BITS Pilani

Dr. Manik Gupta
Associate Professor
Department of CSIS

BITS Pilani Hyderabad Campus

innovate achieve lead

BITS Pilani
Hyderabad Campus

Distributed Data Systems(CS G544)
Lecture 9-13

Monday, 2nd Sept 2024

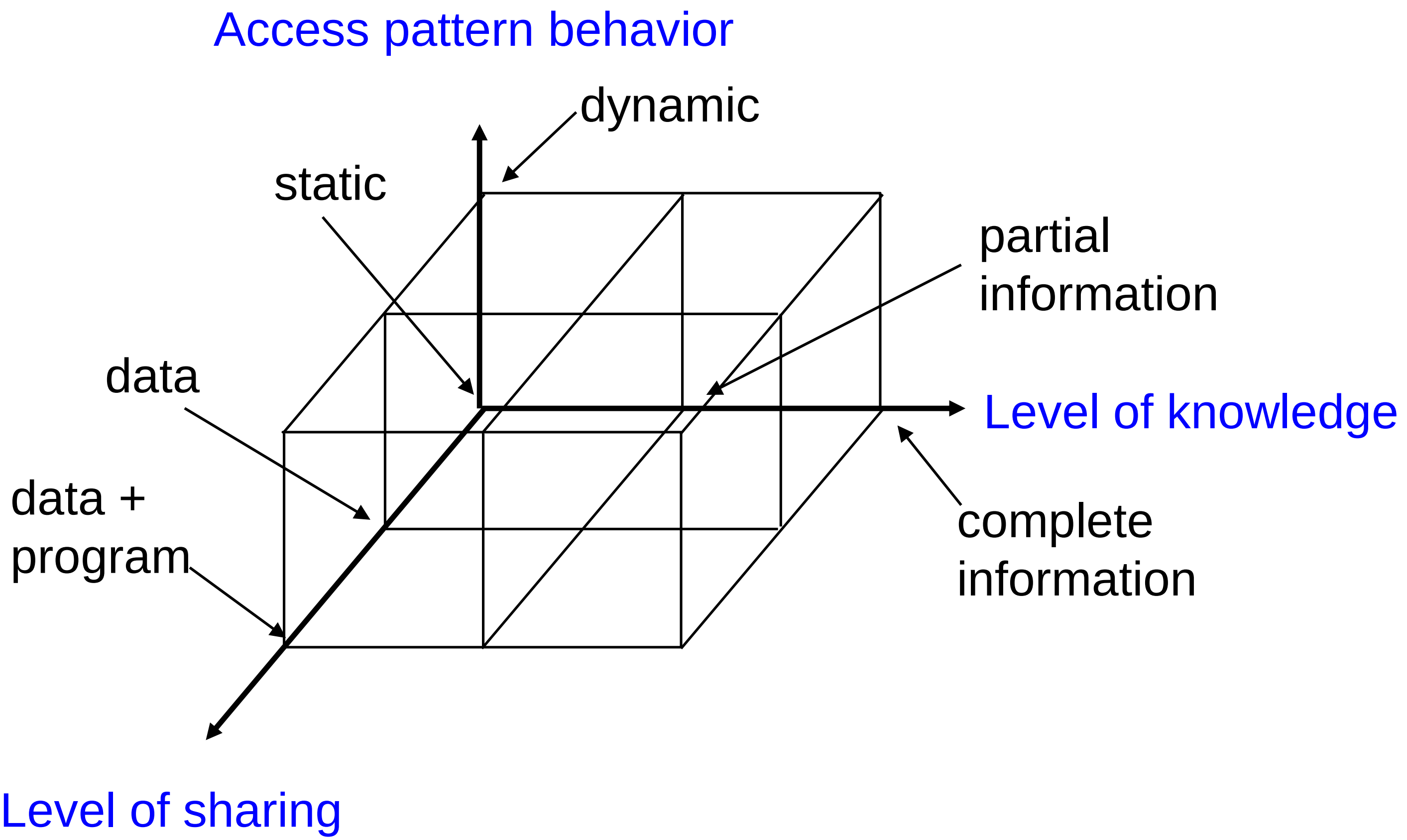
Lecture Recap

- Introduction to Distributed databases
- Distributed DBMS Architecture

Design Problem

- In the general setting :
  - 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 Distributed DBMS, the placement of applications entails
  - placement of the **distributed DBMS software**; and
  - placement of the **applications** that run on the databases

Dimensions of the Problem



Framework of Distribution

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

1. Level of sharing
   - there is **no sharing:** each application and its data execute at one site
   - level of data sharing; all the programs are replicated at all the sites, but data files are not
   - **data-plus-program sharing**, both data and programs may be shared

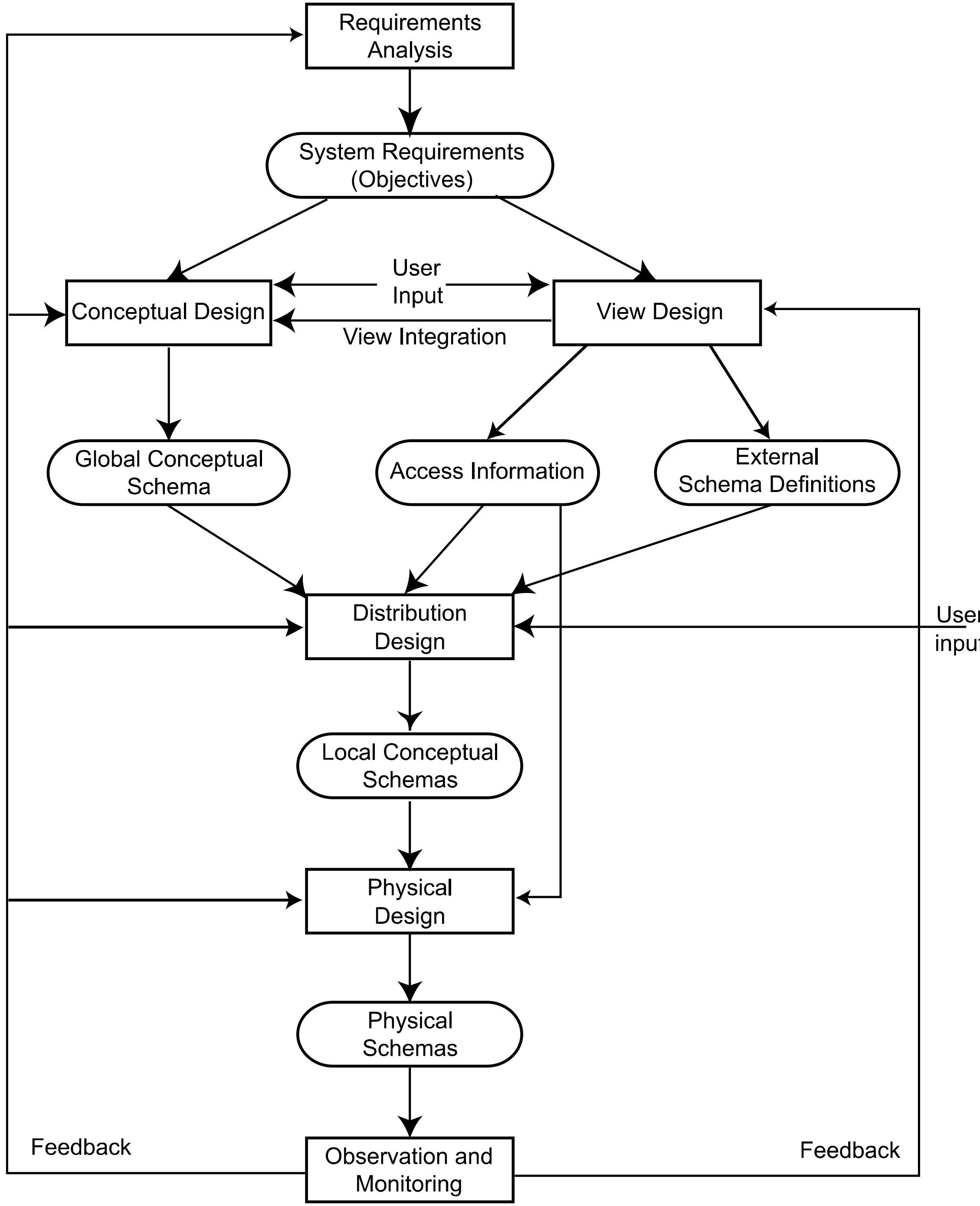
2. Behavior of access patterns
   - The access patterns of user requests may be **static**, so that they do not change over time, or **dynamic**.

