Applying Vertical Fragmentation Techniques in Logical Design of Multidimensional Databases

DEIS
University of Bologna
ITALY

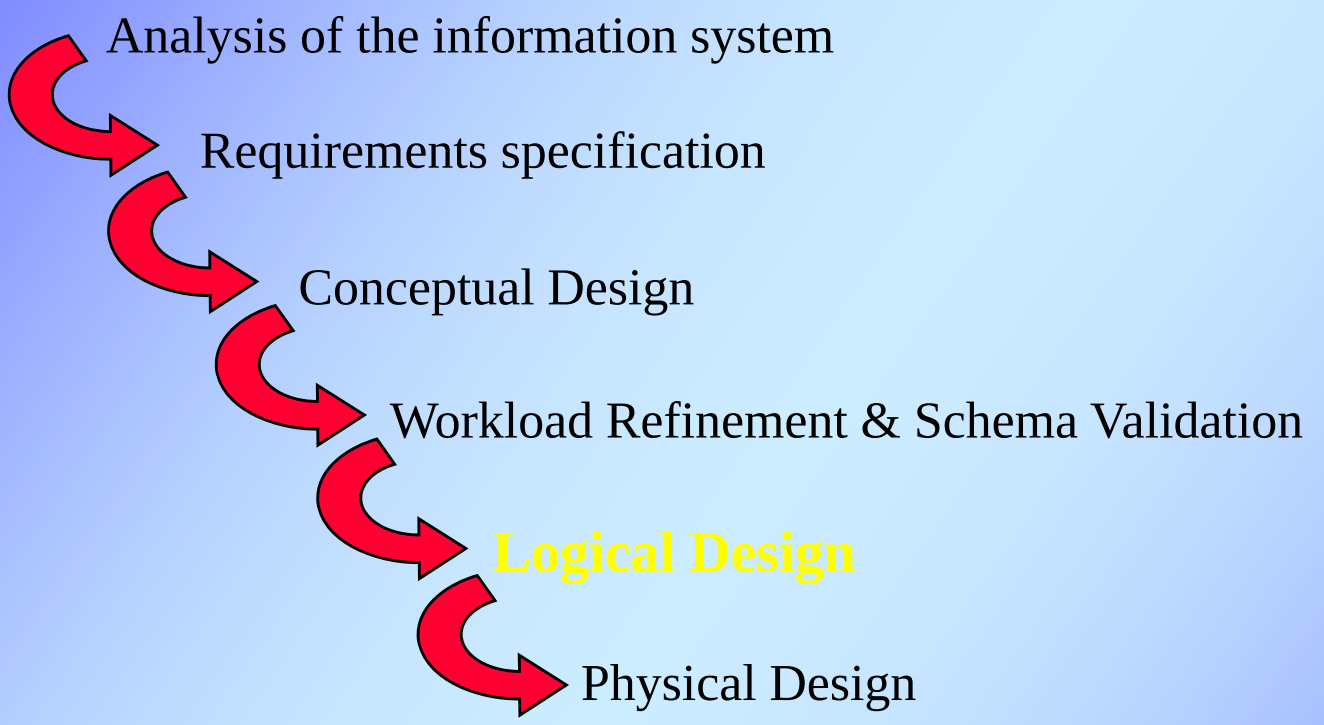
Matteo Golfarelli *Stefano Rizzi*

Outline

- **Vertical Fragmentation**
- **Problem Statement**
- **A Branch-and-Bound approach**
- **Experimental Results**
- **Conclusions**

Design Phases

The methodology for DW design basically includes the following phases:



Logical design entails the choice of the logical structure of data and those optimisations related to such structure:

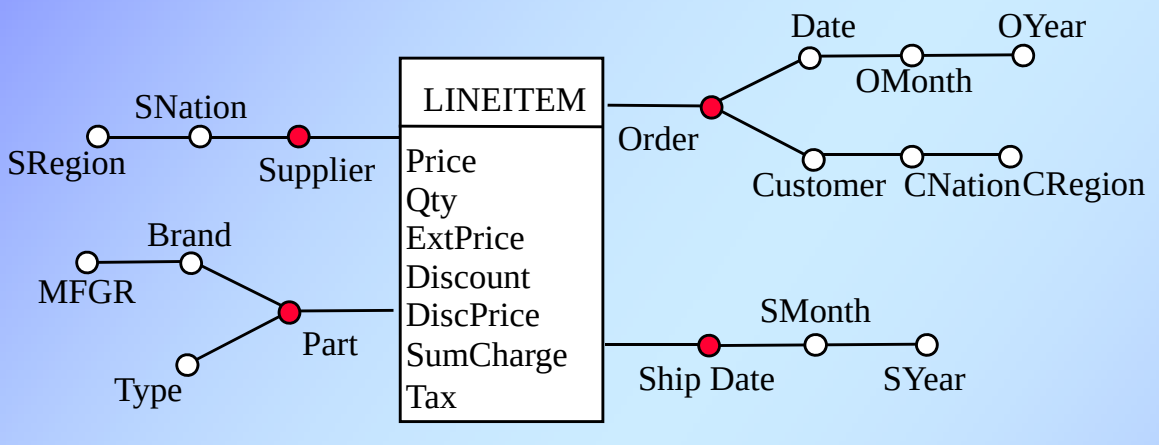
- View materialization
- **Vertical fragmentation**
- Horizontal fragmentation

We consider only DWs implemented on relational DBMSs, in which data are usually modelled using the star/snowflake schema.

