Design of Distributed Databases

General goals of the DDBMS design:

- to provide high performance
- to provide reliability
- to provide functionality
- to fit into the existing environment
- to provide cost-saving solutions

Importance of design

Reasons of poor efficiency:

- Hardware: 10%
- DBMS: 15%
- Database design: 25%
- Applications design: 55%

Costs of improvement :

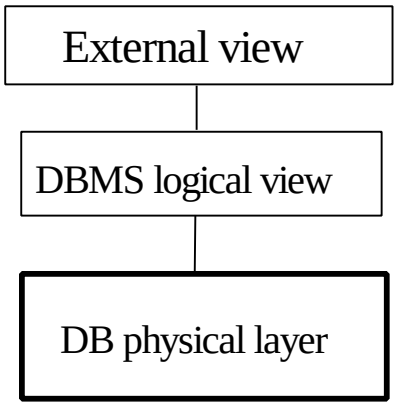
- Hardware: 10%
- DBMS: 20%
- Database design: 40%
- Applications design: 30%

The design phase usually requires the most time within the development process.

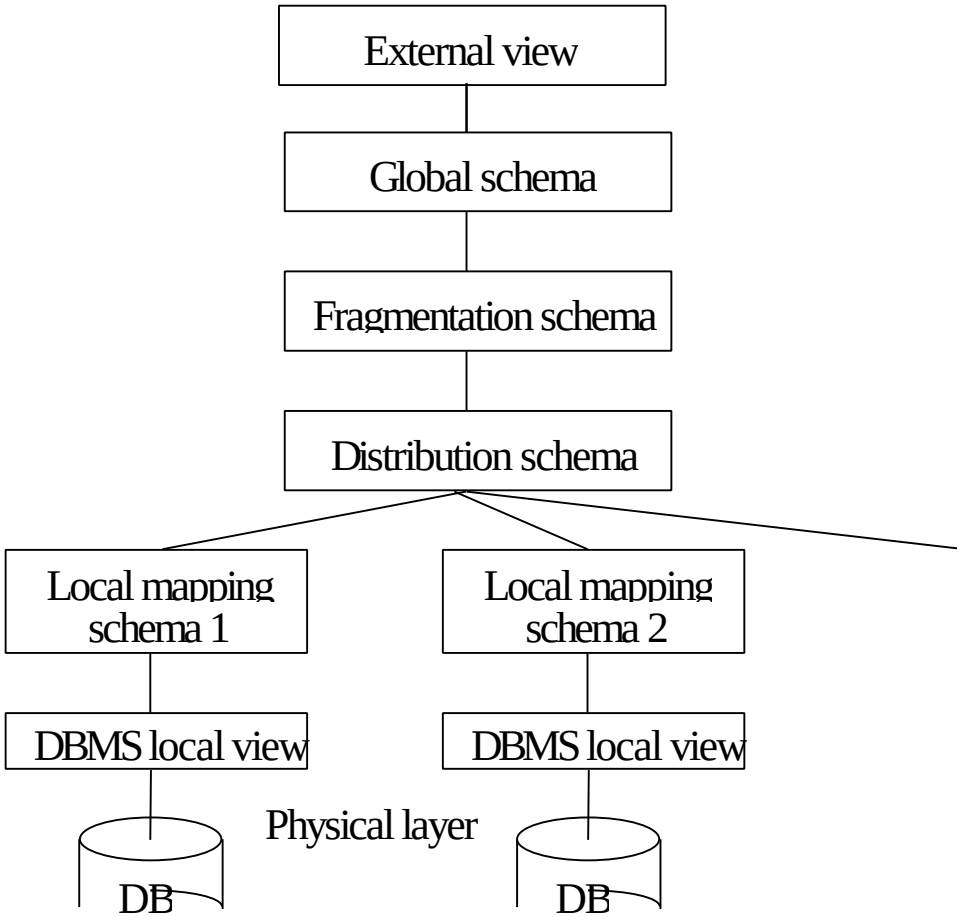
- 3-5 code lines per work hour

Logical Database Structure Models

Non-distributed databases



Distributed databases



Steps of Distributed Database Design

There are in general several design alternatives.

