary Horizontal Fragmentation of Relation PROJ

$$Pr = \{\text{LOC=“Montreal”, LOC=“New York”, LOC=“Paris”}, \\ \text{BUDGET} \leq 200000, \text{BUDGET} > 200000\}$$

# Primary Horizontal Fragmentation

An iterative algorithm that would generate a **complete and minimal set of predicates** $P_r'$ given a set of simple predicates $P_r$.

$PAY_1$

| TITLE | SAL |
|---|---|
| Mech. Eng. | 27000 |
| Programmer | 24000 |

$PAY_2$

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

Horizontal Fragmentation of Relation PAY

# Primary Horizontal Fragmentation

## Horizontal Partitioning of Relation PROJ

$PROJ_1$

| PNO | PNAME | BUDGET | LOC |
|---|---|---|---|
| P1 | Instrumentation | 150000 | Montreal |

$PROJ_3$

| PNO | PNAME | BUDGET | LOC |
|---|---|---|---|
| P2 | Database Develop. | 135000 | New York |

$PROJ_4$

| PNO | PNAME | BUDGET | LOC |
|---|---|---|---|
| P3 | CAD/CAM | 250000 | New York |

$PROJ_6$

| PNO | PNAME | BUDGET | LOC |
|---|---|---|---|
| P4 | Maintenance | 310000 | Paris |

# Derived Horizontal Fragmentation

A derived horizontal fragmentation is defined on a member relation of a link according to a selection operation specified on its owner.

It is important to remember two points.

- First, the link between the owner and the member relations is defined as an **equi-join**.
- Second, an **equi-join** can be implemented by means of **semijoins**.

# Checking for Correctness

- **Completeness :** The completeness of a primary horizontal fragmentation is based on the selection predicates used.
- **Reconstruction :** Reconstruction of a global relation from its fragments is performed by the union operator in both the primary and the derived horizontal fragmentation. Thus, for a relation R with fragmentation

$$F_R = f\{R_1;R_2; : : : ;R_w\} ,$$

$$R = \bigcup R_i, \quad \forall R_i \in F_R$$

- **Disjointness.**

# Derived Horizontal Fragmentation

$EMP_1$

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

$EMP_2$

| 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. |

Derived Horizontal Fragmentation of Relation EMP

# Vertical Fragmentation

- The objective of vertical fragmentation is to partition a relation into a set of smaller relations so that many of the user applications will run on only one fragment.

- In this context, an "optimal" fragmentation is one that produces a fragmentation scheme which minimizes the execution time of user applications that run on these fragments.

- a vertical fragmentation of a relation **R** produces fragments $R_1, R_2, : : : ; R_r$, each of which contains a subset of R's attributes as well as the primary key of **R**.

# Vertical Fragmentation

- Vertical fragmentation has been investigated within the context of centralized database systems as well as distributed ones.
- Its motivation within the centralized context is as a design tool, which allows the user queries to deal with smaller relations, thus causing a smaller number of page accesses
- Vertical partitioning is inherently more complicated than horizontal partitioning. This is due to the total number of alternatives that are available.

# Vertical Fragmentation

- Two types of heuristic approaches exist for the vertical fragmentation of global relations:

  **Grouping and Splitting**

- Information Requirements of Vertical Fragmentation (Based on Application)

# Vertical Fragmentation

- Consider relation PROJ of Figure (Earlier) Assume that the following applications are defined to run on this relation. In each case we also give the SQL specification.

- **q1:** Find the budget of a project, given its identification number.

```
SELECT  BUDGET
FROM    PROJ
WHERE   PNO=Value
```

# Vertical Fragmentation

- **q2:** Find the names and budgets of all projects.

  SELECT PNAME, BUDGET

  FROM PROJ

- **q3:** Find the names of projects located at a given city.

  SELECT PNAME

  FROM PROJ

  WHERE LOC=Value

- **q4:** Find the total project budgets for each city.

  SELECT SUM (BUDGET)

  FROM PROJ

  WHERE LOC=Value

# Vertical Fragmentation

- According to these four applications, the attribute usage values can be defined. As a notational convenience, we let

$A_1$ = PNO, $A_2$ = PNAME, $A_3$ = BUDGET, and $A_4$ = LOC.

- The usage values are defined in matrix form (Figure ), where entry (i, **j**) denotes use($\mathbf{q_i}$, $\mathbf{A_j}$ ).

- Attribute usage values are not sufficiently general to form the basis of attribute splitting and fragmentation.

# Vertical Fragmentation

- **A**ttribute **A**ffinity matrix (**AA**)

$$
\begin{array}{c c}
 & \begin{array}{cccc} A_1 & A_2 & A_3 & A_4 \end{array} \\
\begin{array}{c} q_1 \\ q_2 \\ q_3 \\ q_4 \end{array} & \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}
\end{array}
$$

# Vertical Fragmentation

- **Clustering Algorithm :** The fundamental task in designing a vertical fragmentation algorithm is to find some means of grouping the attributes of a relation based on the attribute affinity values in AA. (attribute affinity matrix (AA).)