Multidimensional cubes

A ***multidimensional database*** (MD) is a set *FS* of multidimensional cubes $f_1 ... f_n$ each characterized by:

- A set of dimensions *Patt*(*f*)
- A set of measures *Meas*(*f*)
- A set of attributes *Attr*(*f*)



The attributes in *Attr*(*f*) are related into a directed acyclic graph by a set of functional dependencies; by writing $a_i \rightarrow a_j$ we mean that $a_i \in Attr(f)$ determines $a_j \in Attr(f)$.

Def. Given a cube *f*, an ***aggregation pattern*** on *f* is a set $P \subset Attr(f)$ such that no functional dependency exists between each pair of attributes in *P*.

We say that a pattern P_i is ***coarser*** than a pattern P_j if for each attribute $a_i^* \in P_i$, an attribute $a_j^* \in P_j$ such that $a_j \rightarrow a_i$ always exists.

{SNation,Customer,ShipDate} $\rightarrow$ {SRegion,Customer}

The workload

The workload *QS* is defined by a set of pairs (q_i, η_i), where q_i denotes a query and η_i its expected frequency.

A query q can be characterized by:

- ***Its pattern:*** *Patt*(q)
- ***The set of measures it requires:*** *Meas*(q)
- ***Its selectivity:*** *Sel*(q) defined as the ratio between the number of tuples returned by q and the cardinality of the view at pattern *Patt*(q)

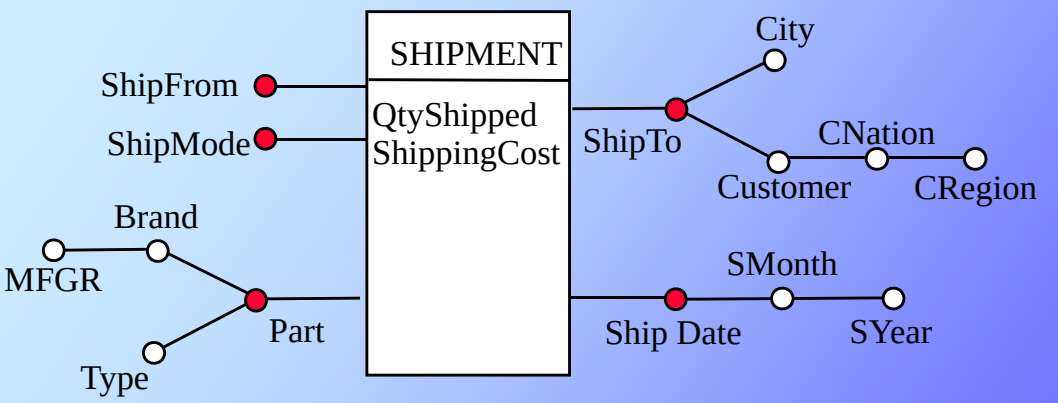
Example: "*The total quantity sold for each medium polished part ordered from each American Supplier*" is characterized by:

Patt(q)={Supplier,Part} *Meas*(q)={Qty}

We also consider ***drill-across queries***: queries that require measures taken from distinct, though related, cubes.

Example: "*The total cost paid by the customer of each region to receive each part*". It is a drill-across query since:

1. It requires measures from both *LineItem* and *Shipment*: {DiscountPrice, ShippingCost}
2. Its aggregation pattern {CRegion,Part} is common to both the cubes



View Materialization

- Given a cube *f*, each pattern on *f* determines a candidate view for materialization.

- We assume that a view materialization step has been carried out on each cube of the MD and that V is the set of views (both primary and secondary) selected for materialization.

- In the view materialization step, drill-across queries are substituted by their “projections” on the cubes involved.

Vertical Fragmentation of Views

The main motivations for adopting vertical ***fragmentation*** in DW are as follows:

1. Measures included in the same view may be often requested separately or may be not requested at all. The system performance will be increased by ***partitioning*** such views into smaller tables containing only those measures appearing together within queries.

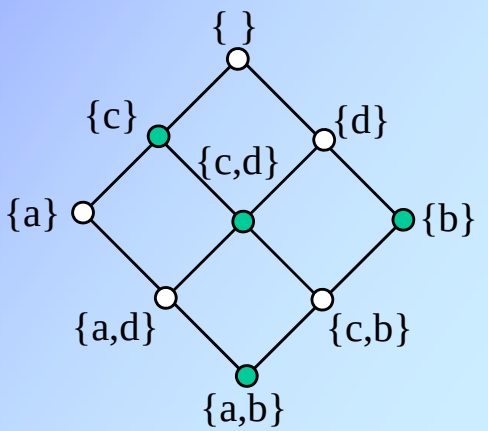
2. Drill-across queries require data from views defined on different cubes. The access cost may be decreased by ***unifying*** two or more views on the same pattern into a larger table where all the measures required are stored together.

