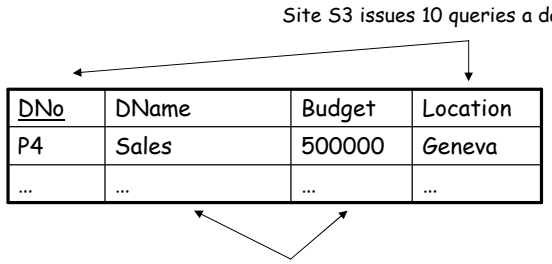
3. Vertical Fragmentation

- Vertical Fragmentation of a single relation
- Modeling the access to the relations

- Example



Site S1 issues 5 queries a day using these attributes

Site S2 issues 20 queries a day using these attributes

Site S3 issues 10 queries a day using these attributes

Similarly as for tuples in the horizontal fragmentation we can also analyze for the vertical fragmentation of how attributes are accessed by applications running on different sites. Using this information then the goal would be to place attributes there were they are used most. In this example we see that two attributes are always accessed jointly, and that site S2 is the one that uses them most. So S2 might be a candidate to place the attributes. For similar reasons DNo and Location are best placed at S3.

Correct Vertical Fragments

- Vertical Fragmentation ?

DNo	Location
P4	Geneva
...	...

Fragment moved to S3

DName	Budget
Sales	500000
...	...

Fragment moved to S2

- Primary key must occur in every vertical fragment, otherwise the original relation cannot be reconstructed from the fragments

DNo	Location
P4	Geneva
...	...

Fragment moved to S3

DNo	DName	Budget
P4	Sales	500000
...	...	...

Fragment moved to S2

- Possible vertical fragments are all subsets of attributes that contain the primary key

If we partition the attributes in the way described before we have a problem: Since the fragment moved to S2 does not contain the primary key of the relation (which is underlined), we no longer can reconstruct which tuple in this fragment corresponds to which tuple in the fragment kept at S3 and we could not reconstruct the relation. Therefore there is no other possibility than to keep also the primary key attribute at S2. This form of replication of data values is unavoidable in vertical fragmentation. Therefore in the following we assume that the primary key attributes are always replicated to all fragments.

Modeling Access Characteristics

- Possible Queries
  q1 SELECT Budget FROM DEPARTMENT WHERE Location="Geneva"
  q2 SELECT Budget FROM DEPARTMENT WHERE Budget>100000
  q3 SELECT Location FROM DEPARTMENT WHERE Budget>100000
  q4 SELECT DName FROM DEPARTMENT WHERE Location="Paris"
  etc.

	DName	Budget	Location
q1, q3	0	1	1
q2	0	1	0
q4			

DEPARTMENT

DNo	DName	Budget	Location
P4	Sales	500000	Geneva
...	...	...	...

Matrix Q describing which type of query accesses which attributes

Modeling the access characteristics for vertical fragmentation differs from the case of horizontal fragmentation for an important reason: the number of attributes is compared to the number of tuples fairly small.

One of the consequences from that observation is that is very well possible that many different applications access exactly the same subset of attributes of a relation. This is unlikely to occur in horizontal fragmentation for subsets of tuples.

In the example we see of how applications are characterized as different queries. In the matrix Q we describe now simply which of the queries (applications) access which attributes. Queries accessing the same subset of attributes are treated the same in the following.

Modeling Access Characteristics

- Access frequencies from different sites

	S1	S2	S3
q1, q3	20	0	15
q2	0	10	20
q4	10	5	0

Queries using attributes Budget and Location are issued from S3 15 times a day

Matrix S describing which type of query is accessed how frequently from each site

What is relevant for modeling the access characteristics is of how attributes are accessed at different sites. Thus we record in a second matrix M the access frequency for each site (in our example three sites S1, S2, S3) for each type of application. The application types correspond to the ones we have identified previously with matrix Q. Matrices Q and M together thus model of how the relation is accessed by the distributed applications.