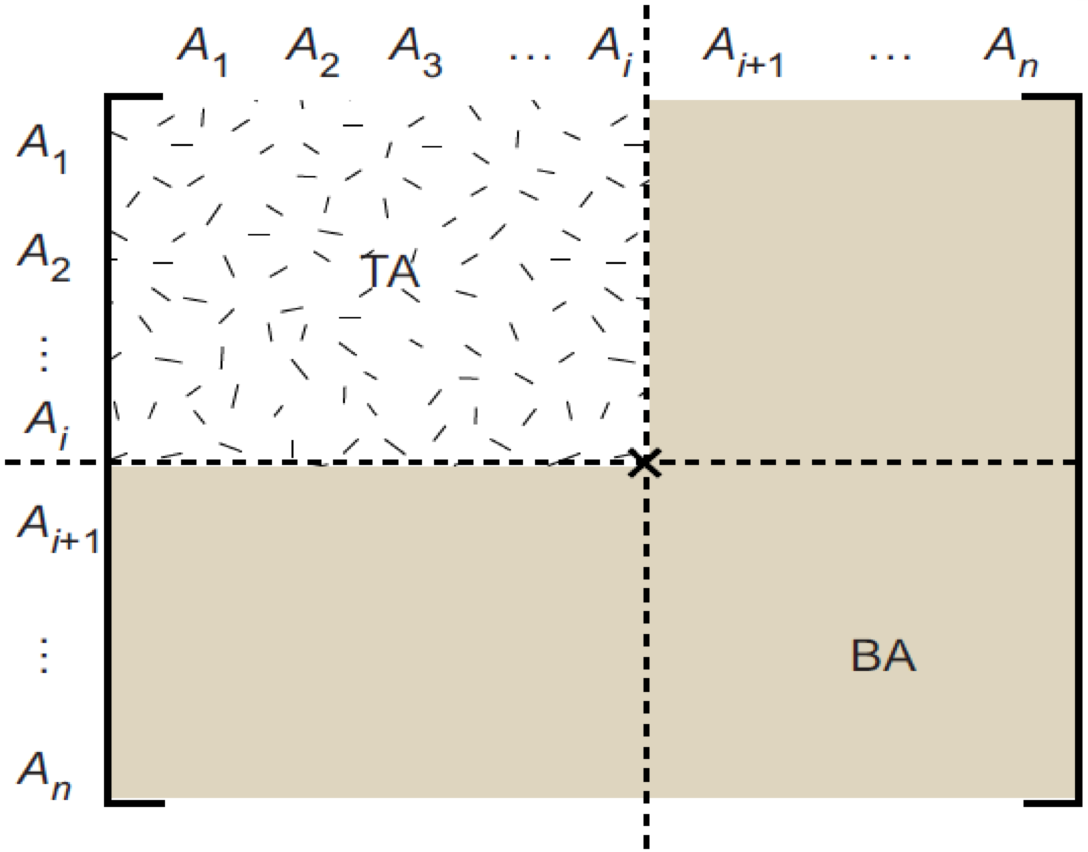
1. It is designed specifically to determine groups of similar items as opposed to, say, a linear ordering of the items (i.e., it clusters the attributes with larger affinity values together, and the ones with smaller values together).
2. The final groupings are insensitive to the order in which items are presented to the algorithm.
3. The computation time of the algorithm is reasonable: $O(n^2)$, where n is the number of attributes.
4. Secondary interrelationships between clustered attribute groups are identifiable

# Vertical Fragmentation

- **Partitioning Algorithm :** The objective of the splitting activity is to find sets of attributes that are accessed solely, or for the most part, by distinct sets of applications
- For example, if it is possible to identify two attributes, $A_1$ and $A_2$, which are accessed only by application $q_1$, and attributes $A_3$ and $A_4$, which are accessed by, say, two applications $q_2$ and $q_3$, it would be quite straightforward to decide on the fragments.
- The task lies in finding an algorithmic method of identifying these groups

# Vertical Fragmentation

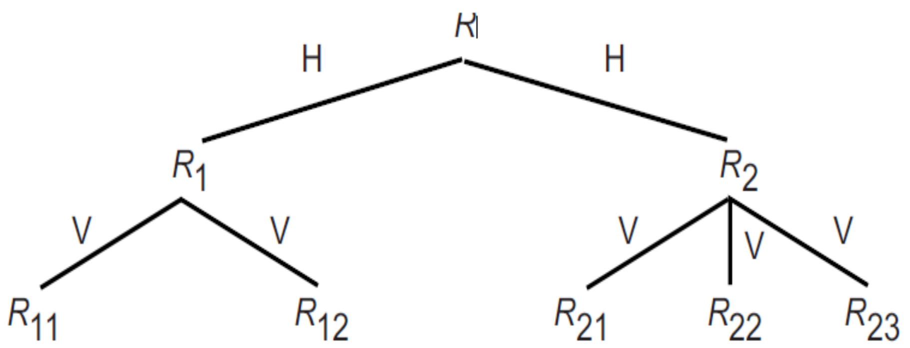
- **Partitioning Algorithm :** The objective of the splitting activity is to find sets of attributes that are accessed solely, or for the most pa




Locating a Splitting Point

# Hybrid Fragmentation

- In most cases a simple horizontal or vertical fragmentation of a database schema will not be sufficient to satisfy the requirements of user applications.
- In this case a vertical fragmentation may be followed by a horizontal one, or vice versa, producing a treestructured partitioning




Hybrid Fragmentation

# Allocation

- Assume that there are a set of fragments

  $F = \{F_1, F_2, : : : , F_n\}$

  and a distributed system consisting of sites

  $S = \{S_1; S_2; : : : , S_m\}$ on which a set of applications

  $Q = \{q_1, q_2, : : : , q_q\}$ is running.

- The allocation problem involves finding the "optimal" distribution of **F** to **S**.
- The optimality can be defined with respect to two measures : **Minimal Cost** and **Performance.**

# Allocation

- Information Requirements
- Application Information
- Site Information
- Allocation Model
- Constraints
- Total Cost
- Solution Methods

# Data Directory

- ❖ The distributed database schema needs to be stored and maintained by the system.

- ❖ This information is necessary during distributed query optimization,

- ❖ The schema information is stored in a data dictionary/directory, also called a catalog or simply a directory. A directory is a meta-database that stores a number of information

# Chapter-03

## Part-02