3. Level of knowledge on access pattern behavior
   - **Complete information**, where the access patterns can reasonably be predicted and do not deviate significantly from these predictions, or **partial information**, where there are deviations from the predictions

Distribution Design

- Top-down
  - Mostly in designing systems from scratch
  - More suitable for tightly integrated, homogeneous distributed DBMSs
- Bottom-up
  - When the databases already exist at a number of sites, more suited to multi-databases

Top Down Design



Top down design process

- The activity begins with a **requirement analysis**. This gives details about the **data and processing needs of the users.**
- Requirements study also specifies where the final system is expected to stand with respect to the identified objectives of the DDBS.
- These objectives are defined w.r.t., performance, reliability, availability, economics, and expandability.

Top down design process

- The requirements document is input to two parallel activities- **View design** and **Conceptual design**
  - View Design: Deals with defining the interfaces for the end user.
  - Conceptual Design: Process by which the enterprise is examined to determine entity types and relationships.
- There is a relationship between view design and conceptual design-
  - Conceptual design can be interpreted as integration of user views
  - Conceptual design process must also consider the existing and future application requirements.

Top down design process

- In view and conceptual design activities the user needs to **specify the data entities** and must determine the **applications that will run on the database** as well as **statistical information** about these applications.
  - This includes specification of frequency of user applications, the volume of various information etc.

Top down design process

- From the conceptual design step we get the definition of Global Conceptual Schema (GCS).
- **GCS and access pattern information** are input to distributed design step.
- The objective of this stage is to design local conceptual schemas (LCSs) by distributing the entities (in GCS) over the sites in the distributed system.

Top down design process

- In general entities (relations) are the distribution units.
- Rather than distributing relations, it is quite common to divide them into sub-relations (fragments) which are then distributed.
- The distribution design activity consists of two steps-
  - Fragmentation
  - Allocation

Top down design process

- The last step in design process is **physical design**, which **maps LCSs to physical storage devices** available at corresponding sites. The inputs to this process are LCSs and access pattern information.
- The design and development activity of any kind is an ongoing process requiring constant monitoring and fine-tuning. Hence, observation and monitoring activity is also included in the process. The result is the feedback which will be sent to one or more of the earlier phases.

Distribution Design Issues

- Why fragment at all?
- How to fragment?
- How much to fragment?
- How to test correctness of decomposition?
- How to allocate?
- Information requirements for fragmentation and allocation?

Fragmentation

- Can we not just distribute relations?
- What is a reasonable unit of distribution?
  - Relation is not a suitable unit
    - views are subsets of relations ➔ locality of accesses defined on subsets
    - extra communication if whole relation is used (both in case of no/full replication)
  - Fragments of relations (sub-relations)
    - Parallel execution of a single query by dividing it into a set of sub queries that operate on fragments
- Disadvantages of Fragmentation
  - Views that are defined on more than one fragment will require extra processing. Minimizing distributed joins is a fundamental fragmentation issue.
  - Semantic data control (especially integrity enforcement) more difficult - simpler task of checking for dependencies would result in chasing after data in a number of sites

Fragmentation Alternatives – Horizontal

$PROJ_1$: projects with budgets less than \$200,000

$PROJ_2$: 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 Alternatives – Vertical

$PROJ_1$: information about project budgets

$PROJ_2$: 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

$PROJ_1$

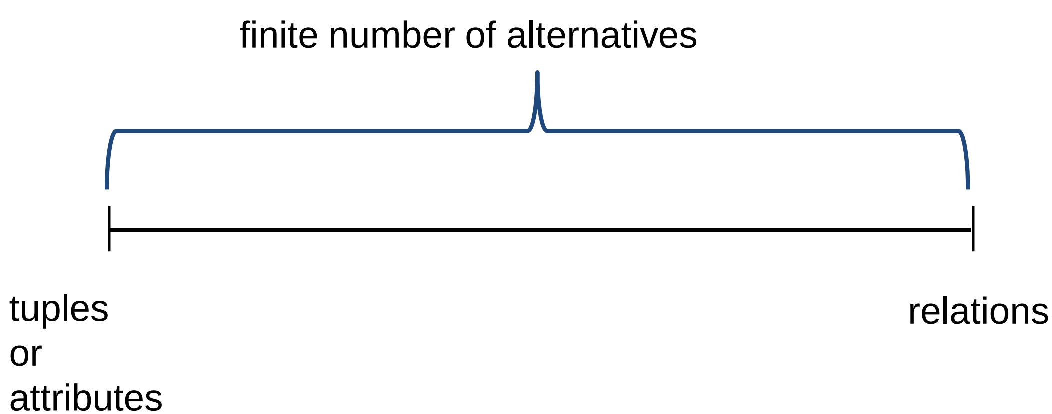
PNO	BUDGET
P1	150000
P2	135000
P3	250000
P4	310000
P5	500000

$PROJ_2$

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

Degree of Fragmentation

- The degree of fragmentation goes from one extreme that is, not to fragment at all, to the other extreme, to fragment to the level of individual tuples (in the case of horizontal fragmentation) or to the level of individual attributes (in the case of vertical fragmentation).