Top-down approach: first the general concepts, the global framework are defined, after then the details.
Down-top approach: first the detail modules are defined, after then the global framework.

If the system is built up from a scratch, the top-down method is more accepted.
If the system should match to existing systems or some modules are yet ready, the down-top method is usually used.

General design steps according to the structure:

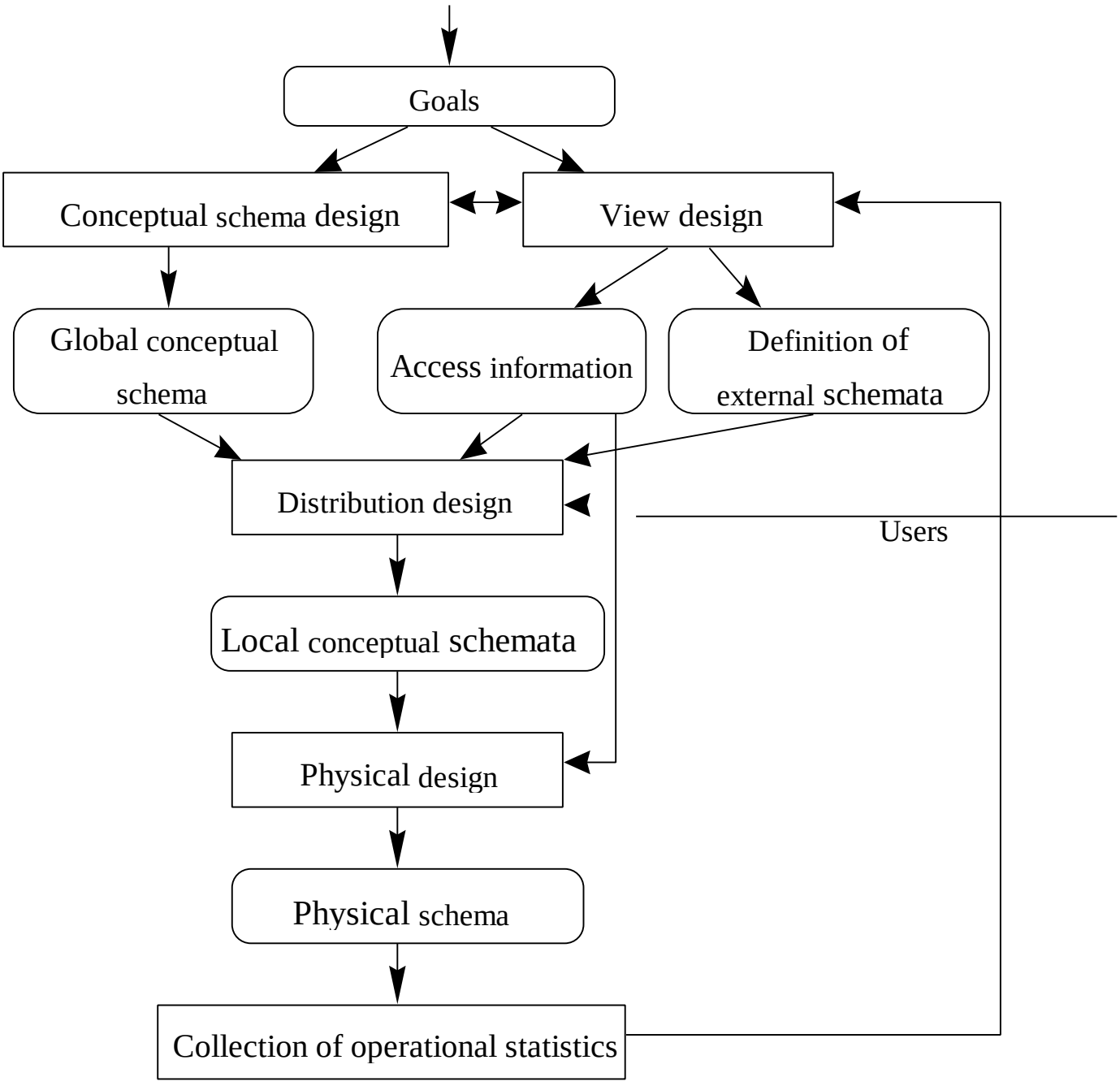
- analysis of the external, application requirements
- design of the global schema
- design of the fragmentation
- design of the distribution schema
- design of the local schemes
- design of the local physical layers

