

# Distributed Data Management

## Part 1 - Schema Fragmentation

©2011, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Schema Fragmentation - 1

## Today's Question

1. Distribution of Relational Databases
2. Horizontal Fragmentation of Relational Tables
3. Vertical Fragmentation of Relational Tables

## 1. Relational Databases

DEPARTMENTS

DNo	DName	Budget	Location
P4	Sales	500000	Geneva
P2	Marketing	300000	Paris
P3	Development	250000	Munich
P1	Development	150000	Bangalore
P5	Marketing	120000	Geneva
P6	Development	90000	Paris
P7	Research	80000	Paris

SALARIES

Skill	Salary
Director	200000
Manager	100000
Assistant	50000

EMPLOYEES

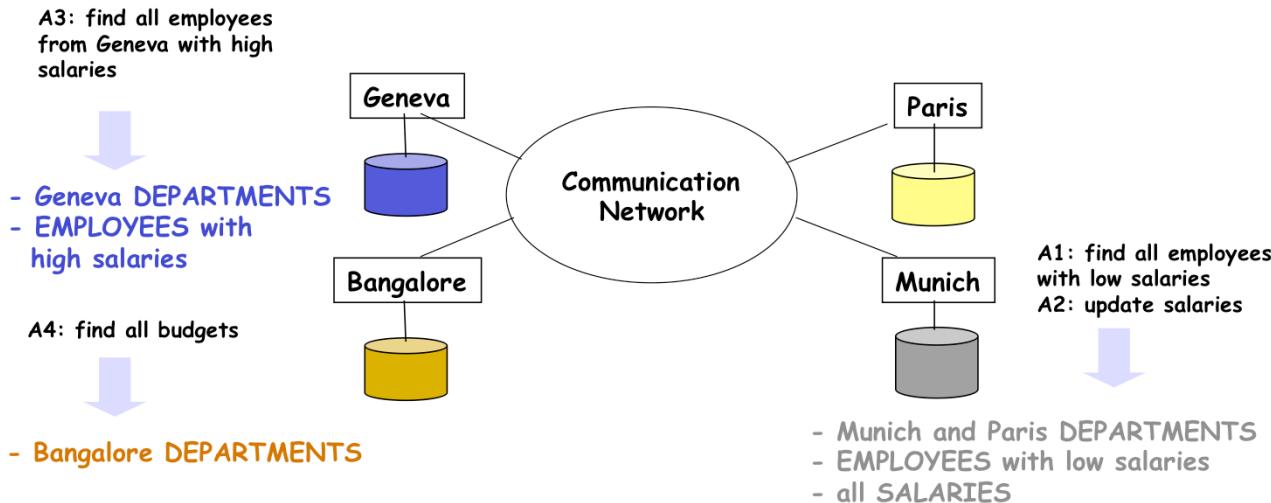
ENo	EName	Title	DNo
E1	Smith	Manager	P1
E2	Lee	Director	P1
E3	Miller	Assistant	P2
E4	Davis	Assistant	P3
E5	Jones	Manager	P3

©2011, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Schema Fragmentation - 3

In this lecture we study the problem of managing distributed relational databases, in particular the problem of distributing the data of a relational database in a network. In the following we assume that we are familiar with the basic notions of the relational data model, including the notions of relation, attribute, query, primary and foreign keys, relational algebra operators, and relational calculus (in particular the notion of predicates and basic logical operators). The principles that we will learn about apply, however, beyond the relational data model and can be applied to other data models as well.

## Distributed Relational Databases



Given a relational database:

Improve performance of queries by properly distributing the database to the physical locations (design of a distribution schema)

©2011, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Schema Fragmentation - 4

Sharing a (relational) database that by distributed applications in a network opens the possibility to optimize the access to the database, by taking into account the frequency of access by the different applications in the network to the database. Obviously it is of advantage is to move the data close to the location where it is used most frequently, in order to reduce the cost of accessing data over the network. The general assumption in distributed data management is that the communication cost for accessing data over the network is high as compared to local access to data.

The example illustrates the situation, where a relational database, for which we have given a schema on the previous slide, is distributed over the sites where the database is accessed. The applications are indicated by A1-A4). A distribution schema, i.e. the description which parts of the database are distributed to which site, and the application, given as queries, are shown. A possible reasoning for distributing the data as shown in the example could be as follows: since A3 needs all information about Geneva departments and high salary employees we put the related data in that site. In Bangalore only the DEPARTMENT table is accessed, but parts of it are allocated to other sites as they are used there, therefore only the locally relevant data is kept. Paris has no applications, so no data is put there. Munich has all other data, in particular, for example, the salary table, which is also used in Geneva, but more frequently in Munich, as it is updated there.

In the following we will introduce methods of how to make the specification of such a distribution schema precise and provide algorithms that support the process of developing such a distribution schema. This task is called also *distributed database design*.

## What Do You Think ?

- Problems to solve when distributing a relational database

## Assumptions on Distribution Design

- Distributed relational database system is available
  - allow to distribute relational data over physically distributed sites
  - takes care of transparent processing of database accesses
- Top-down design
  - no pre-existing constraints on how to organize the database
- Access patterns are known and static
  - no need to adapt to changes in access patterns (otherwise redesign)
- Replication is not considered
  - reasonable assumption if updates are frequent

The problem of distributing a relational database is a very general one: we will make a number of assumptions in order to be able to focus on specific questions. We will not concern ourselves with the issue of developing a distributed database system architecture. This requires to address a number of important problems, such as communication support, management of the data distribution schema, and processing of distributed queries. We assume that if we can specify of how the data is to be distributed all other issues are taken care of.

We also assume that there exist no a-priori constraints on how we distribute the database, be it of technical or organizational nature. We are free to decide which data goes where. The access patterns are assumed to be static, or changing so slowly that we can afford to perform a re-design whenever needed. Thus we can design our distribution schema off-line. Finally we do not take advantage of replication, which is a reasonable assumption in update-intensive environments. The use of replication in the methods that follow is possible, but we will not consider it for the purpose of clarity.