Need to find the suitable level of partitioning within this range

Correctness of Fragmentation

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

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

  - The reconstructability of the relation from its fragments ensures that constraints defined on the data in the form of dependencies are preserved.
- Disjointness
  - If relation R is decomposed into fragments R_1, R_2, ..., R_n, and data item d_i is in R_j, then d_i should not be in any other fragment R_k ($k \neq j$).

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.
- Non-replicated
  - partitioned : each fragment resides at only one site
- Replicated
  - fully replicated : each fragment at each site
  - partially replicated : each fragment at some of the sites
- Rule of thumb

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

otherwise replication may cause problems

Discussion: Comparison of Replication Alternatives

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

Information Requirements

- Four categories:
  - Database information
  - Application information
  - Communication network information
  - Computer system information
- The latter two categories are completely quantitative in nature and are used in allocation models rather than in fragmentation algorithms.

Fragmentation

- Horizontal Fragmentation (HF)
  - **Primary Horizontal Fragmentation** (PHF) - of a relation is performed using predicates that are defined on that relation.
  - **Derived Horizontal Fragmentation** (DHF) - is the partitioning of a relation that results from predicates being defined on another relation.
- Vertical Fragmentation (VF)
- Hybrid Fragmentation (HF)

Example Database

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

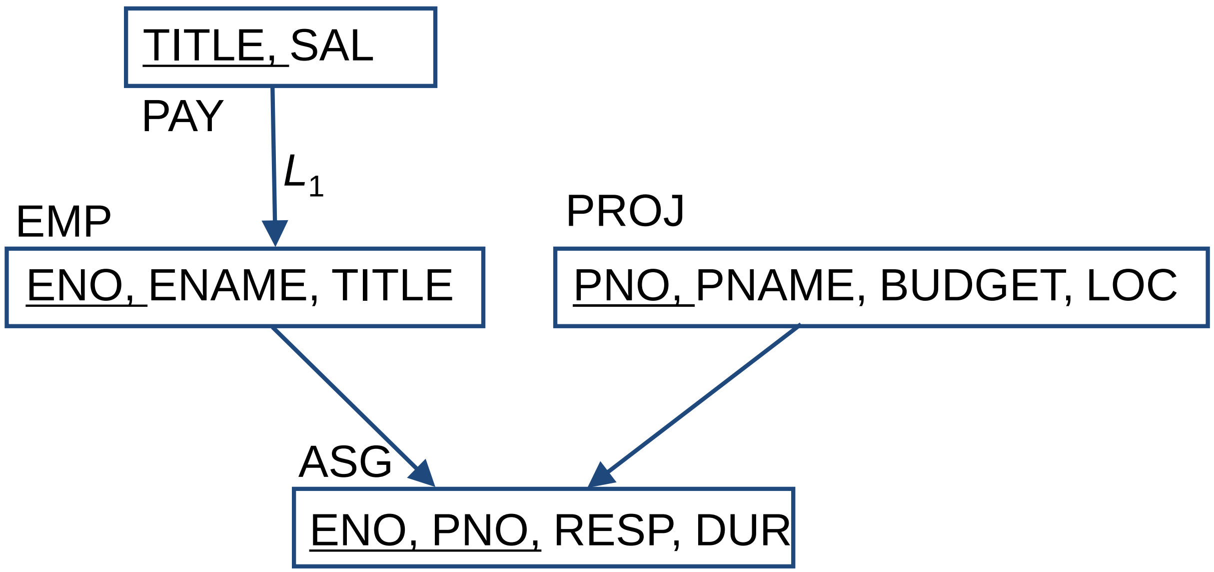
PAY

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

PHF – Information Requirements

Database Information

- concerns the global conceptual schema
- how the database relations are connected to one another, especially with joins.
- Expression of relationships among relations using links (Join graph)



PHF – Information Requirements

- Given link L_1, the owner and member functions have the following values:
  - owner(L_1) = PAY
  - member(L_1) = EMP

- The quantitative information required about the database is the cardinality of each relation R, denoted card(R).

PHF - Information Requirements

Application Information

- Qualitative information guides the fragmentation activity
- Consists of predicates used in use queries
- Analyze applications to determine “most important” predicates.
- **80/20 rule! The most active 20% of user queries account for 80% of the total data accesses.**

PHF - Information Requirements

- **Simple predicates** : Given $R[A_1, A_2, \ldots, A_n]$, a simple predicate p_j is

  $p_j : A_i \ \theta \ Value$

  where $\theta \in \{=,<,\leq,>,\geq,\neq\}$, $Value \in D_i$ and D_i is the domain of A_i.
- For relation R, we define set of all simple predicates

  $P_r = \{p_1, p_2, \ldots, p_m\}$
- Example :
  - PNAME = "Maintenance"
  - BUDGET ≤ 200000

PHF - Information Requirements

- **Minterm predicates** : **Conjunction of simple predicates**
- A *minterm* is a combination of predicates that covers all possible outcomes (true or false) for each predicate.

Given R and $P_r = \{p_1, p_2, \ldots, p_m\}$

Define set of minterm predicates as

$$M = \{m_1, m_2, \ldots, m_z\} \text{ as}$$
$$M = \{\, m_i \mid m_i = \wedge_{p_j \in Pr}\, p_j^* \,\},\ 1 \leq j \leq m,\ 1 \leq i \leq z$$

where $p_j^* = p_j$ or $p_j^* = \neg(p_j)$. (natural or negated form)

Example

- Consider relation PAY. Following are some of the possible simple predicates that can be defined on PAY.
  - p_1: TITLE = “Elect. Eng.”
  - p_2: TITLE = “Syst. Anal.”
  - p_3: TITLE = “Mech. Eng.”
  - p_4: TITLE = “Programmer”
  - p_5: SAL ≤ 30000
- Following are some of the minterm predicates that can be defined based on these simple predicates.
  - m_1: TITLE = “Elect. Eng.” ^ SAL ≤ 30000
  - m_2: TITLE = “Elect. Eng.” ^ SAL > 30000
  - m_3: ┐(TITLE = “Elect. Eng.”) ^ SAL ≤ 30000
  - m_4: ┐(TITLE = “Elect. Eng.”) ^ SAL > 30000
  - m_5: TITLE = “Programmer” ^ SAL ≤ 30000
  - m_6: TITLE = “Programmer” ^ SAL > 30000

PHF – Information Requirements

- Application Information in terms of quantitative information about user applications
  - **Minterm selectivities**: $sel(m_i)$
    - Number of tuples of the relation that would be accessed by a user query which is specified according to a given minterm predicate m_i.
  - **Access frequencies**: $acc(q_i)$
    - Frequency with which a user application q_i accesses data.
    - Access frequency for a minterm predicate can also be defined.