DDBMS -specific design steps:

- design of the fragmentation
- design of the distribution schema

During the requirement analysis phase, also the fragmentation and distribution requirements are considered.

Top-down Database Design



Down-Top Design

Usually existing and heterogeneous databases are integrated into a common distributed system.

Steps of integration:

- *Common data model selection*
As the different component databases may have different data models and the DDBMS should based on a single, common data model, the first step is to convert the different models into a common model.
The common model is usually an intermediate model, different from the data model of the components

- *Translation of each local schema into the common model*
The different schema element descriptions should be converted into this common model.

- *Integration of the local schema into a common global schema*
Beside the collection of the component descriptions, the integration should deal with the matching of the different semantic elements and with the resolving of the different types of inconsistency.

- *Design the translation between the global and local schemes*
To access all of the components on a homogeneous way, a conversion procedure should be applied.

Goals of the Fragmentation and Distribution Design

Local processing
It is desirable to perform as much tasks as possible at the local level, i.e. without any access to the other sites. The local processing provides an easier management and a more efficient execution.
Although a complete locality is an aim from the performance point of view, it can not be realized due to the distribution requirements of the system.

Availability and reliability of the DDB
It is desirable to distribute the data over different sites to provide higher availability. Thus, in the case of site failures a site can replace the other, no service or functionality will be lost.
The replication, distribution is a useful method to perform a recovery if the data on some site would be destroyed.

Distribution of processing load
It is desirable to distribute the processing power over the different sites to provide a higher throughput. The different processing steps of a complex task will be distributed among several sites enabling a parallel processing too.

Storage cost reduction
It may be cost effective if not every site is equipped with the same high performance and costly elements. It is enough to install only some of such specialized sites, the others can use it in a shared way. *We should find the trade-off of these requirements*

Design of the Fragmentation

The purpose of this phase is to determine the non-overlapping pieces, fragments of the global database which can be stored as a unit on different sites.

The data elements having the same properties, behavior are assigned to the same fragment.

Main aspects of the fragmentation:

Granularity:
The granularity determines at which level of database storage can be the fragmentation performed. If it is too low (field) then it needs a lot of management cost. If it is too rough (user level) then the unnecessary elements should be replicated causing a higher cost.

Fragmentation strategy
- horizontal fragmentation:. In this case the granularity is at the tuple level, and the attribute values of the tuple determine the corresponding fragment. The relation is partitioned horizontally into fragments.
- vertical fragmentation: the assignment of the data elements in a table is based on the schema position of the data. In this case the different projections are the content of the fragments.
- mixed fragmentation: the data are fragmented by both the vertical as the horizontal method.

Horizontal Fragmentation

The relation is partitioned horizontally into fragments..

Primary fragmentation:
the assignment of the tuple depends on the attribute values of the tuple
Derived fragmentation:
the assignment of the tuple depends not on the attributes of this tuple, but on the attributes of another tuple(s).

The fragmentation is described by an expression which value for every tuple determines the corresponding fragment.

Basic terms of fragmentation expression:

- *Simple predicate*:
  attribute = value
- *Minimal term predicate*:
  if P is a set of simple predicates
  $y = \wedge_{p \in P} p^*$
  where $p^* = P$ or $\neg p$ and $y \neq$ false
- *Fragment* :
  the set of tuples for which a minimal term predicate is true

A predicate is *relevant* if it contributes to the distinction of fragments with different access patterns

Horizontal Fragmentation

How to build the P set of predicates?

First we should analyze the access schemes of the tuples. Then we determine the groups of tuples having different references by at least one application. Analyzing the attribute values within the groups we have to fine a set of simple predicates for every fragment groups.