## Degree of Fragmentation

- Complete relations are too coarse, single attribute values are too fine
  - Determine proper parts (fragments) of relations
  - Idea: use a SQL query against a relation to specify fragments
- Example

```
SELECT Dno, DName FROM DEPARTMENT WHERE Budget > 200000
```

vertical fragment

The diagram illustrates the fragmentation of a relational table. On the left, a large brace labeled "horizontal fragment" groups all seven rows of the table. Above the first row, another brace labeled "vertical fragment" groups the first three columns (DNo, DName, and Budget). The table has four columns: DNo, DName, Budget, and Location. The data is as follows:

DNo	DName	Budget	Location
P4	Sales	500000	Geneva
P2	Marketing	300000	Paris
P3	Development	250000	Munich
P1	Development	150000	Bangalore
P5	Marketing	120000	Geneva
P6	Development	90000	Paris
P7	Research	80000	Paris

©2011, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis      Schema Fragmentation - 7

A first important question is: to which degree should fragmentation occur, i.e., which parts of a relation can be distributed independently. We will call these parts in the following "fragments". Restricting the distribution to complete relations appears to be too limited in general, in particular when considering tables containing information relevant for different sites. On the other extreme, deciding on the distribution for each single attribute value or tuple seems to be a too costly approach when considering the distribution of very large tables. A flexible way to create fragments that can be distributed to the sites is to use queries which can select subsets of a relational table, as shown in the example. These fragments can be of two different kinds:

1. **horizontal fragments** of a table are defined through selection (i.e. what is specified in the WHERE clause of a SQL query). These are subsets of tuples of a relation.
2. **vertical fragments** of a table are defined through projection (i.e. what is specified in the SELECT clause of a SQL query). These are subtables consisting of a subset of the attribute columns.

## Correct Fragmentation

- **Completeness**
  - decomposition of a relation  $R$  into fragments  $R_1, \dots, R_n$  is complete if every attribute value found in one of the relations is also found in one of the fragments
- **Reconstruction**
  - if a relation is decomposed into fragments  $R_1, \dots, R_n$  then it should also be possible to reconstruct the relation  $R$  from its fragments (e.g. by applying appropriate relational operators such as join, union etc.)
- **Disjointness**
  - if a relation is decomposed into fragments  $R_1, \dots, R_n$  then every attribute value should be contained only in one of the fragments
- **Attention:** Reconstruction and (full) disjointness cannot be achieved at the same time (more later)

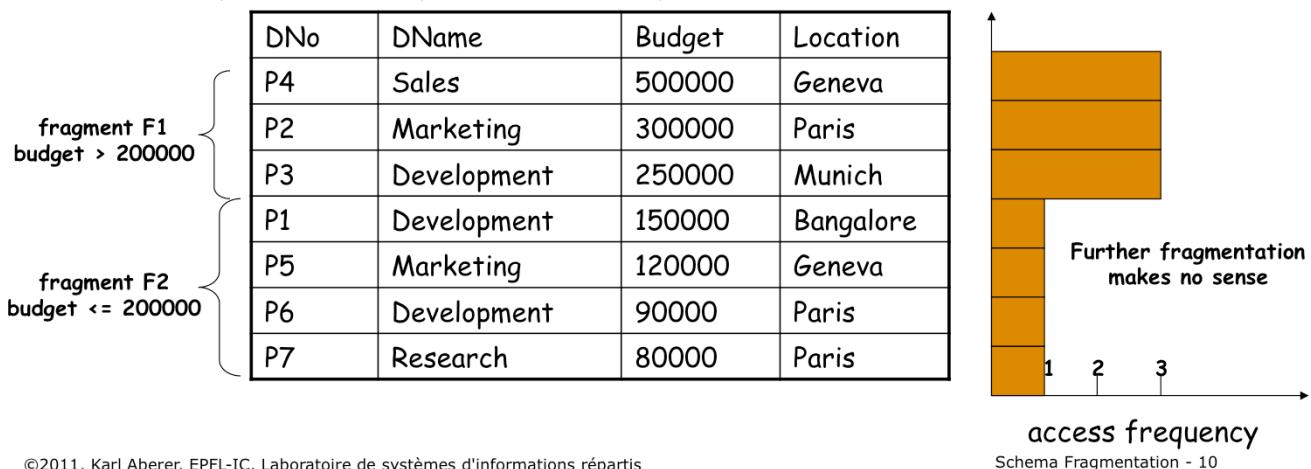
When decomposing a relational table into fragments a number of minimal requirements has to be satisfied in order to avoid the loss of information. First, we have to make sure that every data value of the original table is found in one fragment, otherwise we loose this data value. This property is called **completeness**. Second, we must be able to **reconstruct** the original table from the fragments. This is a problem very similar to the one encountered when normalizing relational database schemas by decomposition of tables. Also there it can occur that by improper decomposition we can no more reconstruct the original table. Finally, the fragments should be **disjoint** (in order to avoid update dependencies) as far as possible. We will see later that the last two conditions of reconstruction and disjointness can not be completely satisfied at the same time in general.

## Questions

- Why is fragmentation performed?
  1. To improve access to relatively static databases
  2. To adapt the access patterns to distributed data
  3. To protect against node failures in a distributed database
- The reconstruction property expresses that
  1. In case of failure of a node holding a fragment the data can be recovered from another fragment
  2. The original data can be fully recovered from the fragments
  3. Every query that can be answered on the original database can also be answered on the fragmented database
- At which phase of the database lifecycle is fragmentation performed ?
  1. At database design time
  2. During distributed query processing
  3. During updates to a distributed database

## 2. Primary Horizontal Fragmentation

- *Horizontal Fragmentation of a single relation*
- Example
  - Applications A1 at Geneva site:  
Q1: "update department budgets > 200000 three times a month"  
Q2: "update other departments monthly"



We begin by fragmenting a single relational table horizontally. Since the fragmentation of a table should depend on the usage of the table, we have first to describe of how the table is accessed. In other words, we need a model for the table access. One possible model would be to give for every single tuple the frequency of access of a specific application, as illustrated in the histogram on the right. Since we do however not consider fragmentation of a table into single tuples, this description is at a too fine granularity. Also one can see that for many tuples the access frequency will be the same (as a consequence of the structure of the application executing an SQL query)

Thus we rather model the access only for those parts of relations that potentially qualify for fragmentation. Thus the model we are interested in has to describe two things: first what are possible (horizontal) fragments about which we want to say something, and second what we want to say about the access. The answer to the first question is a consequence of our idea of using SQL to describe fragments: the horizontal fragment will be described by some form of predicate (or logical expression) that consists of

## Modeling Access Characteristics

- Describe (potential) horizontal fragments and the access to them
  - select subsets of the relation using predicates
  - all tuples in a fragment are accessed with the same frequency
  - access frequency  $af(A_i, F_j)$ : given a tuple in fragment  $F_j$ , how often is it on average accessed by application  $A_i$  per time unit
- Examples:
  - update of some tuple in  $F_j$  by  $A_i$  occurs  $t$  times per time unit:  
$$af(A_i, F_j) = t / \text{size}(F_j)$$
  - query by  $A_i$  accesses all tuples in  $F_j$  and query occurs  $t$  times per time unit:  
$$af(A_i, F_j) = t$$
  - query by  $A_i$  accesses on average 10% of all tuples in  $F_j$  and query occurs  $t$  times per time unit:  
$$af(A_i, F_j) = t / 10$$
- Obtain the necessary information
  - provided by developer