Primary Horizontal Fragmentation

- PHF is defined by a **selection operation on the owner relations** of a database schema.
- Definition :

$$R_j = \sigma_{F_j}(R), \;\; 1 \leq j \leq w$$

where F_j is a selection formula, which is (preferably) a minterm predicate.

- 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.

Set of horizontal fragments also referred to as minterm fragments.

Example

- Define horizontal fragments based on project location.

$$PROJ_1 = \sigma_{LOC=\text{“Montreal”}}(PROJ)$$
$$PROJ_2 = \sigma_{LOC=\text{“New York”}}(PROJ)$$
$$PROJ_3 = \sigma_{LOC=\text{“Paris”}}(PROJ)$$

$PROJ_1$

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

PHF – Algorithm

Given: A relation *R,* the set of simple predicates P_r

Output: The set of fragments of $R = \{R_1, R_2, \ldots, R_w\}$ which obey the fragmentation rules.

Preliminaries :

1. P_r should be *complete*
2. P_r should be *minimal*

Completeness of Simple Predicates

- A set of simple predicates P_r is said to be *complete* if and only if
  - There is an equal probability of access by every application to any tuple belonging to any minterm fragment that is defined according to P_r.

- Example : Assume PROJ[PNO,PNAME,BUDGET,LOC] has two applications defined on it.
  - Find the budgets of projects at each location. (1)
  - Find projects with budgets less than $200000. (2)

Completeness of Simple Predicates

According to (1),

P_r = {LOC=“Montreal”,LOC=“New York”,LOC=“Paris”}

is complete, however with respect to (2), it is not complete.

- If the only application that accesses PROJ wants to access the tuples according to the location, the set is complete since each tuple of each fragment $PROJ_i$ has the same probability of being accessed
- According to second application, some of the tuples within each $PROJ_i$ have a higher probability of being accessed.

Modify

P_r = {LOC=“Montreal”,LOC=“New York”,LOC=“Paris”, BUDGET≤200000,BUDGET>200000}

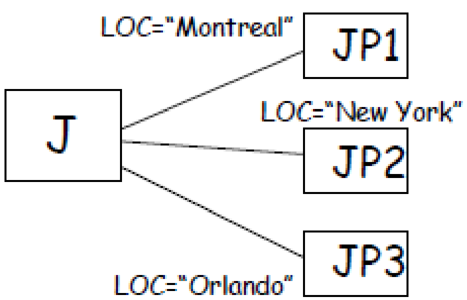
which is complete.

Explanation

$$JP_1 = \sigma_{LOC = "MONTREAL"}(J)$$

$$JP_2 = \sigma_{LOC = "NewYork"}(J)$$

$$JP_3 = \sigma_{LOC = "Orlando"}(J)$$



JP1

JNO	JNAME	BUDGET	LOC
J1	Instrumental	150,000	Montreal

JP2

JNO	JNAME	BUDGET	LOC
J2	GUI	135,000	New York
J3	CAD/CAM	250,000	New York

JP3

JNO	JNAME	BUDGET	LOC
J4	Database Dev.	310,000	Orlando

Tuple “J2” has higher access probability than tuple “J3” in JP2 (2 applications access J2, but one application access J3)

Minimality of Simple Predicates

- If a predicate influences how fragmentation is performed, (i.e., causes a **fragment *f* to be further fragmented into, say, f_i and f_j**) then there should be **at least one application that accesses f_i and f_j differently.**
- In other words, the simple predicate should be *relevant* in determining a fragmentation.
- If all the predicates of a set P_r are relevant, then P_r is *minimal*.

Minimality of Simple Predicates

Example :

P_r ={LOC="Montreal",LOC="New York", LOC="Paris",

BUDGET≤200000,BUDGET>200000}

is minimal (in addition to being complete).

However, if we add

PNAME = "Instrumentation"

then P_r is not minimal.

- New predicate is not relevant with respect to P_r.
- There is no application that would access the resulting fragments any differently.

COM_MIN (COMputation of MINterm predicates) Algorithm

Given: a relation *R* and a set of simple predicates P_r

Output: a *complete* and *minimal* set of simple predicates P_r' for P_r

Rule 1: Each fragment is accessed differently by at least one application.

f_i *of* P_r': fragment f_i defined according to a minterm predicate defined over the predicates of P_r'.

COM_MIN Algorithm

Input: R: relation; Pr: set of simple predicates
Output: Pr': set of simple predicates
Declare: F: set of minterm fragments
begin

find $p_i \in Pr$ such that p_i partitions R according to *Rule* 1 ;
$Pr' \leftarrow p_i$;
$Pr \leftarrow Pr - p_i$;
$F \leftarrow f_i$ $\{f_i$ is the minterm fragment according to $p_i\}$;
repeat

find a $p_j \in Pr$ such that p_j partitions some f_k of Pr' according to *Rule* 1 ;
$Pr' \leftarrow Pr' \cup p_j$;
$Pr \leftarrow Pr - p_j$;
$F \leftarrow F \cup f_j$;
if $\exists p_k \in Pr'$ *which is not relevant* **then**

$Pr' \leftarrow Pr' - p_k$;
$F \leftarrow F - f_k$;

until Pr' *is complete* ;

end

COM_MIN algorithm

- First Step: Initialization
  - Algorithm begins by finding a predicate that is relevant and partitions the input relation. The repeat-until loop iteratively adds predicates to this set, ensuring minimality at each step. Perform this iteratively till set P_r' is both minimal and complete
- Second Step:
  - Derive set of minterm predicates that can be defined on predicates in set P_r'
  - Difficulty is that the set of minterm predicates can be quite large!
- Third Step:
  - There is a **need to reduce the number of minterm predicates** by **eliminating** of some of minterm fragments that may be meaningless.
  - This elimination is performed by identifying minterms that might be **contradictory to a set of implications *I*.**

Reducing minterm predicates

This reduction can be achieved by eliminating some of the minterm fragments that may be meaningless. This elimination is performed by identifying those minterms that might be contradictory to a set of implications I. For example, if $Pr' = \{p_1, p_2\}$, where

$$p_1 : att = value\_1$$
$$p_2 : att = value\_2$$

and the domain of att is $\{value\_1, value\_2\}$, it is obvious that I contains two implications:

$$i_1 : (att = value\_1) \Rightarrow \neg(att = value\_2)$$
$$i_2 : \neg(att = value_1) \Rightarrow (att = value\_2)$$

The following four minterm predicates are defined according to Pr':

$$m_1 : (att = value\_1) \wedge (att = value\_2)$$
$$m_2 : (att = value\_1) \wedge \neg(att = value\_2)$$
$$m_3 : \neg(att = value\_1) \wedge (att = value\_2)$$
$$m_4 : \neg(att = value\_1) \wedge \neg(att = value\_2)$$

In this case the minterm predicates m_1 and m_4 are contradictory to the implications I and can therefore be eliminated from M.