Besides, it should be noted that:

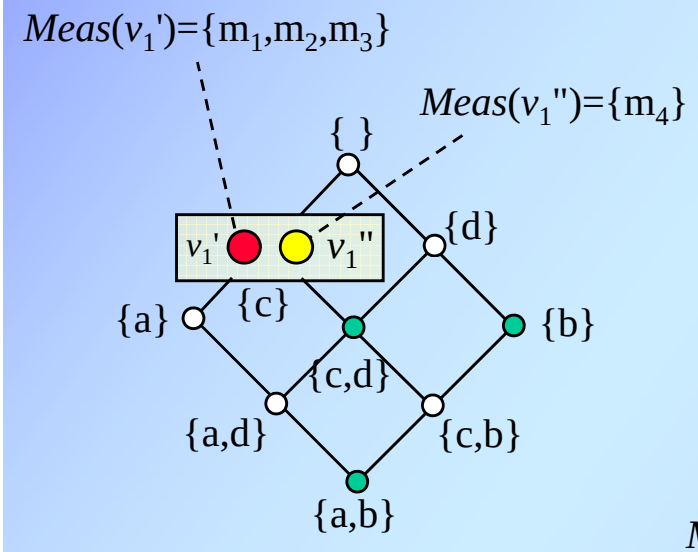
- The storage overhead due to surrogate keys replication:
  - decreases for coarsen aggregation patterns where some dimensions are hidden.
  - becomes negligible if also derived and support measures are considered.
- The effectiveness of fragmentation for MDs may be higher than for operational non-redundant databases since, while in the latter case it is known a priori on which table(s) each query will be executed, in MDs the presence of redundant views makes multiple solutions possible.

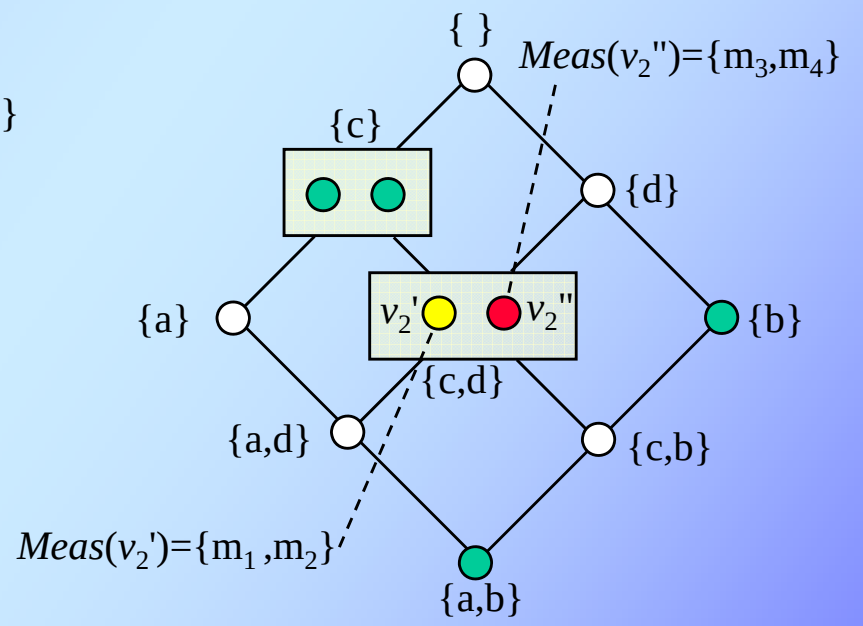
Partitioning of Views

Let us consider a cube f such that $Patt(f)$={a,b}, $Meas(f)$={m_1,m_2,m_3,m_4}, $Attr(f)$={a,b,c,d} and such that the functional dependencies determine the following lattice structure



$QS=\{q_1, q_2\}$
$Patt(q_1)=Patt(q_2)=\{c\}$
$Meas(q_1)=\{m_1,m_2,m_3\}$
$Meas(q_2)=\{m_3,m_4\}$





q_1 will be executed at pattern {c} on v_1', while the best choice for q_2 depends on the trade-off between reading less measures (v_2") and accessing less tuples (v_1'+ v_1").