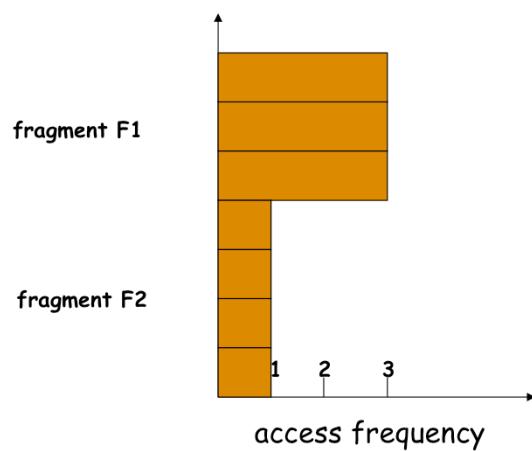
©2011, Karl Aberer, EPFL IC, Laboratoire de systèmes d'informations répartis

Schema Fragmentation - 11

The necessary information on access frequencies either can be provided by a developer, who knows the application and can derive the necessary (approximate) specification, or is obtained from analysis of database access logs. The second approach is technically more challenging, and typically will require statistical analysis or data mining tools (we will introduce basic data mining techniques at the end of this lecture). The access frequencies are measured in terms of average number of accesses to a tuple of the fragment within a time unit. Thus each access to each tuple is counted as a single access for an application. This information can be computed in different ways, as is illustrated in the examples.

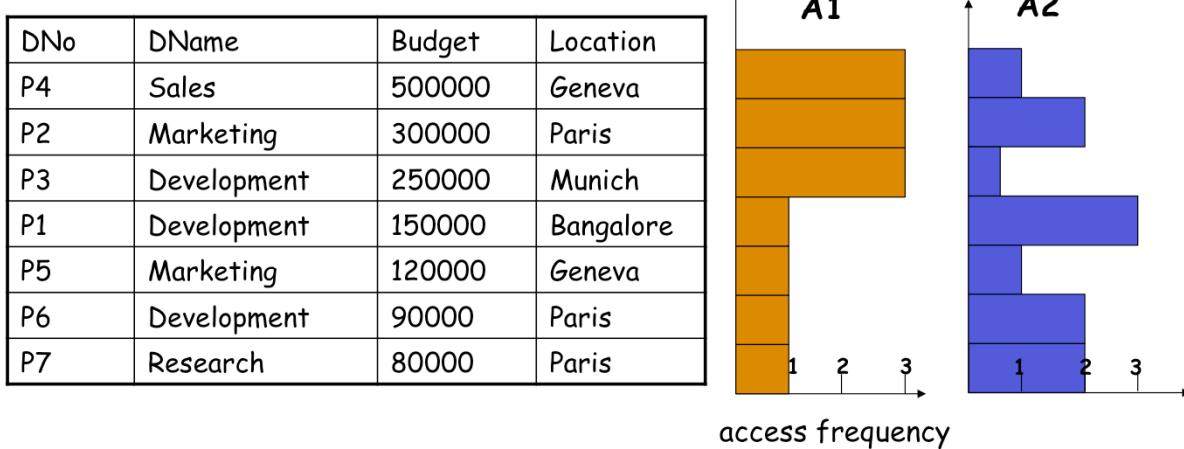
## Question

- Example: application A1 accesses
  - Fragment F1: with frequency 3
  - Fragment F2: with frequency 1
- A1 accesses the whole relation with frequency
  1. 13/7
  2. 4/7
  3. 14/7



## Example Multiple Applications

- Applications A2 running in Paris site:
  - Q3: "request the Bangalore dept budget on average three times a month"
  - Q4: "request some Geneva dept budget twice a month"
  - Q5: "request some Paris dept budget 6 times a month"
  - Q6: "request Munich dept budget every second month"



©2011, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Schema Fragmentation - 13

If we can describe the access to a relation by one application we can do the same also for other applications as shown in the example. For a single relation we know what are the potential horizontal fragments, those that are identified by the access model as having same access frequency. With multiple applications we see that there exist different possible combinations of access frequencies for different tuples since different applications fragment the relations differently. Each different combination potentially might lead to a different decision on where to locate the tuple.

## Tuple Access Frequencies

- Access frequency per tuple and application

DNo	DName	Budget	Location	af1	af2
P4	Sales	500000	Geneva	3	1
P2	Marketing	300000	Paris	3	2
P3	Development	250000	Munich	3	0.5
P1	Development	150000	Bangalore	1	3
P5	Marketing	120000	Geneva	1	1
P6	Development	90000	Paris	1	2
P7	Research	80000	Paris	1	2

The tuple access frequency table captures the frequencies of accesses of the different applications to individual tuples. We see that P6 and P7 have the same combinations of frequency, therefore these two tuples should become part of the same fragment as we will see in the following. In practice access frequencies that are similar can be considered as being the same, to simplify the subsequent fragmentation process.

## Example Access Frequencies to Fragments

- Each fragment can be described as conjunction of predicates, e.g.

$F1: \text{Location} = \text{"Paris"} \wedge \text{Budget} > 200000$

- There exist the following different combinations of access frequencies  $\langle af1, af2 \rangle$  for applications running at the two sites

$\langle af1, af2 \rangle$	$\text{Location} = \text{"Paris"}$	$\text{Location} = \text{"Geneva"}$	$\text{Location} = \text{"Munich"}$	$\text{Location} = \text{"Bangalore"}$
Budget $> 200000$	$\langle 3, 2 \rangle$	$\langle 3, 1 \rangle$	$\langle 3, 0.5 \rangle$	n/a
Budget $\leq 200000$	$\cancel{\langle 1, 2 \rangle}$	$\langle 1, 1 \rangle$	n/a	$\langle 1, 3 \rangle$

F1

**Important Observation:** if all tuples in a set of tuples are accessed with the same frequency by all applications, then whichever method we use to optimize access to tuples, these tuples will be assigned to the same site

Therefore it makes no sense to make a further distinction among them, i.e. fragments in which all tuples are accessed with equal probability are the smallest we have to consider

©2011, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'info

What we can do is to enumerate all possible combinations of access frequencies as shown in this example. We take each possible combination of horizontal fragments from the two applications. If we form the conjunction of the predicates describing the fragments in each of the applications (which can be done by using the logical AND connector), then we obtain fragments of the relational tables for which the access frequency is the same for all tuples for both applications. We find these access frequencies thus in the entries of the table capturing all possible combinations of predicates.

An important observation relates now to the fact that we have not to further fragment the table than it is done by combining all possible fragments of all applications, since whichever method we use to distributed the tuples to different sites, it will not be able to distinguish them (through the access frequency) and thus they will be moved to the same site.