PHORIZONTAL Algorithm

Makes use of COM_MIN to perform fragmentation.

Input: a relation R and a set of simple predicates P_r

Output: a set of minterm predicates M according to which relation R is to be fragmented

PHORIZONTAL Algorithm

Input: R: relation; Pr: set of simple predicates
Output: M: set of minterm fragments
begin

$Pr' \leftarrow$ COM_MIN(R, Pr) ;
determine the set M of minterm predicates ;
determine the set I of implications among $p_i \in Pr'$;
foreach $m_i \in M$ **do**
  if *m_i is contradictory according to I* **then**
    $M \leftarrow M - m_i$

end

PHF – Example

Two candidate relations : PAY and PROJ.

Fragmentation of relation PAY

- Single application: Check the salary info and determine raise.
- Employee records kept at two sites $\Rightarrow$ application run at two sites
- Simple predicates to partition relation PAY are

  p_1 : SAL $\leq$ 30000

  p_2 : SAL > 30000

  $P_r = \{p_1, p_2\}$

  $P_r' = \{p_1\}$ This is complete and minimal since p_2 would not partition f_1 (which is the minterm fragment formed with respect to p_1) according to Rule 1.

- Minterm predicates

  m_1 : (SAL $\leq$ 30000)

  m_2 : **NOT**(SAL $\leq$ 30000) $=$ (SAL > 30000)

PHF – Example

- Therefore, we define two fragments $F_s = \{S_1, S_2\}$ according to M

PAY_1

TITLE	SAL
Mech. Eng.	27000
Programmer	24000

PAY_2

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

PHF – Example

Fragmentation of relation PROJ

- Applications:
  - Find the name and budget of projects given their location.
    - Issued at three sites
  - Access project information according to budget
    - one site accesses ≤200000, other accesses >200000

- Simple predicates
- For application (1)

  p_1 : LOC = “Montreal”

  p_2 : LOC = “New York”

  p_3 : LOC = “Paris”
- For application (2)

  p_4 : BUDGET ≤ 200000

  p_5 : BUDGET > 200000

- $P_r = P_r' = \{p_1,p_2,p_3,p_4,p_5\}$

PHF – Example

- Minterm fragments left after elimination of obvious implications
  - m_1 : (LOC = "Montreal") $\wedge$ (BUDGET $\leq$ 200000)
  - m_2 : (LOC = "Montreal") $\wedge$ (BUDGET > 200000)
  - m_3 : (LOC = "New York") $\wedge$ (BUDGET $\leq$ 200000)
  - m_4 : (LOC = "New York") $\wedge$ (BUDGET > 200000)
  - m_5 : (LOC = "Paris") $\wedge$ (BUDGET $\leq$ 200000)
  - m_6 : (LOC = "Paris") $\wedge$ (BUDGET > 200000)

Implications help eliminate minterm predicates

$$i_1 : p_1 \Rightarrow \neg p_2 \wedge \neg p_3$$

$$i_2 : p_2 \Rightarrow \neg p_1 \wedge \neg p_3$$

$$i_3 : p_3 \Rightarrow \neg p_1 \wedge \neg p_2$$

$$i_4 : p_4 \Rightarrow \neg p_5$$

$$i_5 : p_5 \Rightarrow \neg p_4$$

$$i_6 : \neg p_4 \Rightarrow p_5$$

$$i_7 : \neg p_5 \Rightarrow p_4$$

PHF – Example

innovate achieve lead

$PROJ_2$ and $PROJ_5$ are empty

$PROJ_1$

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal

$PROJ_3$

PNO	PNAME	BUDGET	LOC
P2	Database Develop.	135000	New York

$PROJ_4$

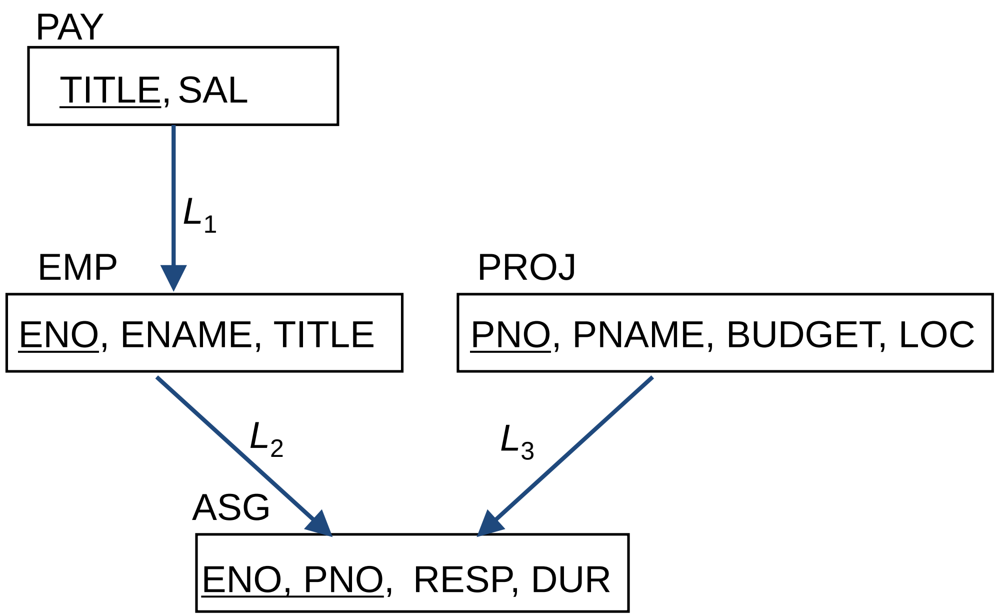
PNO	PNAME	BUDGET	LOC
P3	CAD/CAM	250000	New York