Unification of Views

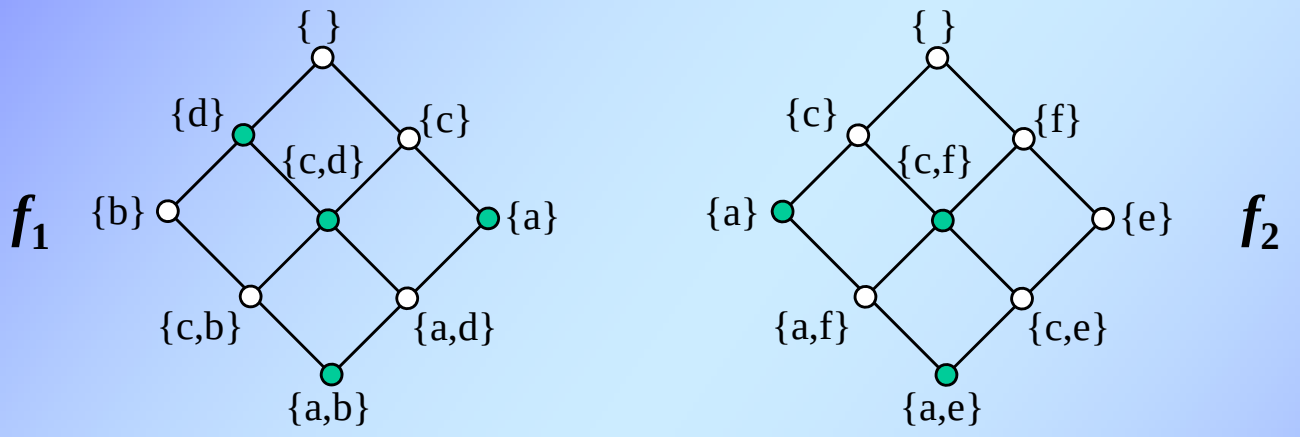
Let us consider two cube f_1, f_2 such that

$Patt(f_1)$={a,b} $Patt(f_2)$={a,e}

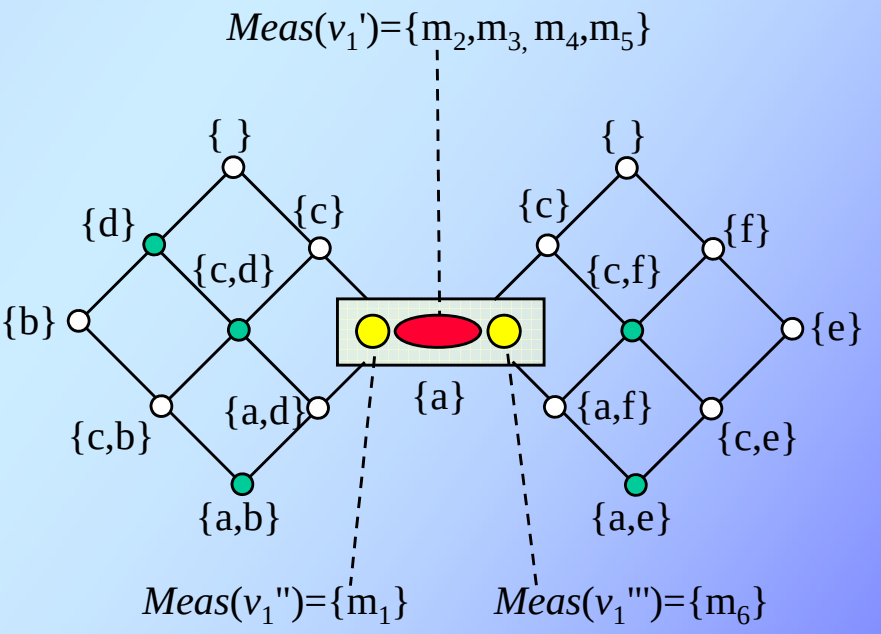
$Meas(f_1)$={m_1,m_2,m_3} $Meas(f_2)$={m_4,m_5,m_6}

$Attr(f_1)$={a,b,c,d} $Attr(f_2)$={a,e,c,f}

and such that the functional dependencies determine the following lattice structures



Given a query q_1 such that $Patt(q_1)$ ={a} and $Meas(q_1)$ = {m_2 ,m_3 ,m_4 ,m_5}, the following optimisation is possible:



Problem statement I

Def.: Given a cube $f \in FS$, we define ***minterms*** the largest subset of measures which appear together in at least one query of *QS* and do not appear separately in any other query in *QS*. We denote with *MS* the set of all minterms of *FS*.

Example: *f=LineItem QS*=$\{q_1,q_2\}$

Meas(q_1)={Price,Qty,ExtPrice,Discount}
Meas(q_2)={Price,Qty,DiscountPrice,SumCharge,Tax}

MS(*LineItem*)={ {Price,Qty},{DiscPrice,SumCharge,Tax},
{ExtPrice,Discount}}

Def.: Given the set of related cubes *FS*, we define a ***term*** as a set of measures which either is a minterm of a cube in *FS* or is the union of the minterms, even from different cubes in *FS*, within a set $\overline{MS}$ such that

$$\forall M_k \in \overline{MS} \exists q_i \in QS, \exists M_j \in \overline{MS},\ j \neq k; M_k \subset Meas(q_i) \wedge M_j \subset Meas(q_i)$$

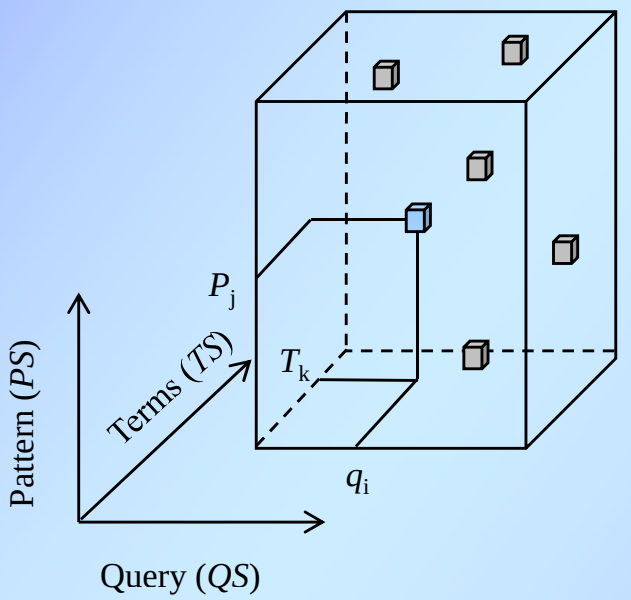
We denote with *TS* the set of terms for *FS*.