## Describing Horizontal Fragments

- **(Simple) predicates P:** testing the value of a single attribute
  - Examples:  $P = \{\text{Location} = \text{"Paris"}, \text{Budget} > 200000\}$
- **Minterm predicates  $M(P)$ :** Combining all simple predicates taken from  $P$  using "and" and "not" ( $\wedge$  and  $\neg$ )
  - Example:  
If  $P = \{\text{Location} = \text{"Paris"}, \text{Budget} > 200000, \text{DName} = \text{"Sales"}\}$   
then  $\text{Location} = \text{"Paris"} \wedge \neg \text{Budget} > 200000 \wedge \text{DName} = \text{"Sales"}$   
is 1 out of 8 possible elements in  $M(P)$

Formally: Given a relation  $R[A_1, \dots, A_n]$ , then a simple predicate  $p$  is  
 $p: A_i \text{ op Value}$   
where  $\text{op} \in \{=, <, \leq, >, \geq, \text{not } =\}$  and  
 $\text{Value} \in D_i, D_i \text{ domain of } A_i$

Formally: Given  $R[A_1, \dots, A_n]$  and simple predicates  $P = \{p_1, \dots, p_m\}$ , then the set  $M(P)$  of minterm predicates consists of all predicates of the form

$$\wedge_{p_i \in P} p_i^*$$

where  $p_i^*$  is either  $p_i$  or  $\neg p_i$ .

As we have seen we need conjunctions of predicates in order to describe fragments in the general case. We make now the description of horizontal fragments more precise: Given a relation we can assume that there exists a set of atomic predicates that can be used to describe horizontal fragments, these are called simple predicates. From those we can compose complex predicates by using conjunctions and negations. More precisely we consider all possible compositions of all simple predicates using conjunction and negation. This set we call **minterm predicates** and it constitutes the set of all predicates that we consider for describing horizontal fragments.

One might wonder why disjunctions (OR) are not considered. In fact they would be of no use as they would allow to only define fragments that are the union of some fragments one obtains from minterm predicates. In other words with minterm predicates we obtain the finest partitioning of the relational table that can be obtained by using a given set of simple predicates, and this is sufficient to describe the access frequencies for all tuples.

## Horizontal Fragments

- A horizontal fragment  $F_i$  of a relation  $R$  consists of all tuples that satisfy a minterm predicate  $m_i$
- Example:  
 $m_1 : \text{Location} = \text{"Paris"} \wedge \neg \text{Budget} > 200000 \wedge \text{DName} = \text{"Research"}$   
 $m_2 : \neg \text{Location} = \text{"Geneva"} \wedge \text{Budget} > 200000$

DNo	DName	Budget	Location
P4	Sales	500000	Geneva
P2	Marketing	300000	Paris
P3	Development	250000	Munich
P1	Development	150000	Bangalore
P5	Marketing	120000	Geneva
P6	Development	90000	Paris
P7	Research	80000	Paris

F2

F1

All possible horizontal fragments are those subsets of a relation that can be selected by using a minterm predicate over a given set of simple predicates.

## Complete and Minimal Fragmentation

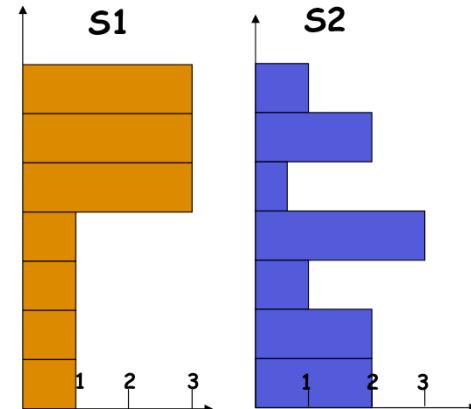
- How many simple predicates do we need ?
  - e.g. is  $P=\{\text{Budget} > 200000, \text{Budget} \leq 200000\}$  a good set ?
- At least as many such that the access frequency within a horizontal fragment is uniform for all tuples for all applications  
(otherwise we could not model the access)  
→ complete set of simple predicates
- but no more  
→ minimal set of simple predicates

The situation is now as follows. Different applications will use (propose) different simple predicates in order to describe the access to a relation. They will need as many simple predicates as necessary, to obtain fragments for which the access frequency for the specific application is uniform. As we have seen in order to describe the combined access frequencies of multiple applications to the relations we have to combine those simple predicates into complex predicates. Thus possible fragments are constructed from minterm fragments over the set of simple predicates that is the union of the set of all simple predicates used by the different applications. This set allows to construct any possible intersection of fragments originating from different applications through minterm predicates. However, a set of simple predicates obtained in this manner can contain simple predicates that are not useful, such that we would consider too many minterm predicates which lead to no additional fragments.

## Example

- $P1 = \{\text{Location} = \text{"Paris"}, \text{Budget} > 200000\}$  not complete
- $P2 = \{\text{Location} = \text{"Paris"}, \text{Location} = \text{"Munich"}, \text{Location} = \text{"Geneva"}, \text{Budget} > 200000\}$  complete, minimal ?
- $P3 = \{\text{Location} = \text{"Paris"}, \text{Location} = \text{"Munich"}, \text{Location} = \text{"Geneva"}, \text{Budget} > 200000, \text{Budget} \leq 200000\}$ , complete but not minimal

DNo	DName	Budget	Location
P4	Sales	500000	Geneva
P2	Marketing	300000	Paris
P3	Development	250000	Munich
P1	Development	150000	Bangalore
P5	Marketing	120000	Geneva
P6	Development	90000	Paris
P7	Research	80000	Paris



©2011, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Schema Fragmentation - 19

We illustrate the difference between complete and minimal set of predicates in this example.  $P1$  is not complete since it does not allow e.g. to distinguish Geneva from Munich, which have different access frequencies for A1 and A2.  $P3$  is obviously not minimal. The question is whether  $P2$  is complete and minimal.

## Example Minimal Fragmentation

$F1 : \text{Location} = \text{"Paris"} \wedge \neg \text{Location} = \text{"Geneva"} \wedge \text{Budget} > 200000$   
 $F2 : \text{Location} = \text{"Paris"} \wedge \neg \text{Location} = \text{"Geneva"} \wedge \neg \text{Budget} > 200000$   
 $F3 : \neg \text{Location} = \text{"Paris"} \wedge \text{Location} = \text{"Geneva"} \wedge \text{Budget} > 200000$   
 $F4 : \neg \text{Location} = \text{"Paris"} \wedge \text{Location} = \text{"Geneva"} \wedge \neg \text{Budget} > 200000$   
 $F5 : \neg \text{Location} = \text{"Paris"} \wedge \neg \text{Location} = \text{"Geneva"} \wedge \text{Budget} > 200000$   
 $F6 : \neg \text{Location} = \text{"Paris"} \wedge \neg \text{Location} = \text{"Geneva"} \wedge \neg \text{Budget} > 200000$

$\langle af1, af2 \rangle$	$\text{Location} = \text{"Paris"}$	$\text{Location} = \text{"Geneva"}$	$\text{Location} = \text{"Munich"}$	$\text{Location} = \text{"Bangalore"}$
Budget > 200000	$\langle 3, 2 \rangle (F1)$	$\langle 3, 1 \rangle (F3)$	$\langle 3, 0.5 \rangle (F5)$	n/a
Budget <= 200000	$\langle 1, 2 \rangle (F2)$	$\langle 1, 1 \rangle (F4)$	n/a	$\langle 1, 3 \rangle (F6)$

- $P2' = \{\text{Location} = \text{"Paris"}, \text{Location} = \text{"Geneva"}, \text{Budget} > 200000\}$  is complete AND minimal

Here we see that actually the predicate  $\text{Location} = \text{"Munich"}$  is not needed. The observation is that for the given database this predicate is only useful to distinguish  $F5$  and  $F6$ , but this can already be done with another predicate, namely  $\text{Budget} > 200000$ . Therefore  $P2$  is in fact complete and but not minimal.  $P2'$  is complete and minimal since we cannot eliminate any further simple predicate from  $P2'$  without loosing completeness.