$PROJ_6$

PNO	PNAME	BUDGET	LOC
P4	Maintenance	310000	Paris

Derived Horizontal Fragmentation

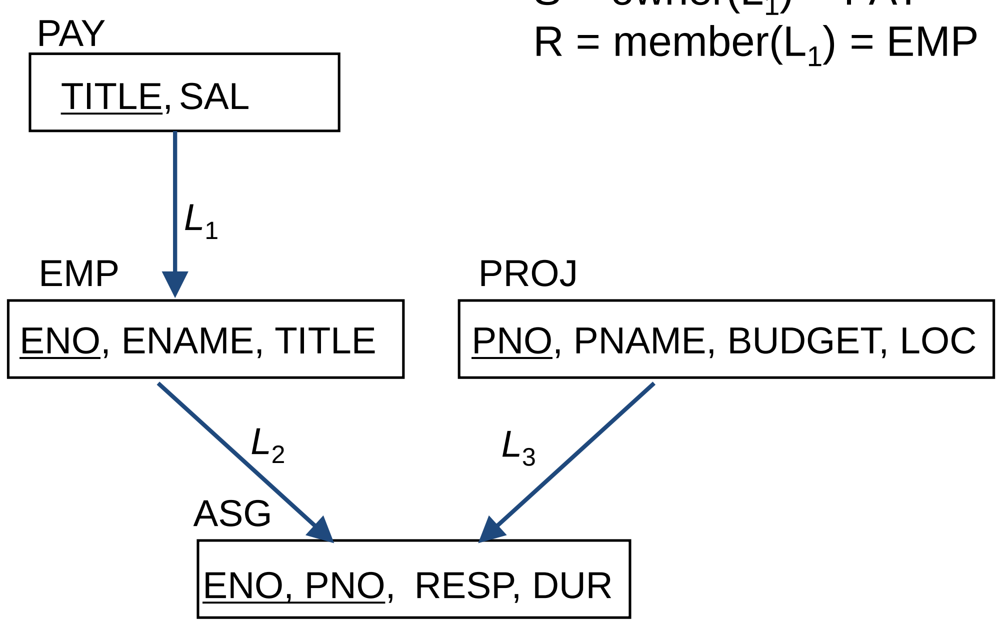
- Defined on a **member relation** of a link according to a selection operation specified on its owner.
  - Each link is defined as an equijoin.
  - Equijoin can be implemented by means of semijoins.



Derived Horizontal Fragmentation

- Defined on a **member relation** of a link according to a selection operation specified on its owner.
  - Each link is an equijoin.
  - Equijoin can be implemented by means of semijoins.

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



- Partition a member relation according to the fragmentation of its owner,
- But the resulting fragment should be defined only on the attributes of the member relation.

DHF – Definition

Given a link L where *owner* (L) = S and *member* (L) = R, the derived horizontal fragments of R are defined as

$$R_i = R \ltimes_F S_i,\ 1 \leq i \leq w$$

where w is the maximum number of fragments that will be defined on R and

$$S_i = \sigma_{F_i}(S)$$

where F_i is the formula according to which the primary horizontal fragment S_i is defined.

DHF – Example

- Given link L_1 where owner (L_1) = PAY and member (L_1) = EMP
- Group engineers into two groups according to their salary: those making less than or equal to \$30,000, and those making more than \$30,000. Two fragments EMP_1 and EMP_2 are defined as follows:

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

where (set of partitions of the owner relation)

$$PAY_1 = \sigma_{SAL \leq 30000}(PAY)$$
$$PAY_2 = \sigma_{SAL > 30000}(PAY)$$

- Set of partitions of the owner relation (e.g., PAY1 and PAY2),
- Member relation,
- Set of semijoin predicates between the owner and the member (e.g., EMP.TITLE = PAY.TITLE)

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.

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.
- Has been studied within the centralized context – mainly as a design tool
  - Allowing user queries to deal with smaller relations, thus causing a smaller number of page accesses
  - Most "active" subrelations can be identified and placed in a faster memory subsystem
- More difficult than horizontal, because more alternatives exist.

Vertical Fragmentation

- Two heuristic approaches:
  - **Grouping** - starts by assigning each attribute to one fragment, and at each step, joins some of the fragments until some criteria is satisfied
  - **Splitting -** starts with a relation and decides on beneficial partitioning based on the access behavior of applications to the attributes
  - We discuss splitting technique since it fits naturally with the top down design methodology.

VF – Information Requirements

- Application Information- Vertical partitioning 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**". This measure is the ***affinity*** of attributes, which indicates how closely related the attributes are.
  - Attribute affinities
    - a measure that indicates how closely related the attributes are – notion of "togetherness"
  - Attribute usage values
    - Given a set of queries $Q = \{q_1, q_2, \ldots, q_q\}$ that will run on the relation $R[A_1, A_2, \ldots, A_n]$,
    - $use(q_i, \bullet)$ can be defined for each application if the designer knows the applications that will run on the DB.

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

VF – Definition 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

Let A_1= PNO, A_2= PNAME, A_3= BUDGET, A_4= LOC

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

- Attribute usage values are not sufficiently general to form the basis of attribute splitting and fragmentation.
- This is because these values do not represent the weight of application frequencies.
- The frequency measure can be included in the definition of the attribute affinity measure $aff(A_i,A_j)$, which measures the bond between two attributes of a relation according to how they are accessed by applications.

VF – Affinity Measure *aff*(A_i,A_j)

The attribute affinity measure between two attributes A_i and A_j of a relation $R[A_1, A_2, \ldots, A_n]$ with respect to the set of applications $Q = (q_1, q_2, \ldots, q_q)$ is defined as follows :

$$aff(A_i, A_j) = \sum_{k|use(q_k,A_i)=1 \wedge use(q_k,A_j)=1} \sum_{\forall S_l} ref_l(q_k) acc_l(q_k)$$

where $ref_l(q_k)$ is the number of accesses to attributes (A_i, A_j) for each execution of application q_k at site S_l and $acc_l(q_k)$ is the application access frequency measure previously defined and modified to include frequencies at different sites.

VF – Calculation of $aff(A_i, A_j)$

- Assume each query in the previous example accesses the attributes once during each execution.
- Also assume the access frequencies

$$\begin{array}{c} \\ q_1 \\ q_2 \\ q_3 \\ q_4 \end{array} \begin{array}{c} \begin{array}{ccc} S_1 & S_2 & S_3 \end{array} \\ \begin{bmatrix} 15 & 20 & 10 \\ 5 & 0 & 0 \\ 25 & 25 & 25 \\ 3 & 0 & 0 \end{bmatrix} \end{array}$$

- Then
  $aff(A_1, A_3) = 15*1 + 20*1+10*1$
  $= 45$
- The attribute affinity matrix AA is

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

VF – Clustering Algorithm

- Fundamental task in designing a vertical fragmentation algorithm
  - find some means of grouping the attributes of a relation based on the attribute affinity values in AA.