Example: In the example above:

TS=*MS*(LineItem)∪{{Price,Qty,ExtPrice,Discount},{Price,Qty,DiscPrice,
SumCharge,Tax},{Price,Qty,ExtPrice,Discount,DiscPrice,SumCharge,Tax}}

Problem statement II

Given *FS*, a solution to the fragmentation problem is encoded by a ***fragmentation array***, i.e., a binary array ***C*** with three dimensions corresponding to, respectively, the queries $q_i \in QS$, the patterns $P_j \in PS$ and the terms $T_k \in TS$.



A 1 in cell C_{ijk} denotes that, when answering query q_i, the measures in $Meas(q_i) \cap T_k$ will be obtained from view v_{jk} characterized by $Meas(v_{jk}) = T_k$ and $Patt(v_{jk}) = P_j$.

We denote with *VS'* the set of *fragments* defined by ***C***:

$$VS' = \left\{ v_{jk} ; \exists j,k; \sum_{q_i \in QS} C_{ijk} \geq 1 \right\}$$

We will call ***lost minterms*** those not included in any term generating a fragment.

Problem statement III

The fragmentation encoded by ***C*** is ***feasible*** if the following constraints are satisfied:

1) For each query, every measure required must be obtained from exactly one fragment (**non ambiguous query execution**):

$$\forall q_i \in QS, \forall m \in Meas(q_i) \left(\sum_{P_j \in PS; Patt(q_i) \geq P_j} \sum_{T_k \in TS; m \in T_k} C_{ijk} = 1 \right)$$

2) For each pattern, each measure must belong to at most one fragment (**non redundant fragmentation**):

$$\forall P_j \in PS, \forall T_k, T_h \in TS; (k \neq h \wedge T_k \cap T_h \neq \varnothing) \left(\left(\sum_{q_i \in QS} C_{ijk} = 0 \right) \vee \left(\sum_{q_i \in QS} C_{ijh} = 0 \right) \right)$$

3) Each fragment in *VS'* must be a fragmentation of one or more views in *VS* (**consistency with the views materialized**):

$$\forall P_j \in PS, \forall T_k \in TS; \sum_{q_i \in QS} C_{ijk} \geq 1 (\forall m \in T_k \exists v \in VS; Patt(v) = P_j \wedge m \in Meas(v))$$