A P set of predicates can be used to describe the fragmentation if P is complete and minimal.

P is *complete* if and only if any two tuples belonging to the same fragment are referenced with the same probability by any application.
P is *minimal* if all its predicates are relevant.

Fragmentation steps:

1. initialization , P = ∅
2. take a p predicate so that there exist at least one application that references the tuples meeting p differently as the tuples not meeting p.
3. extend P with a new p predicate similar to the step 2.
4. eliminate non-relevant predicates
5. if P is not complete go to step 3
6. if P is complete, P is the result

Sometimes the cross-product of the application level criteria are used as global criteria.

Horizontal Fragmentation

Example 1

The sample table to be fragmented:

Employee(Name, Department, Skill, Salary)

The applications requiring access to the Employee table:

Application 1: it will access the employees in the department with Department id = 1.

Application 2: it will access the programmers independently from their departments

The relevant simple predicates

- for application 1:

Department = 1

- for application 2:

Skill = ‘Programmer’

Predication set:

P = { Department = 1, Skill = ‘Programmer’}

The minterm predicates:

Department = 1 AND Skill = ‘Programmer’

Department = 1 AND Skill <> ‘Programmer’

Department <> 1 AND Skill = ‘Programmer’

Department <> 1 AND Skill <> ‘Programmer’

A not relevant predicate:

Salary > 250000

Horizontal Fragmentation

Example 2

The sample table to be fragmented:

Employee(Name, Area, Skill, Salary)

Department(DeptId, Name, Area)

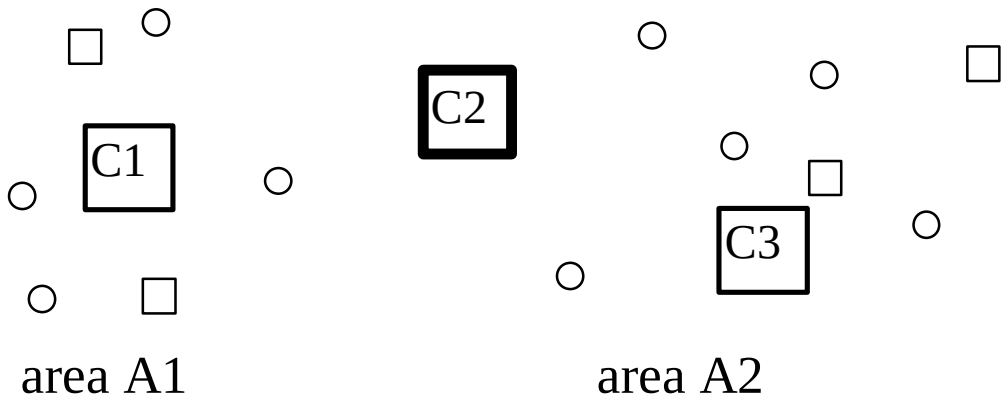
There are two areas (A1,A2) and three application centers (C1,C2,C3).

The applications requiring access to the tables:

Application 1: it will access the employees The likelihood of accesses: from C1 : A1:80%, A2:20%; from C2 : A1:50%, A2:50%; from C3 : A1:20%, A2: 80%

Application 2: it will access departments with the following probabilities: from C1: A1:100%, from C3:A2:100%

Application 3: it will access departments with the following probabilities: from C1: DeptId<100:100%, from C2: Deptid between 100 and 200: 100%, from C3: DeptId>200:100%



Horizontal Fragmentation

Example 2

The simple predicates for Employee

Area = A1
Area = A2

The simple predicates for Department:

Area = A1
Area = A2
DeptId < 100
DeptId between 100 and 200
DeptId > 200

We can reduce the minterm predicate set based on the dependencies:

Area <> A2 $\Leftrightarrow$ Area = A1
DeptId < 100 $\Rightarrow$ Area = A1
DeptId > 200 $\Rightarrow$ Area = A2

The minterm predicates for Employee:

Area = A1
Area = A2

The minterm predicates for Department:

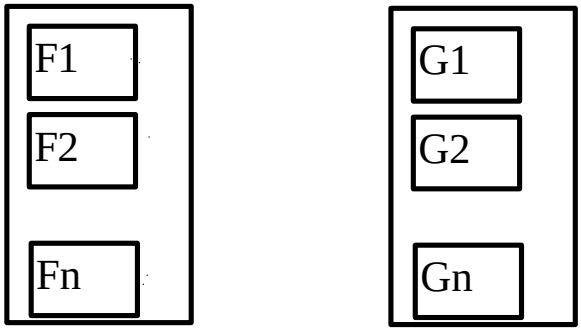
DeptId < 100
DeptId between 100 and 200 AND Area = A1
DeptId between 100 and 200 AND Area = A2
DeptId > 200

Derived Horizontal Fragmentation

The assignment of the tuple depends not on the attributes of this tuple, but on the attributes of another tuple(s).

The derived fragmentation is used mainly in the case of join.

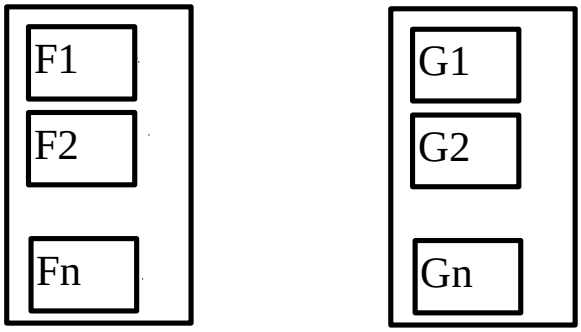
Join of fragmented tables:
Generally, to determine all pairs of matching tuple-pairs we should check every possible tuple-pairs for matching. Thus every tuples of every fragments of one table is compared with every tuples of every fragments of the other table.
The join graph in generally:



It requires a high network traffic and a lot of comparisons

The cost can be reduced if not every fragments are to be compared with each others.

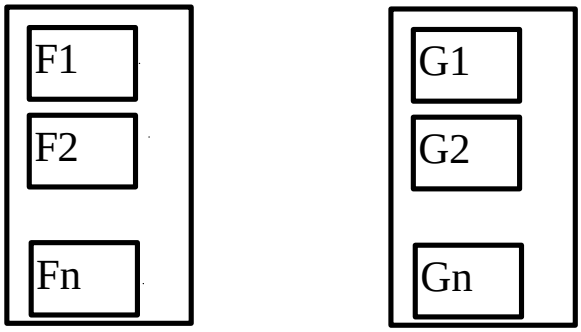
Partitioned join:



Derived Horizontal Fragmentation

The most optimal case is when every fragment is compared with only one other fragment.

Simple join graph:



The join cost can be minimized if the corresponding fragments are located on the same site.
In this case the fragmentation of the two participating tables are not independent from each others. The fragmentation of one table can be *derived* from the fragmentation of the other table.

The result of the join operation is the union of the results of the fragment joins.

Do not forget:
Different joins require different fragmentation, thus we should analyze which type of join will dominate the applications, how can we combine the different fragmentation requirements.
If join A based on condition C1, the join B is based on condition C2, then the fragmentation based on C1 $\wedge$ C2 meets both requirements.

Derived Horizontal Fragmentation

Example

two tables:

Car (number, type, age, owner)
Citizen (id, name, city)

The application requires very often to query the cars with the data of the owner together.
The owner data can be assigned to the car only through the join operation of the two tables.

The join condition is car.owner = citizen.id.

The citizen table is horizontally fragmented by the city attribute.

In order to perform the join in a simple way, the fragmentation of the car table is derived from the fragmentation of the citizen table.

The derivation is performed by the execution of a semijoin operation:

$$car_I = car \times_{owner = id} citizen_I$$

where I denotes the fragment index. The car table has as many fragments as the citizen table has.

Thus the join can be partitioned into non-overlapping parts:

$$car \times_{owner = id} citizen = \cup_I car_I \times_{owner = id} citizen_I$$

Vertical Fragmentation

The assignment of the data elements in a table is based on the attribute identifier of the data. In this case the different projections are the content of the fragments.