- Bond energy algorithm takes as input the attribute affinity matrix, permutes its rows and columns, and generates a **clustered affinity matrix (CA).**

VF – Clustering Algorithm

- 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.
- BEA finds an ordering of attributes such that the **global affinity measure** is maximized.

$$AM = \sum_{i} \sum_{j} \text{(affinity of } A_i \text{ and } A_j \text{ with their neighbors)}$$

VF – Clustering Algorithm

- Input: The *AA* matrix
- Output: The clustered affinity matrix *CA* which is a permutation of *AA*
  - *Initialization*: Place and fix one of the columns of *AA* in *CA*.
  - *Iteration*: Place the remaining *n-i* columns (where i is the number of columns already placed in CA) in the remaining *i*+1 positions in the *CA* matrix. For each column, choose the placement that makes the most contribution to the global affinity measure. Continue this step until no more columns remain to be placed.
  - *Row order*: Once the column ordering is determined, the placement of the rows should also be changed so that their relative positions match the relative positions of the columns.

VF – Clustering Algorithm

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

Let us define the *bond* between two attributes A_x and A_y as

$$bond(A_x, A_y) = \sum_{z=1}^{n} aff(A_z, A_x) aff(A_z, A_y)$$

Then AM can be written as

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

VF – Clustering Algorithm

- "Best" placement of an attribute?
- Need to define what is meant by contribution of an attribute to the global affinity measure.
- If we consider placing a new attribute A_k between attributes A_i and A_j in the clustered affinity matrix, the net contribution to the global affinity measure of placing attribute A_k between A_i and A_j is given by

$$cont(A_i, A_k, A_j) = 2bond(A_i, A_k)+2bond(A_k, A_l) -2bond(A_i, A_j)$$

where $$bond(A_x,A_y) = \sum_{z=1}^{n} aff(A_z,A_x)aff(A_z,A_y)$$

Example

- Study the contribution of moving attribute A_4 between attributes A_1 and A_2, given by the formula

  $cont(A_1,A_4,A_2) = 2bond(A_1,A_4) + 2bond(A_4,A_2) - 2bond(A_1,A_2)$

  Computing each term, we get

  $bond(A_1,A_4) = 45*0+0*75+45*3+0*78 = 135$

  $bond(A_4,A_2) = 11865$

  $bond(A_1,A_2) = 225$

$$\begin{array}{c|cccc|} & A_1 & A_2 & A_3 & A_4 \\ \hline A_1 & - & 0 & 45 & 0 \\ A_2 & 0 & - & 5 & 75 \\ A_3 & 45 & 5 & - & 3 \\ A_4 & 0 & 75 & 3 & - \end{array}$$

  Therefore,

  $cont(A_1,A_4,A_2) = 2*135+2*11865-2*225 = 23550$

- Note that the calculation of the bond between two attributes requires the multiplication of the respective elements of the two columns representing these attributes and taking the row-wise sum.

BEA – Example

Consider the following *AA* matrix and the corresponding *CA* matrix where A_1 and A_2 have been placed. Place A_3:

$$\begin{array}{c} \\ A_1 \\ A_2 \\ A_3 \\ A_4 \end{array} \begin{array}{c} \begin{array}{cc} A_1 & A_2 \end{array} \\ \left[\begin{array}{cc} 45 & 0 \\ 0 & 80 \\ 45 & 5 \\ 0 & 75 \end{array}\right] \end{array} \qquad \begin{array}{c} \\ A_1 \\ A_2 \\ A_3 \\ A_4 \end{array} \begin{array}{c} \begin{array}{ccc} A_1 & A_3 & A_2 \end{array} \\ \left[\begin{array}{ccc} 45 & 45 & 0 \\ 0 & 5 & 80 \\ 45 & 53 & 5 \\ 0 & 3 & 75 \end{array}\right] \end{array}$$

Ordering (0-3-1) :

$$\begin{aligned} cont(A_0,A_3,A_1) &= 2bond(A_0\ ,\ A_3) + 2bond(A_3\ ,\ A_1) - 2bond(A_0\ ,\ A_1) \\ &= 2*\ 0 + 2*\ 4410 - 2*0 = 8820 \end{aligned}$$

Ordering (1-3-2) :

$$\begin{aligned} cont(A_1,A_3,A_2) &= 2bond(A_1\ ,\ A_3) + 2bond(A_3\ ,\ A_2) - 2bond(A_1,A_2) \\ &= 2*\ 4410 + 2*\ 890 - 2*225 = 10150 \end{aligned}$$

Ordering (2-3-4) :

$$cont\ (A_2,A_3,A_4) = 1780$$

BEA Example

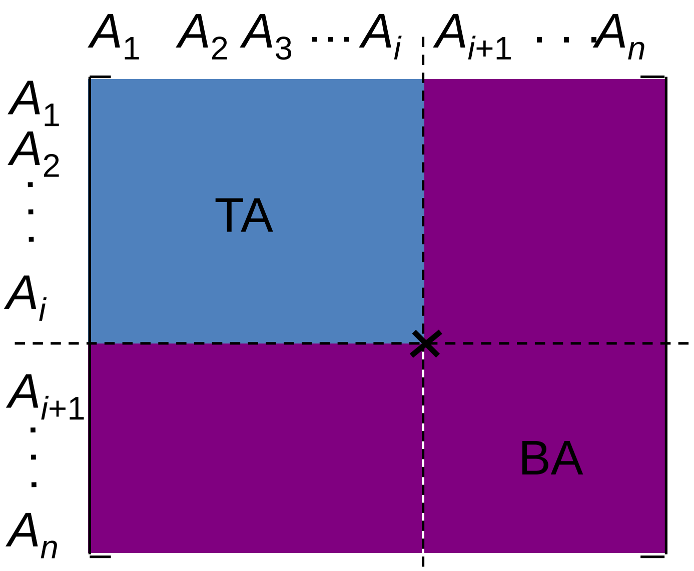
- Since the contribution of the ordering (1-3-2) is the largest, we select to place A_3 to the right of A_1.
- Similar calculations for A_4 indicate that it should be placed to the right of A_2
- Finally, the rows are organized in the same order as the columns and the result is shown as follows

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

VF – Partitioning Algorithm

- How can you divide a set of clustered attributes $\{A_1, A_2, \ldots, A_n\}$ into two (or more) sets $\{A_1, A_2, \ldots, A_i\}$ and $\{A_{i+1}, \ldots, A_n\}$ such that there are no (or minimal) applications that access both (or more than one) of the sets.