Modeling Access Characteristics

- Every subset of attributes will be most likely accessed differently by some application
  - Thus trying to find subsets that are uniformly accessed is futile
  - Rather find subsets of attributes that are most similarly accessed
- Determine how often attributes are accessed jointly (affinity)
  - aff(Dname, Budget) = 0
    aff(Budget, Location) = 20+15 = 35
    aff(Dname, Dname) = 15
    aff(Budget, Budget) = 65

$$aff(A_i, A_j) = \sum_{k \text{ such that } Q_{ki}=1, Q_{kj}=1} \sum_{l=1}^{s} S_{kl}$$

	Budget	DName	Location
Budget	65	0	35
DName	0	15	
Location	35		50

Matrix A describing attribute affinity

	DName	Budget	Location
q1, q3	0	1	1
q2	0	1	0
q4	1	0	1

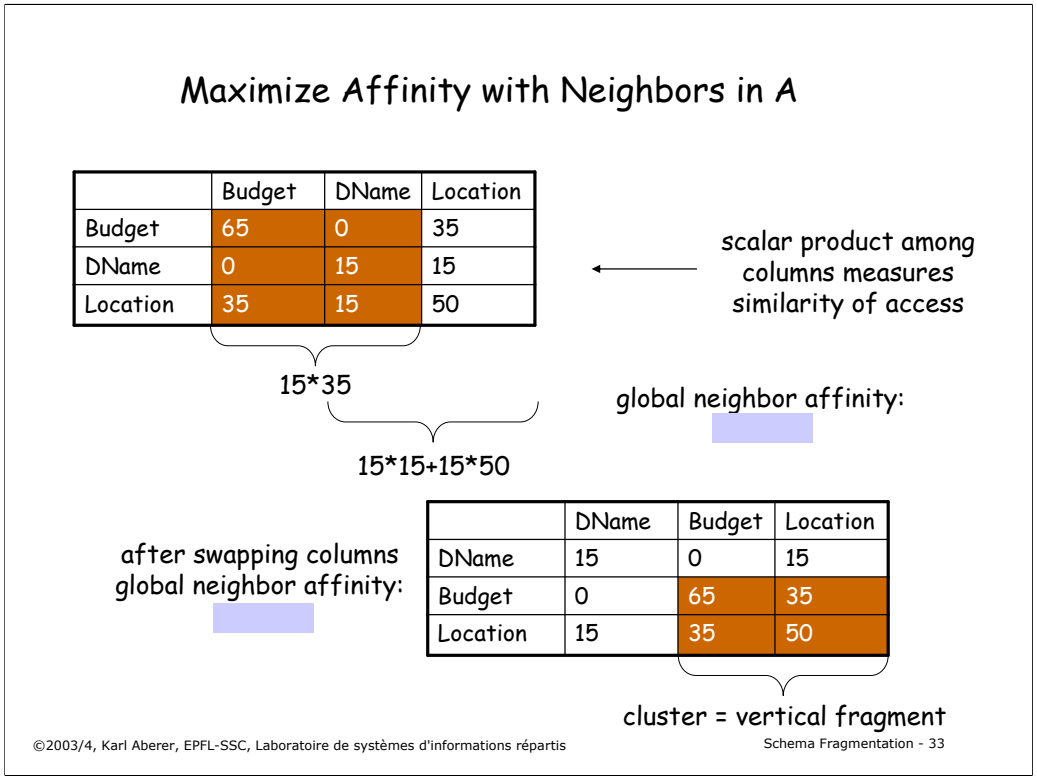
	S1	S2	S3
q1, q3	20	0	15
q2	0	10	20
q4	10	5	0

For horizontal fragmentation the goal was to group together in fragments those tuples that are accessed exactly the same by all applications. The corresponding idea for attributes would be to partition the subsets of attributes in a way that each subset of the partition is accessed by all applications the same. Given the fact that the number of applications is probably large compared to the number of attributes such an approach most likely ends up in fragmenting the relation into fragments containing one attribute each (ignoring the additional primary key attribute), thus always the finest possible fragmentation would occur. This is obviously not very interesting.

Thus we relax our goals in order to achieve reasonable sized fragments containing multiple attributes, and require only that attributes are accessed **similarly** within a fragment. In the following we thus introduce a method that allows to detect similar access patterns to attributes, based on clustering.

A first step towards this goal is to identify for each pair of attributes how often they are accessed jointly. Finding attributes that are similarly accessed by many applications and clustering them together will be the basis for identifying vertical fragments.

For that purpose we compute from the information contained in our access model (matrices M and Q) an affinity matrix A, as described and illustrated above.



Given the matrix Q we can now determine of how well neighboring attributes in A (neighboring rows, resp. columns) fit to each other in terms of access pattern. To that end we compute for each neighboring two columns in the matrix the neighborhood affinity as the scalar product of the columns.This provides a measure how similar the two columns are (remember: of the scalar product is 0 two vectors are orthogonal). Adding up all similarity values for all neighboring columns provides a global measure of how well columns fit.