It is very important to understand that this fact depends on the actual state of the database, i.e., the content of the relation. As soon as for example a tuple would enter the database, which contains a department in Munich with budget less than 200000, the predicate  $\text{Location} = \text{"Munich"}$  will be needed to describe the new fragment (provided it is accessed differently than current fragment  $F6$ ).

## Questions

- Consider the access frequencies below:
- How many horizontal fragments would a minimal and complete fragmentation have?
  1. 3
  2. 4
  3. 6
- Which of the following sets of simple predicates is complete?
  1. {Location = "Munich", Budget > 200000 }
  2. {Location = "Munich", Location = "Bangalore"}
  3. {Location = "Paris", Budget <= 200000 }
  4. None of those

<af1, af2>	Location = "Paris"	Location = "Geneva"	Location = "Munich"	Location = "Bangalore"
Budget > 200000	<3, 1>	<3, 1>	<1, 3>	n/a
Budget <= 200000	<1, 1>	<1, 1>	n/a	<1, 3>

## Determining a Minimal Fragmentation

- Given fragments generated by a set  $M(P)$ :  
We say a predicate  $p$  is *relevant* to  $M(P)$  if there exists at least one element of  $m \in M(P)$ , such that when creating the fragments corresponding to  $m_1 = m \wedge p$  and  $m_2 = m \wedge \neg p$  there exists at least one application that accesses the two fragments  $F_1$  and  $F_2$  generated by  $m_1$  and  $m_2$  differently (with different frequencies)

### Algorithm *MinFrag*

```
Start from a complete set of predicates P  
Find an initial  $p \in P$  such that  $p$  relevant to  $M(\{\})$   
set  $P' = \{p\}$ ,  $P = P \setminus \{p\}$   
Repeat until no  $p \in P$  is relevant to  $M(P')$   
  - select  $p \in P$  such that  $p$  relevant to  $M(P')$   
  - set  $P' = P' \cup \{p\}$ ,  $P = P \setminus \{p\}$   
  - if there exists a  $p \in P'$  that is not relevant to  $M(P' \setminus \{p\})$   
    then set  $P' = P' \setminus \{p\}$ 
```

The algorithm MinFrag determines a minimal set of simple predicates from a given set and for a given database. It proceeds by iteratively adding predicates from the given complete set of predicates. While doing that it observes two things: first, it adds only predicates that are relevant, with respect to the currently selected set of predicates. This is expressed by the concept of RELEVANCE. Second, in each step it checks whether one of the already included predicates has become non-relevant through the addition of the new predicate. In fact it might be the case that one predicate  $p_1$  is "more relevant" than another  $p_2$  included earlier, i.e. we can eliminate  $p_2$  without loosing interesting fragments but not vice versa.

## Example MinFrag Algorithm

- $P_3 = \{\text{Location} = \text{"Paris"}, \text{Location} = \text{"Munich"}, \text{Location} = \text{"Geneva"}, \text{Budget} > 200000, \text{Budget} \leq 200000\}$  is a complete set of predicates

Step 1: add Location = "Munich" (ok)

application A1 accesses (Location = "Munich") and  $\neg(\text{Location} = \text{"Munich"})$  differently

Step 2: add Budget > 200000 (ok)

application A1 accesses (Budget > 200000) and  $\neg(\text{Budget} > 200000)$  differently

Step 3: add Budget  $\leq 200000$  (no, is dropped)

no new fragments generated

Step 4: add Location = "Paris" (ok)

application A2 accesses (Location = "Paris") and  $\neg(\text{Location} = \text{"Paris"})$  differently

Step 5: add Location = "Geneva" (ok, but now

Location = "Munich" is dropped)

with (Budget > 200000), Location = "Paris", Location = "Geneva" the predicate Location = "Munich" does not generate any new fragments

We illustrate of how the MinFrag algorithm would work for our example. The dropping of the predicate in step 3 happens for obvious reasons. In step 5 the predicate Location="Munich" is dropped, since as we have seen earlier it is not required to distinguish all possible fragments. Note, that in case Location="Munich" would have only been considered in the last step (rather in the first), it would never have been included into the set P'. Thus the execution of the algorithm depends on the order of processing of predicates from the initial complete set of predicates.

## Eliminating Empty Fragments

- Not all minterm predicates constructed from a complete and minimal set of predicates generate useful fragments
- Example:  
 $\{\text{Location} = \text{"Paris"}, \text{Location} = \text{"Geneva"}, \text{Budget} > 200000\}$  is minimal
- All minterm predicates

$$\begin{array}{lll} F1 : \text{Location} = \text{"Paris"} \wedge \neg \text{Location} = \text{"Geneva"} \wedge \text{Budget} > 200000 \\ F2 : \text{Location} = \text{"Paris"} \wedge \neg \text{Location} = \text{"Geneva"} \wedge \neg \text{Budget} > 200000 \\ F3 : \neg \text{Location} = \text{"Paris"} \wedge \text{Location} = \text{"Geneva"} \wedge \text{Budget} > 200000 \\ F4 : \neg \text{Location} = \text{"Paris"} \wedge \text{Location} = \text{"Geneva"} \wedge \neg \text{Budget} > 200000 \\ F5 : \neg \text{Location} = \text{"Paris"} \wedge \neg \text{Location} = \text{"Geneva"} \wedge \text{Budget} > 200000 \\ F6 : \neg \text{Location} = \text{"Paris"} \wedge \neg \text{Location} = \text{"Geneva"} \wedge \neg \text{Budget} > 200000 \\ F7 : \text{Location} = \text{"Paris"} \wedge \text{Location} = \text{"Geneva"} \wedge \text{Budget} > 200000 \\ F8 : \text{Location} = \text{"Paris"} \wedge \text{Location} = \text{"Geneva"} \wedge \neg \text{Budget} > 200000 \end{array}$$

Finally, after executing MinFrag, it still is possible that certain minterm fragments are to be excluded for logical reasons. It is very well possible as illustrated in this example that we need a certain minimal set of simple predicates in order to properly describe all horizontal fragments of the relation, but that we can construct from this set minterm predicates that produce empty fragments, as shown in the example. The typical example is where multiple equality conditions on the same predicate are included. Then the conjunction of two such predicates in their positive form (unnegated) always leads to a contradictory predicate, resp. an empty fragment. Similarly also if the conjunction contains predicates that are implied by other predicates (e.g.  $\text{Budget} > 200000$  would imply  $\text{Budget} > 100000$ ), the implied predicate can be dropped.

