Implementing Data Cubes Efficiently

Written by:

Venky Harinarayan, Anand Rajaraman, Jeffrey D. Ullman

Stanford University

Published in:

Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996

Awarded SIGMOD Best Paper Award 1996

Presented by:

Lars K. Schunk

*) Special Interest Group on Mangement of Data

Overview of the Presentation

- Introduction and Motivating Example
- The Lattice Framework
- Query-Cost Model
- The Greedy Algorithm
- Performance Guarantee
- Conclusion
- Paper Evaluation

Operational Databases vs. Data Warehouses

Operational databases:

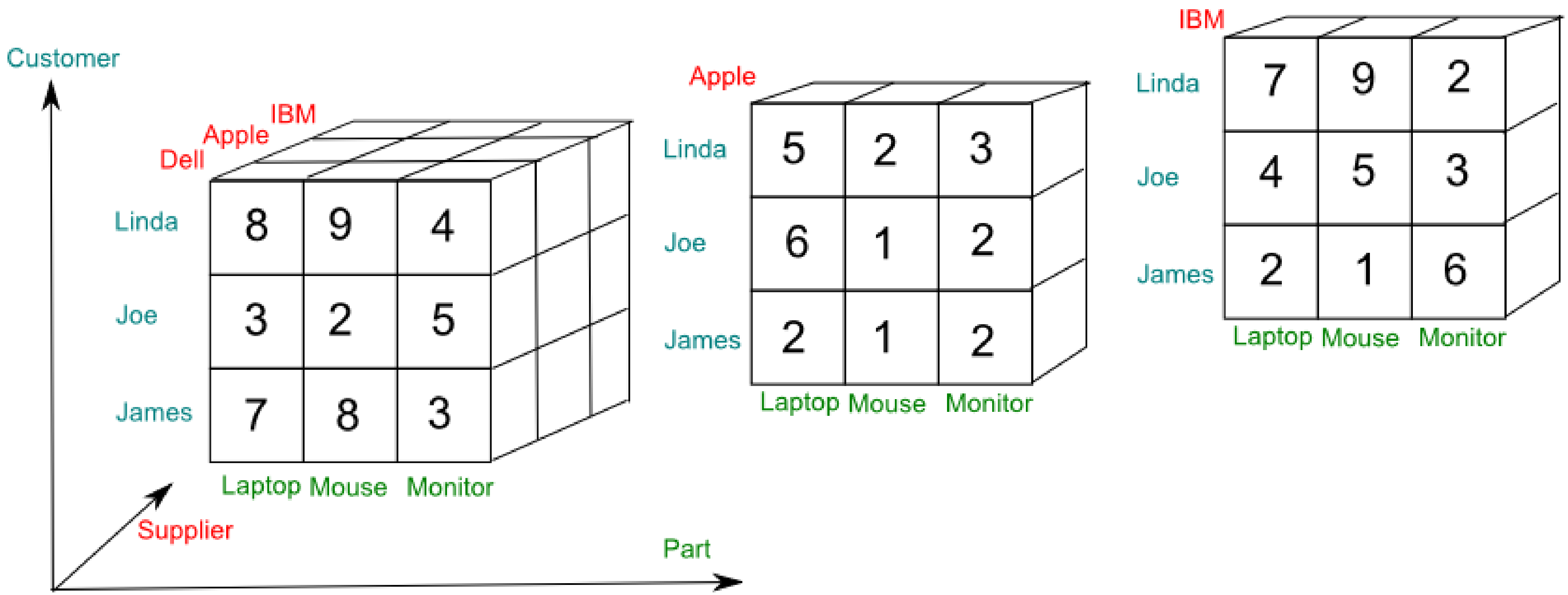
- State information

Data warehouses:

- Historical information
- Very large and grow over time
- Used for identifying trends

Data Warehouse Cubes

- Data are presented as multidimensional data cubes
- Users explore the cubes and discover information



Each cell (*p*, *s*, *c*) stores the sales of **part *p*** that was bought from **supplier *s*** and sold to **customer *c***