The second figure shows an interesting point: by swapping two columns (and the corresponding rows) the global neighborhood affinity value increases substantially. Qualitatively we can see that in fact in the matrix a cluster formed, where the columns related to Budget and Location appear similar to each other. For vertical fragmentation this indicates that the two attributes should stay together in the same vertical fragment, since in many applications if the one attribute is accessed also the other is accessed and thus less communication occurs if the attributes reside in the same sites. Also one can conclude in order to locate good clusters, first, it is important to increase the global neighbor affinity measure.

Bond Energy Algorithm

- Clusters entities (in this case attributes) together in a linear order such that subsequent attributes have strong affinity with respect to A

Algorithm **BEA** (bond energy algorithm)
Given: nxn affinity matrix A
Initialization: Select one column and put it into the first column of output matrix
Iteration Step i: place one of the remaining n-i columns in the one of the possible i+1 positions in the output matrix, that makes the largest contribution to the global neighbor affinity measure
Row Ordering: order the rows the same way as the columns are ordered

Contribution of column, when placing Ak between Ai and Aj:

$cont(Ai, Ak, Aj) = bond(Ai, Ak) + bond(Ak, Aj) - bond(Ai, Aj)$

$bond(Ax, Ay) = \sum_{z=1..n} aff(Ax, Az)\, aff(Az, Ay)$

The question is whether we can always reorganize the matrix in the way described before, by properly exchanging columns and rows (i.e. changing the attribute order). For a set of attributes there exist many possible orderings, i.e. for n attributes n! orderings.

To that end there exists an efficient algorithm for finding clusters: The **bond energy algorithm** proceeds by linearly traversing the set of attributes. In each step one of the remaining attributes is added. It is inserted in the current order of attributes such that the maximal contribution is achieved. This is first done for the columns. Once all columns are determined the row ordering is adapted to the column ordering and the resulting affinity matrix exhibits the desired clustering.

For computing the contribution to the global affinity value one computes the gain obtained by adding a new column, and subtracts from that the loss incurred through separation of previously joint columns. The contribution of a pair of columns is the scalar product of the columns, which is maximal if the columns exhibit the same value distribution (or when considered as vectors in a vector space: point into the same direction)

Example

	A1	A2	A3	A4
A1	45	0	45	0
A2	0	80	5	75
A3	45	5	53	3
A4	0	75	3	78

	A2	A1	A3	
A1	0	45	45	
A2	80	0	5	
A3	5	45	53	
A4	75	0	3	

cont(_, A2,A1)=
bond(_,A2)+bond(A1,A2)-bond(_,A1)=

	A1	A3		
A1	45	45		
A2	0	5		
A3	45	53		
A4	0	3		

	A1	A2	A3	
A1	45	0	45	
A2	0	80	5	
A3	45	5	53	
A4	0	75	3	

cont(A1,A2,A3)=
bond(A1,A2)+bond(A2,A3)-bond(A1,A3)=
225+890-4410=-3295

A1 first choice
A3 already added
(no decision to be made)
which attribute to chose next ?

	A1	A3	A2	
A1	45	45	0	
A2	0	5	80	
A3	45	53	5	
A4	0	3	75	

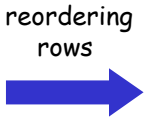
cont(A3,A2,_)=
bond(A3,A2)+bond(A2,_)-bond(A3,_)=
890+0-0=890

This example illustrates of how BEA works. The first two columns can be arbitrarily chosen, since no decision needs to be made. For the third column there exist however three possibilities to place it. For each placement a different contribution can be obtained. In case the neighbouring column required to compute the contribution is empty, the bond value is set to 0. The computation shows that A2 is to be positioned at the third position (if A1 and A2 would have been added at the beginning, in the second step A3 would go into the second position)

Example

- Row reorganization

	A1	A3	A2	A4
A1	45	45	0	0
A2	0	5	80	75
A3	45	53	5	3
A4	0	3	75	78



	A1	A3	A2	A4
A1	45	45	0	0
A3	45	53	5	3
A2	0	5	80	75
A4	0	3	75	78

- How to split the attributes into clusters ?

	A1	A3	A2	A4
A1	45	45	0	0
A3	45	53	5	3
A2	0	5	80	75
A4	0	3	75	78

only in these cases applications
would access tuples from different sites

The last step will result in adding A4 into the fourth position. Then the rows are reordered.

From the resulting affinitiy matrix we can nicely "see" the clusters that would be the optimal vertical fragmentation. But how to compute these clusters ?

Vertical Splitting

maximize split quality sq = acc(VF1)*acc(VF2) – acc(VF1,VF2)^2 !