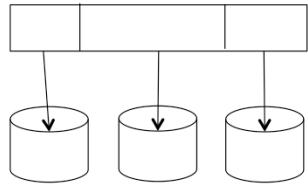
## Summary Primary Horizontal Fragmentation

- Properties
  - Relation is completely decomposed
  - We can reconstruct the original relations from fragments by union
  - The fragments are disjoint (definition of minterm predicates)
- Application provides information on
  - what are fragments of single applications
  - what are the access frequencies to the fragments
- Algorithm MinFrag
  - derives from a complete set of predicates a minimal set of predicates needed to decompose the relation completely
  - without producing unnecessary fragments

## Question

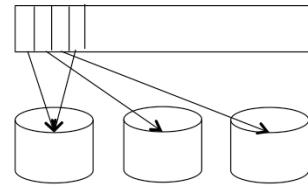
- Which statement is true for MinFrag algorithm
  - 1. The output is independent of the order of the input
  - 2. Produces a monotonically increasing set of predicates
  - 3. Always terminates
  - 4. All of the above statements are true

## Alternative Approaches to Horizontal Fragmentation



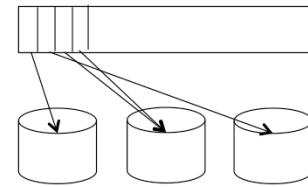
### Range partitioning

- Each node stores an attribute range
- Supports range queries
- Potentially load balancing problems



### Round-robin partitioning

- Tuples are distributed round-robin
- Supports table scans



### Hash partitioning

- A hash function on some attribute values maps to the node
- Supports associative access (all tuples with the same attribute value)

The type of horizontal fragmentation we have studied is also called range partitioning, the idea being that a relation is split into ranges according to some attribute values and then complete ranges are stored on a single node. The advantage of this method is that queries requesting such ranges can get all the data from a single node which can reduce latency and network traffic. However, this approach might also have problems with load balancing if some attribute values become particularly frequent or are particularly frequently requested. Therefore also other methods of partitioning a relational table over multiple nodes in a network have been considered. The simplest is round-robin partitioning where the  $i^{\text{th}}$  tuple of a relation is mapped to the node with number  $i \bmod n$ , when there are  $n$  nodes in the system. If only table scans are done this method performs well, however even for accessing all tuples sharing the same attribute value (associative access) requires access to multiple nodes. To remedy this situation while maintaining good load balancing hash partitioning is being used. There a tuple is mapped to a node  $h(v_1, \dots, v_k)$ , where  $h$  is a hash function with range  $[1, n]$  and  $v_1, \dots, v_k$  are the values of selected attributes  $a_1, \dots, a_k$ . Thus all tuples with the same values of  $a_1, \dots, a_k$  will be stored on the same node, while with a proper hash function the tuples will be uniformly distributed over the different nodes in the network.

## Derived Horizontal Fragmentation

DEPARTMENTS

DNo	DName	Budget	Location
P4	Sales	500000	Geneva
P2	Marketing	300000	Paris
P3	Development	250000	Munich
P1	Development	150000	Bangalore
P5	Marketing	120000	Geneva
P6	Development	90000	Paris
P7	Research	80000	Paris

← horizontal fragment

EMPLOYEES

ENo	EName	Title	DNo
E1	Smith	Manager	P1
E2	Lee	Director	P1
E3	Miller	Assistant	P2
E4	Davis	Assistant	P3
E5	Jones	Manager	P3

derived horizontal fragment

foreign key

Since the process of fragmenting a single relation horizontally is a considerable effort, the question is whether such a fragmentation can be exploited further. In fact, there exists a good reason to do so when considering of how typically relational database schemas are constructed. In general, one finds many foreign key relationships, where one relation refers to another relation by using its primary key as reference. Since these relationships carry a specific meaning it is very likely, that this foreign key relationship will also be used during accesses to the database, i.e. by executing join operations over the two relations. This means that the corresponding tuples in the two relations will be jointly accessed. Thus it is of advantage to keep them at the same site in order to reduce communication cost.

As a consequence it is possible and of advantage to "propagate" a horizontal fragmentation that has been obtained for one relation to other relations that are related via a foreign key relationship and to keep later the corresponding fragments at the same site. We call fragments obtained in this way **derived horizontal fragments**.

## Semi-Join

Formally: Given relations  $R[A_1, \dots, A_r]$  and  $S[B_1, \dots, B_s]$  where  $A_j$  is primary key of  $R$  and  $B_i$  is a foreign key of  $S$  referring to  $A_j$ .

Given a horizontal fragmentation of  $R$  into  $R_1, \dots, R_k$  then this induces the derived horizontal fragmentation

$$S_n = S \triangleright R_n, n=1, \dots, k$$

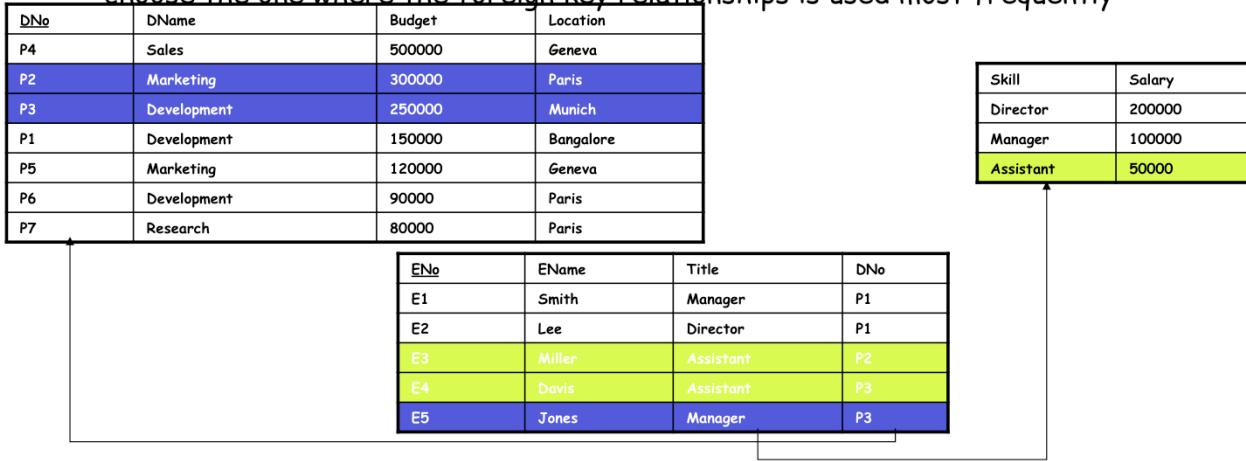
$$\text{Semi-Join: } S \triangleright R = \pi_{B_1, \dots, B_s}(S \triangleright \triangleleft R)$$

$\triangleright$  projection  
 $\triangleleft$  natural join

Formally the derived horizontal fragments can be specified using the so-called **semi-join operator**. The semi-join operator is a relational algebra operator, that joins two relations, and then projects the result to the attributes of one of the two relations. When computing the semi-join of a horizontal fragment with another relation one obtains the corresponding derived horizontal fragments of the second relation.