Fragmentation rule:
Every attribute must belong to at least one fragment, and every fragment must have a tuple identifier.

Vertical partitioning:
every attribute is contained in only one fragment.

Vertical clustering:
an attribute may be contained in more than one fragments.

The vertical clustering causes replication of some data elements. The replication is more advantageous for the read-only applications than for the read-write application. In the later case, the same update operation is performed on several sites.

Identification of the fragmentation
The fragmentation of an R relation schema into R1 and R2 is only then advisable when there are different applications that use either R1 or R2 but not both.

The fragmentation is based on the analysis of the application requirements. Every attribute is checked against every applications. This is a heuristic method.

Vertical fragmentation

There are two main heuristics methods to perform the vertical fragmentation:

- *split approach* : a global relation schema is recursive split into disjoint segments
- *grouping approach* : the single attributes are progressively grouped into larger segments

The main steps in the design of the vertical fragmentation

- analyze the applications
  (A_1,A_2,A_3,A_4)

- analyze the queries
  (Q_1,Q_2,Q_3,Q_4)

- determine the reference-matrix
  (1: Q_i references A_j, 0: not)

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

Vertical fragmentation

- We define the *attribute affinity matrix*

$$\mathrm{aff}(A_i, A_j) = \sum_{k|\mathrm{use}(q_k, A_i)=1 \wedge \mathrm{use}(q_k, A_j)=1} \; \sum_{\forall \mathrm{Site}_l} \mathrm{acc}_l(q_k) * \mathrm{ref}_l(q_k)$$

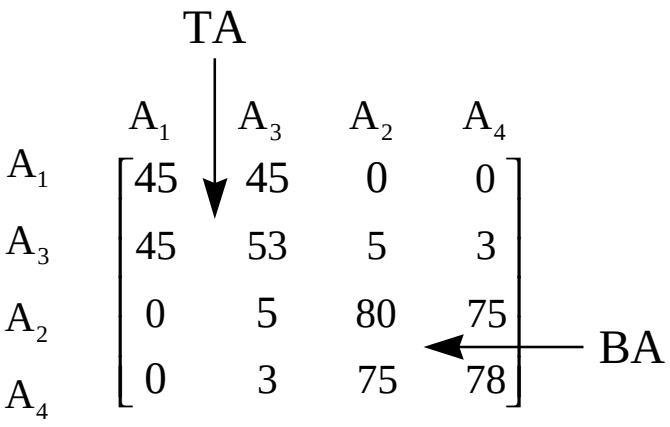
- We apply the algorithm BEA to the *attribute affinity matrix*, in order to compute the table of *attribute array affinity matrix*

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

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

$$\text{goal} : \max \sum_{i=1}^{n} \sum_{j=1}^{n} aff(A_i, A_j) * [aff(A_i, A_{j-1}) + aff(A_i, A_{j+1})]$$

Vertical fragmentation



$AQ(q_i) = \{A_j \mid use(q_i, A_j) = 1\}$
$TQ = \{q_i \mid AQ(q_i) \subseteq TA\}$
$BQ = \{q_i \mid AQ(q_i) \subseteq BA\}$
$OQ = Q - TQ - BQ$

$$CTQ = \sum_{q_i \in TQ} \sum_{\forall Site_j} acc_j(q_i) * ref_j(q_i)$$

$$CBQ = \sum_{q_i \in BQ} \sum_{\forall Site_j} acc_j(q_i) * ref_j(q_i)$$

$$COQ = \sum_{q_i \in OQ} \sum_{\forall Site_j} acc_j(q_i) * ref_j(q_i)$$

$z = CTQ * CBQ - COQ^2$

Mixed Fragmentation

The relation is fragmented both vertically and horizontally.