	A1	A3	A2	A4
A1	45	45	0	0
A3	45	53	5	3
A2	0	5	80	75
A4	0	3	75	78

VF1 VF2

access VF1 only: 0
access VF1 and VF2: 45
access VF2 only: 83
sq = -45^2

	A1	A3	A2	A4
A1	45	45	0	0
A3	45	53	5	3
A2	0	5	80	75
A4	0	3	75	78

VF1 VF2

access VF1 only: 45
access VF1 and VF2: 8
access VF2 only: 75
sq = 45*75-8^2

	A1	A3	A2	A4
A1	45	45	0	0
A3	45	53	5	3
A2	0	5	80	75
A4	0	3	75	78

VF1 VF2

access VF1 only: 50
access VF1 and VF2: 128
access VF2 only: 0
sq = -128^2

	A1	A2	A3	A4
q1	1	0	1	0
q2	0	1	1	0
q3	0	1	0	1
q4	0	0	1	1

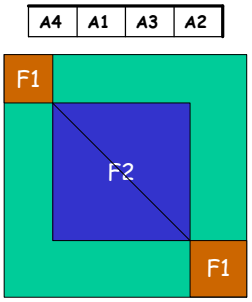
	S1	S2	S3
q1	15	20	10
q2	5	0	0
q3	25	25	25
q4	3	0	0

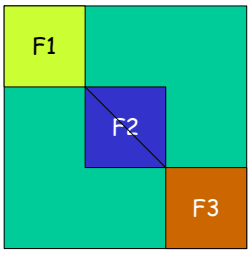
For finding clusters we have to go back to our access model. We can see above that we have three possibilities to split the set of attributes into two fragments. For each of the possibilities we have to determine what would be the result: This is done by computing in how many cases access are made to attributes from one of the two fragments only (this is good) and to attributes from the two fragments (this is bad). To compare we compute the split quality by producing a positive contribution for the good cases and a negative for the bad cases (see formula).

The computation of the number is simple given our access model. For each of the queries q1-q4 (note this model is different from the one we introduced at the beginning in the first example) we select the cases where attributes from one and where attributes from both fragments are accessed, by inspecting matrix Q. For these cases we add over all sites the total number of accesses made by taking them from matrix M.

Vertical Splitting Algorithm

- Two problems remain:
- Cluster forming in the middle of the matrix A
  - shift row and columns one by one
  - search best splitting point for each "shifted" matrix
  - cost O(n^2)
- Split in two fragments only is not always optimal
  - find m splitting points simultanously
  - try 1,2,…,m splitting points and select the best
  - cost O(2^m)





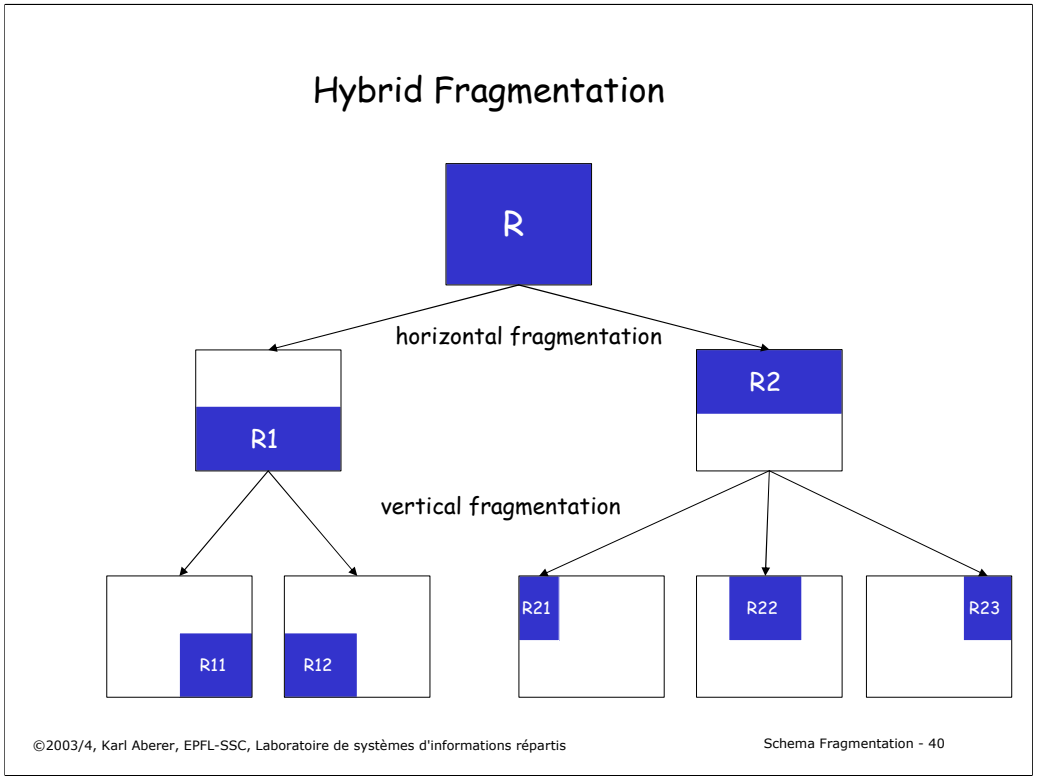
On the previous slide we considered only a simple way to split the matrix: namely selecting a split point along the diagonal, and taking the resulting upper and lower "quadrant" as fragments.