## Multiple Derived Horizontal Fragmentations

- Distribute the primary and derived fragment to the same site
  - tuples related through foreign key relationship will be frequently processed together (relational joins)
- If multiple foreign key relationships exist multiple derived horizontal fragmentations are possible
  - choose the one where the foreign key relationships is used most frequently



©2011, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Schema Fragmentation - 30

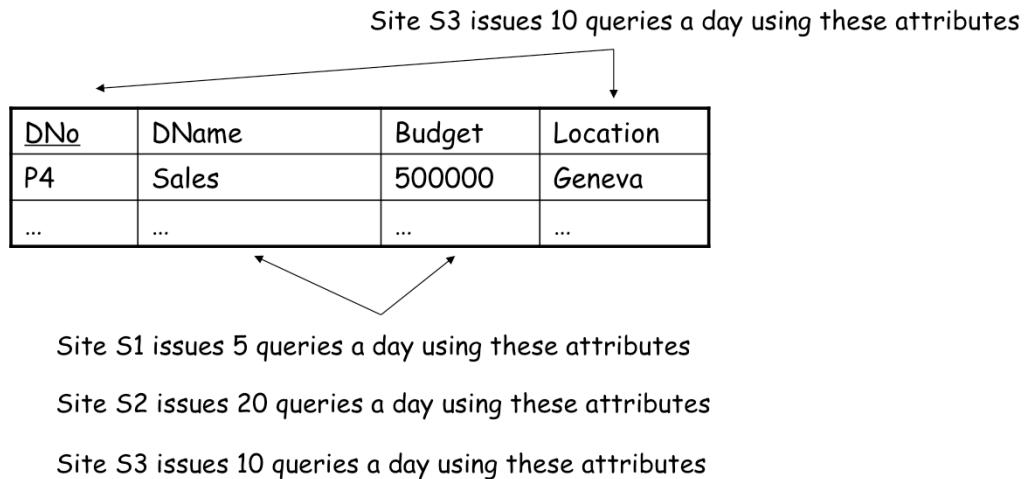
In general, different DHFs can be obtained if the same relation is related to multiple relations through a foreign key relationship. In that case a decision has to be taken from which relation the fragmentation should be derived, since a fragmentation according to different primary fragmentation would make no sense: it would not be possible to keep the tuples in the derived fragments together with the corresponding primary fragments if they are moved to different sites. Therefore the DHF is chosen, which is induced by the relation that is expected to be used most frequently together with the relation for which the DHF is generated.

## Question

- When deriving a horizontal fragmentation for relation  $S$  from a horizontally fragmented relation  $R$ 
  1. Some primary key attribute in  $R$  must be a foreign key in  $S$
  2. Some primary key attribute in  $S$  must be a foreign key in  $R$
  3. Both are possible

### 3. Vertical Fragmentation

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



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 where 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

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

DEPARTMENT

- 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.

- If query  $q_k$  accesses attribute  $a_i$ , we set  $Q_{ki} = 1$ , 0 otherwise

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

Matrix Q

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

One of the consequences from that observation is that it 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. With  $Q_{ki}$  we describe 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)
  - $\text{aff}(\text{Dname}, \text{Budget}) = 0$
  - $\text{aff}(\text{Budget}, \text{Location}) = 20+15 = 35$
  - $\text{aff}(\text{Dname}, \text{Location}) = 15$
  - $\text{aff}(\text{Dname}, \text{Dname}) = 15$
  - $\text{aff}(\text{Budget}, \text{Budget}) = 65$

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

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

	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

Matrix A describing attribute affinity

©2011, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Schema Fragmentation - 36

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

## Maximize Affinity with Neighbors in A

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

scalar product among columns measures similarity of access

$$\text{bond}(\text{Budget}, \text{Dname}) = 15 * 35$$

$$\text{bond}(\text{Dname}, \text{Location}) = 15 * 15 + 15 * 50$$

global neighbor affinity:  
1500

after swapping columns  
global neighbor affinity:  
4550

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

cluster = vertical fragment

Schema Fragmentation - 37

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: if 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.

## Questions

- 60 is the
  - 1. Access frequency from S1 to A
  - 2. Bond among attribute B and C
  - 3. Affinity among attribute B and C
  - 4. Global neighbor affinity among all attributes
- The affinity among A and B is
  - 1. 2
  - 2. 50
  - 3. 9800
  - 4. 19700

	A	B	C
q1	0	1	1
q2	1	1	1
q3	1	1	0

	S1	S2	S3
q1	10	0	20
q2	0	10	20
q3	10	10	0

	A	B	C
A	60	50	30
B	50	100	60
C	30	60	40

## 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:  $n \times n$  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  $A_k$  between  $A_i$  and  $A_j$ :

$$\text{cont}(A_i, A_k, A_j) = \text{bond}(A_i, A_k) + \text{bond}(A_k, A_j) - \text{bond}(A_i, A_j)$$

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

scalar product

©2011, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Schema Fragmentation - 39

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	

$\text{cont}(\_, A2, A1) =$   
 $\text{bond}(\_, A2) + \text{bond}(A1, A2) - \text{bond}(\_, A1) =$   
 $0 + 225 - 0 = 225$

	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	

$\text{cont}(A1, A2, A3) =$   
 $\text{bond}(A1, A2) + \text{bond}(A2, A3) - \text{bond}(A1, A3) =$   
 $225 + 890 - 4410 = -3295$

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	

$\text{cont}(A3, A2, \_) =$   
 $\text{bond}(A3, A2) + \text{bond}(A2, \_) - \text{bond}(A3, \_) =$   
 $890 + 0 - 0 = 890$

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

The diagram illustrates the process of row reordering. On the left, the original table has rows labeled A1, A2, A3, and A4. On the right, after reordering, the rows are labeled A1, A3, A2, and A4. A large blue arrow points from the original table to the reordered table, with the text "reordering rows" written above it.

	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 ?