An example

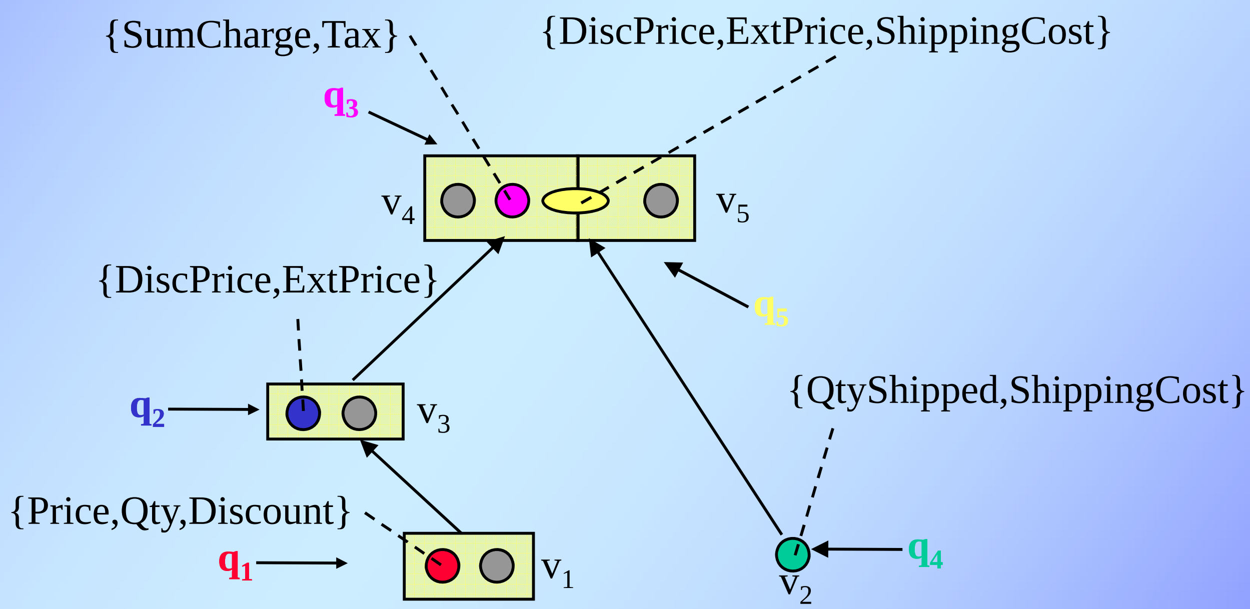
FS = {*LineItem*, *Shipment*}

$QS = \{q_1, q_2, q_3, q_4, q_5\}$

Meas(q_1)={Price,Qty,Discount}
Meas(q_2)={ExtPrice,DiscPrice}
Meas(q_3)={SumCharge,Tax}
Meas(q_4)={QtyShipped,ShippingCost}
Meas(q_5)={ExtPrice,DiscPrice,ShippingCost}

Patt(q_1)={ShipDate}
Patt(q_2)={Part,Customer}
Patt(q_3)={Part,CNation}
Patt(q_4)={CNation,MFGR,ShipDate}
Patt(q_5)={Brand,Nation}

Patt(v_1)={Part,Supplier,Order,ShipDate}
Patt(v_2)={Part,ShipFrom,ShipTo,ShipDate}
Patt(v_3)={Part,Customer}
Patt(v_4)=*Patt*(v_5)={Part,CNation}



TS \ PS	q_1: *Patt*(*LineItem*)	*Patt*(*Shipment*)	*Patt*(v_3)	*Patt*(v_4) = *Patt*(v_5)	q_2: *Patt*(*LineItem*)	*Patt*(*Shipment*)	*Patt*(v_3)	*Patt*(v_4) = *Patt*(v_5)	q_3: *Patt*(*LineItem*)	*Patt*(*Shipment*)	*Patt*(v_3)	*Patt*(v_4) = *Patt*(v_5)	q_4: *Patt*(*LineItem*)	*Patt*(*Shipment*)	*Patt*(v_3)	*Patt*(v_4) = *Patt*(v_5)	q_5: *Patt*(*LineItem*)	*Patt*(*Shipment*)	*Patt*(v_3)	*Patt*(v_4) = *Patt*(v_5)
{Discount,Qty,Price}	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{DiscPrice,ExtPrice}	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
{SumCharge,Tax}	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
{ShippingCost}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{QtyShipped}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{ShippingCost,QtyShipped}	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
{DiscPrice,ExtPrice,ShippingCost}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

The Cost Function

Among all the feasible solutions to the fragmentation problem, we are interested in the one which minimizes the cost for executing the workload.

The cost function we propose abstracts from any assumptions on the access paths, being based on the number of disk pages in which the tuples of interest for a given query are stored.

$$cost(q_i, \boldsymbol{C}) = \sum_{P_j \in PS,\, T_k \in TS} \Phi\left(sel(q_i) \cdot ns(P_j), \left\lceil \frac{ns(P_j)}{\beta_{jk}} \right\rceil \right) \cdot \boldsymbol{C}_{ijk}$$

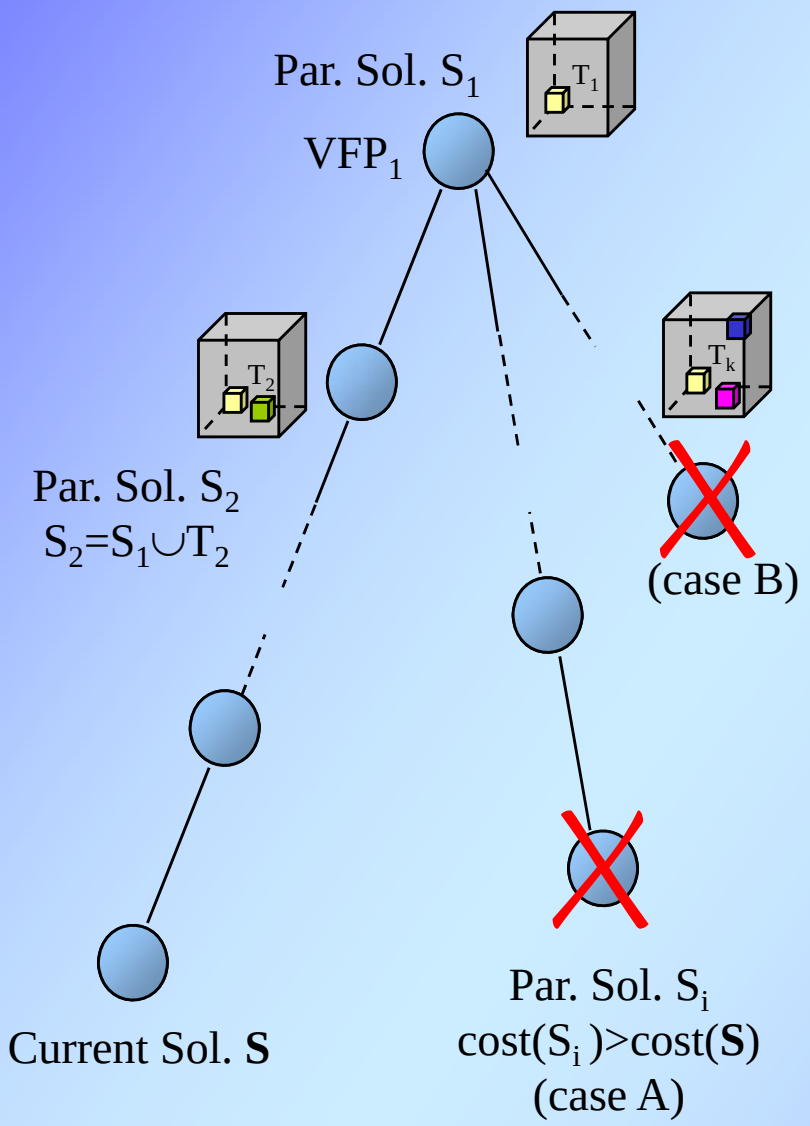
- $sel(q_i) \cdot ns(P_j)$ is the number of tuples returned by q_i
- β_{jk} is the number of tuples per disk page for fragment v_{jk}
- $\Phi(.....)$ is the expected number of pages in which the tuples involved in q_i are stored, estimated with the Cardenas formula Φ.