The *mixed fragmentation* means that a fragment is recursively fragmented.
Two types of sub-fragmentation:

- horizontally fragment is vertically fragmented
- vertically fragment is horizontally fragmented

Fragmentation tree:
The hierarchy of the fragments. The root element is the base table. There are two types of connection: vertically or horizontally fragmentation. For every node, the incoming and the outgoing edges are of different type. Every edge is assigned with the selection criteria. Every outgoing edges of a node belong to the same fragmentation.

In the praxis, the fragmentation tree is usually not higher than two levels.

Example
Base table:

Employee (Id, Name, Age, Salary, Dept)

Fragmentation criteria:

- vertically : V1 : (Id, Name, Age, Dept), V2: (Id, Salary, Dept)
- horizontally : H1: Salary < 100, H2: 100 <Salary <200, H3: 200 < Salary

Fragment placement

The fragments are logical and storage units, containing the data of similar access mode.
The fragments are assigned to sites.

A site can contain different fragments, and there are several candidate sites for the storage of a fragment.

Goals of fragment placement:
Minimal cost:
- storage cost
to find the sites with low storage costs, no unnecessary replications
- data retrieval cost:
to try to place the fragment on the site of the application
- data update cost:
to minimize the replication of the read-write data elements
- communication cost:
to find the sites near the application and to try to maximize the local processing

Maximal performance:
- minimize response time
every application should find a replica of the data on the local or a close site
- maximize throughput
to allow several concurrent applications

Fragment placement strategies

The fragment placement problem has a lot of similarity with the file allocation problem, but it also differs from it in some aspects:
- the fragments belong strongly together
- the fragments are not independent from each others
- the operations are complex and may access several fragments.

Main fragment allocation strategies:
- *nonredundant*:
every fragment is stored at only one site, no replication
- *redundant*:
a fragment may be replicated on different sites

The redundant storage may rise the efficiency but it is more complicated to design it:
- how many and which type of replications should be implemented to gain an appropriate efficiency
- the number of alternatives increases dramatically, as a fragment may be allocated to different sites at the same time.

Placement algorithms:
- best - fit :
nonredundant method, the site with best measure is selected
- all beneficial sites :
redundant, all the sites with measures upper a limit are selected
- additional replication:
redundant, starting from a nonredundant allocation adds beneficial replications to the system

Horizontal fragmentation

Best-fit approach:
The sites will be evaluated how many accesses are required from the applications located on this site:

$$B^1_{ij} = \Sigma_k f_{kj} n_{ki}$$

where

- f_{kj} the frequency of application k at site j
- n_{ki} the number of accesses from application k to fragment i

The site with greatest B_{ij} value is selected for the fragment i.

All beneficial site approach:
The goodness of a site is measured by the benefit of the local read accesses and the costs of remote update accesses:

$$B^2_{ij} = \Sigma_k f_{kj} r_{ki} - C \times \Sigma_k \Sigma_{l<>j} f_{kl} u_{ki}$$

where

- f_{kj} the frequency of application k at site j
- r_{ki} the number of read accesses from application k to fragment i
- u_{ki} the number of update accesses from application k to fragment I
- C update cost relative to the read

Additional replication approach:
Beside the costs involved in B^2_{ij} a so called reliability factor is contained in the measure too. This factor increases with the increasing number of replications.

Vertical Fragmentation

Split approach

The benefits of a split operation is calculated by the following formula:

$$B = \Sigma_{k \in As} f_{ks} n_{ki} + \Sigma_{k \in At} f_{kt} n_{ki} - \Sigma_{k \in A1} f_{kr} n_{ki} - \Sigma_{k \in A2} 2 f_{kr} n_{ki} - \Sigma_{k \in A3} \Sigma_{j \notin r,s,t} f_{kj} n_{ki}$$

where

- r : the site-index of the base relation to be fragmented
- s and t are the site-indexes where the two new fragments of base relation are allocated.
- As and At the applications running at site s or t
- A1: applications on site r accessing only one of the r or s fragments.
- A2: applications on site r accessing both of the r or s fragments.
- A3: applications on site differ from r,s,t accessing both of the r or s fragments.

Grouping approach

The benefit function is extended by components referring to the replications.
We must differ the read and the update accesses