Now assume that the attributes are rotated (a possible rotation is indicated). Then the same upper and lower fragments would look as illustrated in the upper figure, and we would not be able to identify the fragment anymore by using the simple method described before. What this means is that it is possible that we might miss "good" fragments, depending on the choice of the first attribute (which is random). Therefore a better way is to consider all possible rotations of attributes, which increases the cost of search, but also allows to investigate many more alternative.

Another issue is that actually the simultaneous split into multiple fragments reveals good clusters. If for example three good clusters exist, as indicated in the figure below, it is by no means always the case that any combination of two of the clusters will ever be recognized as good clusters, and thus such a split can not be obtained by subsequent binary splits.

Summary Vertical Fragmentation

- Properties
  - Relation is completely decomposed
  - We can reconstruct the original relations from fragments by relational join
  - The fragments are disjoint with exception if primary keys that are distributed everywhere
- Application provides information on
  - what are attributes accessed by single applications
  - what are the access frequencies of the applications
- Algorithm BEA
  - identifies clusters of attributes that are similarly accessed
  - the clusters are potential vertical fragments



To round up the picture we describe of how horizontal clustering and vertical clustering can be combined. The figure illustrates of how a horizontal fragmentation of a relation R is further refined performing vertical fragmentations of the obtained horizontal fragments as if they were separate relations.

An interesting question is why one should perform first the horizontal fragmentation: again the reason is that there exist many more tuples than attributes and thus many more possible horizontal fragmentations than vertical fragmentations can exist. When first choosing a unique vertical fragmentation for all the possible horizontal fragmentations one would unnecessarily constrain the search space.

4. Fragment Allocation

- Problem
  - given fragments F1, …, Fn
  - given sites S1, …, Sm
  - given applications A1, …, Ak
  - find the optimal assignment of fragments to sites such that the total cost of all applications is minimized and the performance is maximized
- Application costs
  - communication, storage, processing
- Performance
  - response time, throughput
- problem is in general NP complete
  - apply heuristic methods from operations research

Generating the fragments (both vertically and horizontally) creates a fragmentation that takes into account in an optimized manner the information that is available about the access behavior of the applications. The fragmentation is as fine as necessary to take into account all important variations in behavior, but not finer. An important issue that we do not treat here, is the problem of allocating the fragments that have been identified to the best possible sites.

This problem can be described in a rather straightforward way, by taking into account all types of costs that occur during processing considering the applications running on the database. The optimality criterion has also to balance between the resource costs incurred and the performance achieved for the user (in terms of response time and throughput).

Having formulated the problem in this manner it can be reduced to standard operations research problems. For the solution a number of heuristic approaches from OR have been adopted.

Summary

- Why do we use different clustering criteria for vertical and horizontal fragmentation (similar access vs. uniform access) ?
- Why does the affinity measure for attributes lead to a useful clustering of attributes in vertical fragments ?
- How does the Bond Energy Algorithm proceed in ordering the attributes ?
- What is the criterion to find an optimal splitting of the ordered attributes ?
- Which variants exist for searching optimal splits ?

References

- Course material based on
  - M. Tamer Özsu, Patrick Valduriez: Principles of Distributed Database Systems, Second Edition, Prentice Hall, ISBN 0-13-659707-6, 1999.
  - Web page: http://www.cs.ualberta.ca/~database/ddbook.html
- Relevant articles
  - Stefano Ceri, Mauro Negri, Giuseppe Pelagatti: Horizontal Data Partitioning in Database Design. SIGMOD Conference 1982: 128-136
  - Shamkant B. Navathe, Stefano Ceri, Gio Wiederhold, Jinglie Dou: Vertical Partitioning Algorithms for Database Design. TODS 9(4): 680-710 (1984)