The total cost for the workload turns out to be:

$$tcost(QS, \boldsymbol{C}) = \sum_{q_i \in QS} \eta_i cost(q_i, \boldsymbol{C})$$

The branching rule

We propose a branch-and-bound approach that solves the vertical fragmentation problem (VFP) optimally.



- ✓ **Each node of the graph is associated to two fragmentation cubes: the first (C) codes the partial solution while the second (D) stores the available fragments;**
- ✓ **Node creation entails adding a new fragment to the current solution, moving it from D to C;**
- ✓ **When a new fragment is added to the current solution, several fragments in D become unfeasible and must be deleted.**

- ✓ **Current node branching is stopped when:**
  - ☞ **The current solution cost is higher then that of the best solution (case A);**
  - ☞ **Due to deletion, the fragments in D do not allow a feasible solution to be found anymore (case B).**

Other Branch-and-bound ingredients

- ***Subproblem selection rule:*** for choosing the next (most promising) subproblem to be processed.

The element C_{ijk} chosen is the one for which $ns(P_j)$ is minimum and $Meas(q_i) \cap T_k$ has maximum cardinality.

- ***Relaxation:*** an easier problem VFR_α whose solution bounds that of VFP_α.

We relax VFP_α by removing constraint (2): in VFR_α, some measures may be replicated in two or more fragments defined on the same pattern.

- ***Lower bound:*** a procedure to calculate the cost of the relaxation VFR_α

One set covering problem for each q_i is calculated. Since in solving VFR_α the number of eligible fragments is higher than that for VFP_α, the cost of VFR_α will be lower or equal to that of VFP_α.

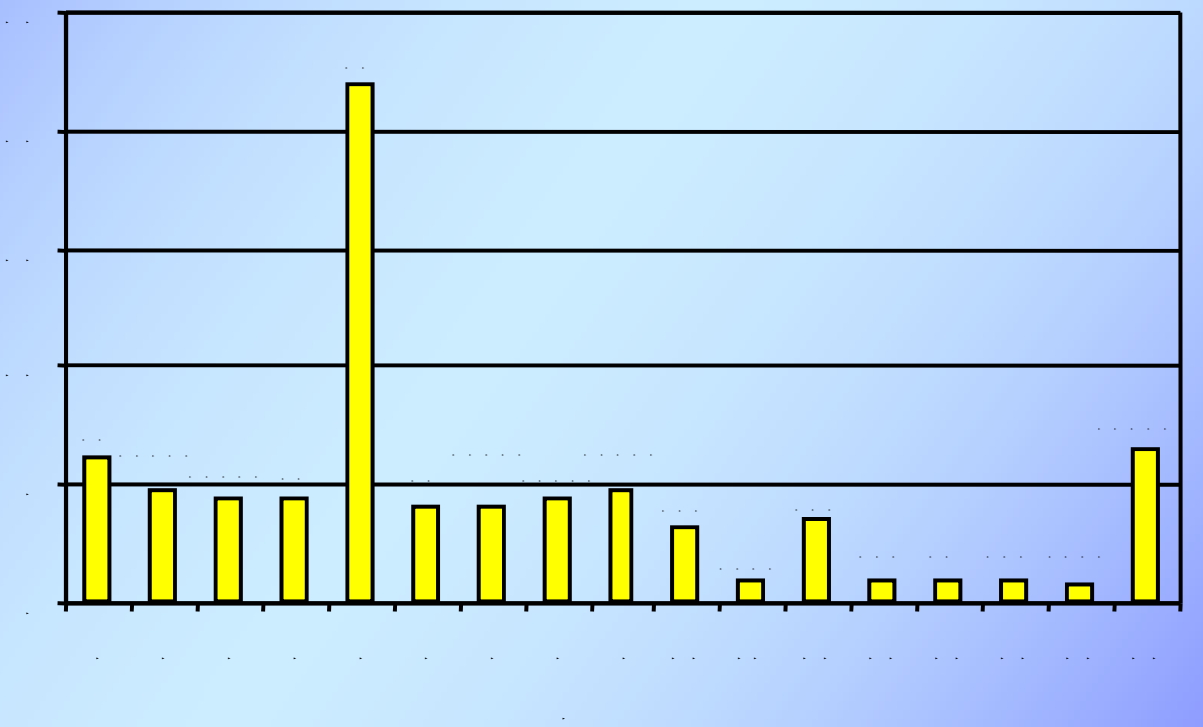
Experimental results I

Tests have been carried out on the TPC-D benchmark:

LineItem: 6.000.000 tuples
PartSupplier: 800.000 tuples → 1 Gb space

17 queries
2GB Space constraint → 11 views

The fragmentation algorithm determined 14 fragments and 9 lost minterms. Indexes on all attributes belonging to keys have been added.

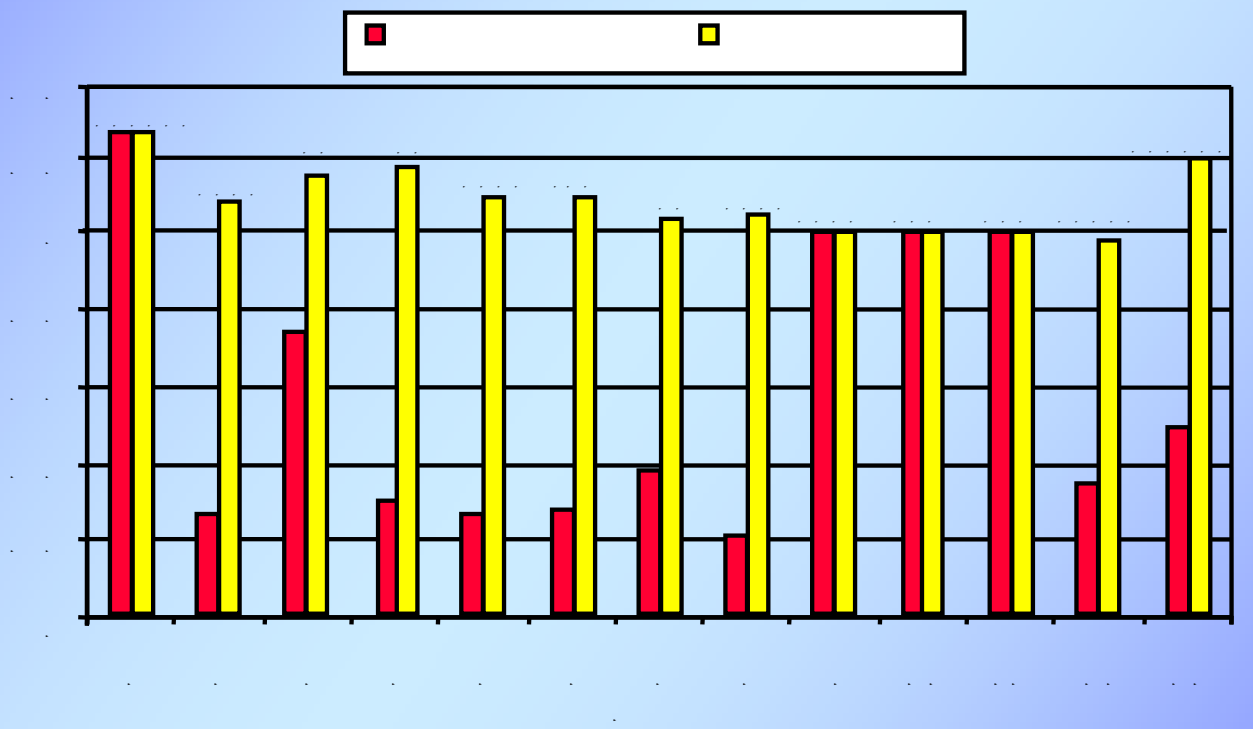


Overall, fragmentation decreases the workload cost from 265.904 to 59.986 pages (more than 4 times!).

Experimental results II

The required space depends on the designer choice:

- ***Materialize lost minterm:***
  - The required space is increased
  - The solution is more general
- ***Do not materialize lost minterm:***
  - The required space decreases
  - The solution is more suited for the specific workload
  - Given a space constraint, further optimisations are possible



Unfragmented solution:	368.840 pgs	
Fragmenting with lost minterm:	442.097 pgs	(+19.8%)
Fragmenting without lost minterm:	306.042 pgs	(-17%)

# Queries	# Views	Subproblems	Time (Min.)
17	8	2775	1
25	12	4439	2
30	13	348925	30
35	14	51099	12
40	16	403420	75

Conclusions

Vertical fragmentation allows the reduction of the cost for executing an expected workload in MDs.

- The strong data redundancy increases the effectiveness of the technique
- The algorithm proposed:

  1) Finds out the optimal solution
  2) Also considers drill-across queries

Future works

- Adopt approaches different from branch-and-bound in order to allow bigger problems to be solved.
- Apply the proposed formalization to the horizontal partitioning problem