- So, we need to locate a splitting point in the clustered attribute matrix



VF – Algorithm

Define

TQ = set of applications that access only *TA attribute set*

BQ = set of applications that access only *BA attribute set*

OQ = set of applications that access both *TA* and *BA attribute set*

and

CTQ = total number of accesses to attributes by applications that access only *TA*

CBQ = total number of accesses to attributes by applications that access only *BA*

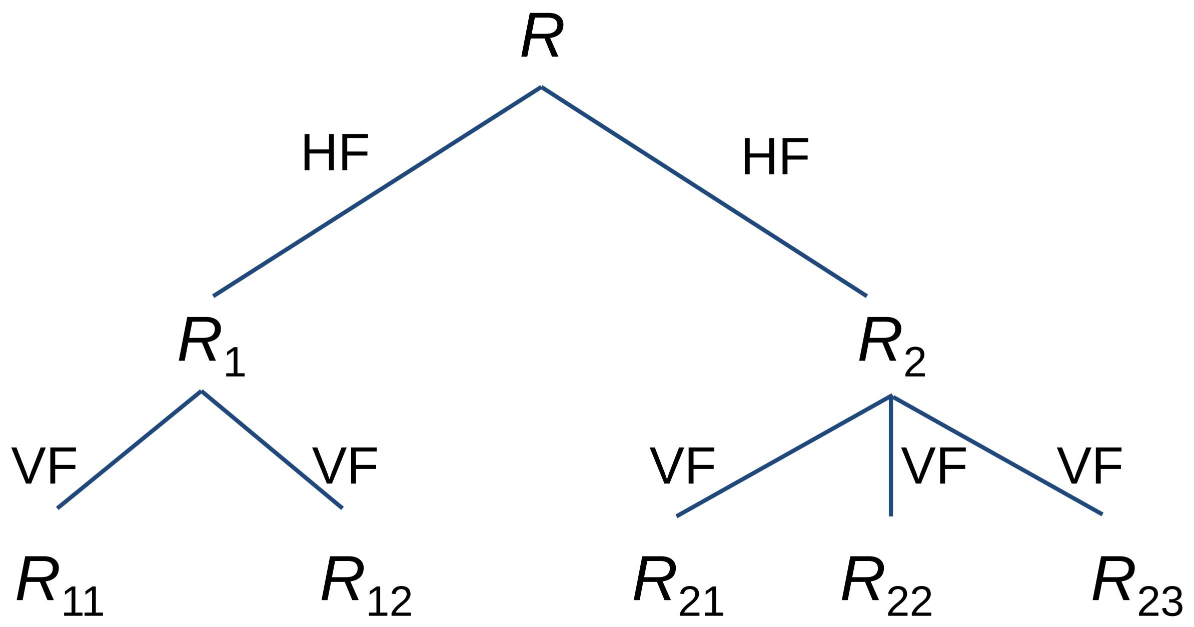
COQ = total number of accesses to attributes by applications that access both *TA* and *BA*

- The best position for division is one which produces the sets *TQ* and *BQ* such that the total accesses to *only one* fragment are maximized, while the total accesses to *both* fragments are minimized.
- The optimization problem is defined as finding the point such that the expression

$$z = CTQ * CBQ - COQ^2$$

Hybrid Fragmentation

- Vertical fragmentation may be followed by a horizontal one, or vice versa, producing a tree- structured partitioning



Allocation Problem

Problem Statement

Given

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

Find the "optimal" distribution of F to S.

Optimality can be defined wrt 2 measures

- Minimal cost
  - cost of storing each F_i at a site S_j, the cost of querying F_i at site S_j, the cost of updating F_i at all sites where it is stored, and the cost of data communication
- Performance

  Minimize Response time and/or maximize throughput

"Optimality" measure should include both the performance and the cost factors.

Information Requirements

- Allocation stage needs the quantitative data about the database
  - the applications that run on it,
  - the communication network,
  - the processing capabilities,
  - and storage limitations of each site on the network.

Information Requirements

Database information

- selectivity of fragments (number of tuples of F_j that need to be accessed in order to process q_i)
- size of a fragment

Application information

- number of accesses of a query to a fragment

$$u_{ij} = \begin{cases} 1 \text{ if query } q_i \text{ updates fragment } F_j \\ 0 \text{ otherwise} \end{cases}$$

$$r_{ij} = \begin{cases} 1 \text{ if query } q_i \text{ retrieves from fragment } F_j \\ 0 \text{ otherwise} \end{cases}$$

- originating site of each query

Site information (storage and processing capacity)

- unit cost of storing data at a site
- unit cost of processing at a site

Network information

- communication cost per frame between sites
- frame size

Allocation Model

- General Form

min(Total Cost)

subject to

response time constraint

storage constraint

processing constraint

- Decision Variable

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

Allocation Model

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

- Total Cost

$$\sum_{\text{all queries}} \text{query processing cost} + \sum_{\text{all sites}} \sum_{\text{all fragments}} \text{cost of storing a fragment at a site}$$

- Storage Cost (of fragment F_j at S_k)
  - (unit storage cost at S_k) $*$ (size of F_j) $* \; x_{jk}$

- Query Processing Cost (for one query)
  - processing component + transmission component

Allocation Model

Query Processing Cost

– Processing component

access cost + integrity enforcement cost + concurrency control cost

– Access cost

$$\sum_{\text{all sites}} \sum_{\text{all fragments}} (\text{no. of update accesses+ no. of read accesses}) * x_{ij} * \text{local processing cost at a site}$$

– Integrity enforcement and concurrency control costs

- Can be similarly calculated

Allocation Model

- Query Processing Cost
  - Transmission component

    cost of processing updates + cost of processing retrievals
  - Cost of updates

$$\sum_{\text{all sites}} \sum_{\text{all fragments}} \text{update message cost} + \sum_{\text{all sites}} \sum_{\text{all fragments}} \text{acknowledgment cost}$$

  - Retrieval Cost

$$\sum_{\text{all fragments}} \min_{\text{all sites}} (\text{cost of retrieval command} + \text{cost of sending back the result})$$

Allocation Model

Constraints

– Response Time

execution time of query $\leq$ max. allowable response time for that query

– Storage Constraint (for a site)

$$\sum_{\text{all fragments}} \text{storage requirement of a fragment at that site} \leq \text{storage capacity at that site}$$

– Processing constraint (for a site)

$$\sum_{\text{all queries}} \text{processing load of a query at that site} \leq \text{processing capacity of that site}$$

Allocation Model

Developed a generic allocation model

- Fairly complex and heuristics need to be used to provide suboptimal solutions

Lecture Summary

- Distributed database design
  - Fragmentation
  - Allocation

Essential Readings

- Chapter 5 Tamer Ozsu

Thanks…

Next Lecture

- Overview of Query Processing

Questions??