The diagram shows a curved arrow pointing from the original table to a new table. In the new table, attributes A1, A3, and A2 are highlighted with a green background, while attribute A4 is highlighted with a yellow background. This visual representation indicates that A1, A3, and A2 are grouped together, and A4 is a separate cluster.

	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

Schema Fragmentation - 41

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

From the resulting affinity 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: 78  
 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

©2011, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

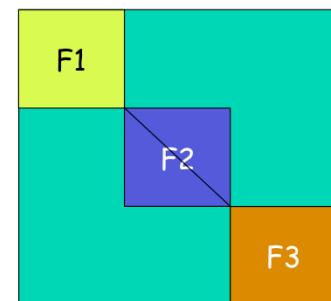
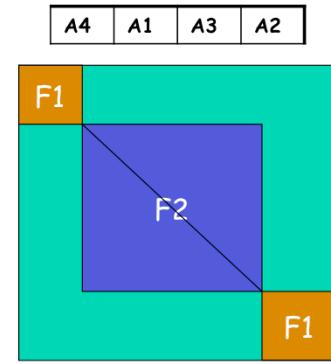
Schema Fragmentation - 42

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 simultaneously
  - try  $1, 2, \dots, m$  splitting points and select the best
  - cost  $O(2^m)$



©2011, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Schema Fragmentation - 43

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

## Questions

- Which statement on the BEA algorithm is wrong?
  1. The result depends on the ordering of the attributes in the input
  2. It produces the optimal clustering for vertical partitioning
  3. The number of iterations is linear in the number of attributes
  4. The complexity of the algorithm is linear in the size of the input
- 30 is the number of accesses
  1. To VF1 only
  2. To VF2 only
  3. VF1 and VF2

	A	B	C
q1	0	1	1
q2	1	1	1
q3	1	1	0

	S1	S2	S3
q1	10	0	20
q2	0	10	20
q3	10	10	0

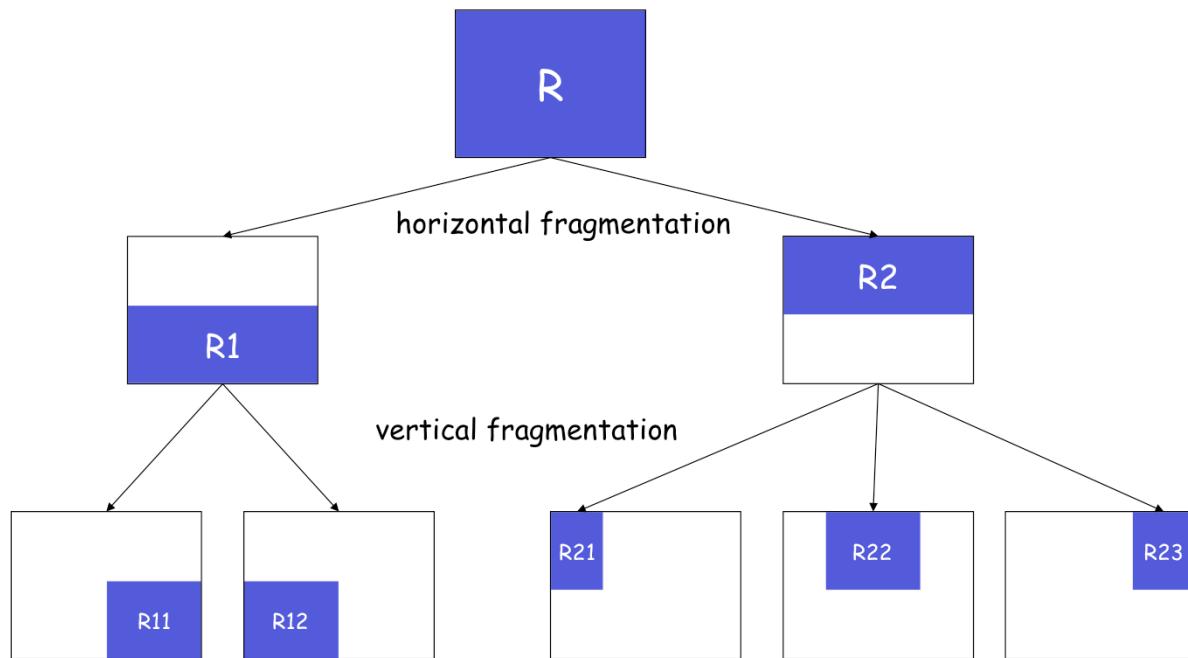
	A	B	C
A	60	50	30
B	50	100	60
C	30	60	40

VF1
VF2

©2011, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Schema Fragmentation - 45

## Hybrid Fragmentation



©2011, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Schema Fragmentation - 46

To complete the picture we discuss 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  $F_1, \dots, F_n$
  - given sites  $S_1, \dots, S_m$
  - given applications  $A_1, \dots, A_k$
  - 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 hard
  - 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.

## 5. Discussion

- Fragmentation in relational DB products
  - Support for horizontal and vertical fragmentation
  - Typically for distribution over computing clusters
  - Manual configuration and automated design tools
  - MySQL, DB2, Oracle etc.
- Fragmentation is being heavily used for distributed RDF databases (Web data management)
  - Both vertical and horizontal fragmentation
- (horizontal) Fragmentation is becoming extremely important in today's information systems
  - Cloud computing
  - Heavy use of horizontal fragmentation
  - Map-Reduce
  - Hadoop, Hbase, SOLR

## Some questions to study

- Why should a relational database be fragmented ?
- At which phase of the database lifecycle is fragmentation performed ?
- What are the alternative approaches to fragment relations ?
- Under which conditions is a fragmentation considered correct ?
- In which environments would replication be an appropriate alternative to fragmentation ?
- How are horizontal fragments specified ?
- When are two fragments considered to be accessed in the same way ?
- What is the difference between simple and minterm predicates ?
- How is relevance of simple predicates determined ?
- Is the set of predicates selected in the MinFrag algorithm monotonically growing ?
- Why are minterm predicates eliminated after executing the MinFrag algorithm ?
- 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)
  - DeWitt, D. and Gray, J. 1992. Parallel database systems: the future of high performance database systems. *Commun. ACM* 35, 6 (Jun. 1992), 85-98.
  - Abadi, D. J., Marcus, A., Madden, S. R., and Hollenbach, K. 2007. Scalable semantic web data management using vertical partitioning, Very Large Data Bases. VLDB Endowment, 411-422.
- Examples of automated DB design tools
  - IBM DB2: Zilio, D. C., Rao, J., Lightstone, S., Lohman, G., Storm, A., Garcia-Arellano, C., and Fadden, S. 2004. DB2 design advisor: integrated automatic physical database design. *Very Large Data Bases - Volume 30*, 1087-1097.
  - Microsoft SQLServer: Agrawal, S., Narasayya, V., and Yang, B. 2004. Integrating vertical and horizontal partitioning into automated physical database design. *SIGMOD '04*,