Aggregations

Consolidated sales

- Add ”ALL” value to the domain of each dimension
- Results in dependent cells

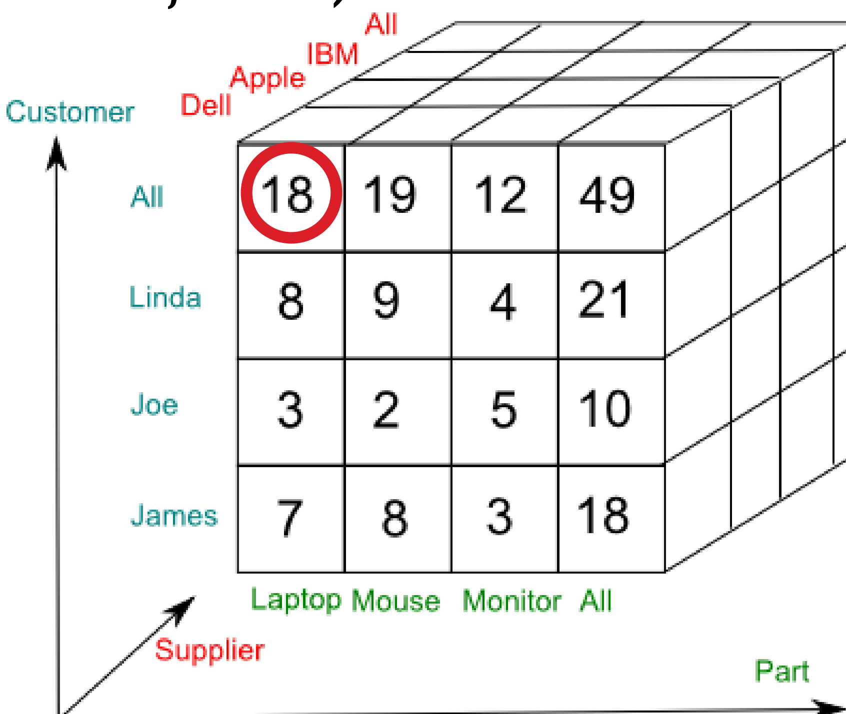
General example

- What is the total sales of a given part *p* from a given supplier *s*?
- Look up value in cell (*p*, *s*, ALL)

All customers

Aggregations (Example)

- **Specific example:** What is the total sales of laptops from Dell, i.e., what is in cell (laptop, Dell, ALL)?



(laptop, Dell, ALL) = 7 + 3 + 8 = 18

The number of dependent cells is usually a large fraction of the total number of cells in the cube, e.g., 70 %

The Problem: Query Performance in Data Warehouses

- Queries are very complex
- Make heavy use of aggregations
- Take very long to complete
- Limit productivity

Solution idea:

- Materialize query results, i.e., precompute query results and store them on disk

Three Alternatives

Materialize the whole data cube

- Best query response time
- Not feasible for large data cubes

Materialize nothing

- No extra space required beyond that for the raw data
- We need to compute every cell on request

Materialize only part of the data cube (our solution)

- Trade-off between space required and query response time
- Which cells should be materialized?

Which cells should be materialized?

Relevant questions

- Frequently asked queries?
- Not-so-frequently asked queries that can be used to answer many other queries quickly?

Solution

- This paper presents an algorithm for picking the right set of query results to materialize

Representing Data Cubes

- The data cube can be represented with a simple table
- The **Sales** Table:

Part	Supplier	Customer	Sales
Laptop	Apple	James	2
Laptop	Apple	Joe	6
Laptop	Apple	Linda	5
Laptop	Dell	James	7
Laptop	Dell	Joe	3
Laptop	Dell	Linda	8
Laptop	IBM	James	2
Laptop	IBM	Joe	4
Laptop	IBM	Linda	7
Monitor	Apple	James	2
Monitor	Apple	Joe	2
Monitor	Apple	Linda	3

↓ 27 rows … ↓

Only independent cells are stored in the table

Representing Data Cubes

- Dependent cells are computed from independent cells
- We use SQL queries on the Sales table

- **Example:** Compute cell (laptop, Dell, ALL)

```
SELECT Part, Supplier, SUM(Sales) AS Sales
FROM Sales
WHERE Part = 'Laptop' and Supplier = 'Dell'
GROUP BY Part, Supplier
```

	Part	Supplier	Sales
1	Laptop	Dell	18

Overview of the Presentation

- Introduction and Motivating Example
- The Lattice Framework
- Query-Cost Model
- The Greedy Algorithm
- Performance Guarantee
- Conclusion
- Paper Evaluation

Cell Organization

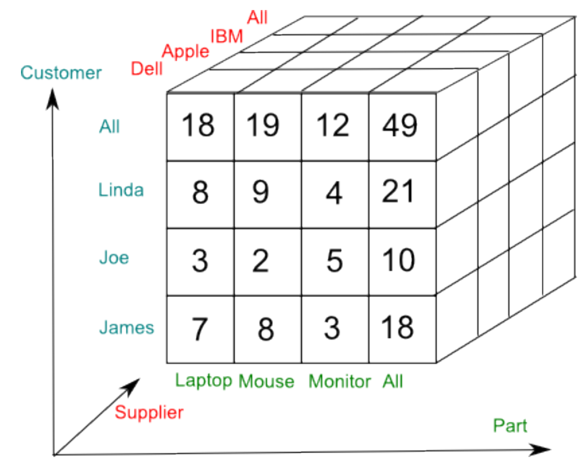
- Cells are organized into sets based on the positions of ALL in their addresses
- For example, all cells with address (p, s, c) = (_, ALL, _) are placed in the same set.
- Each set corresponds to an SQL query result
- A set of cells ≡ a query result ≡ **a view**

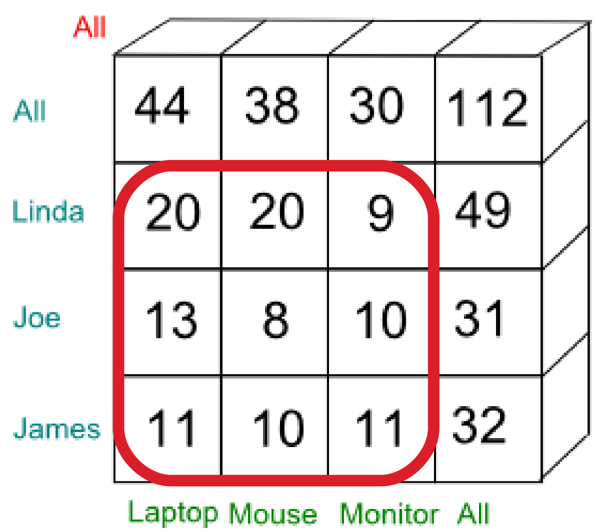
Cell Organization (Example)

part, customer = (_, ALL, _):

```
SELECT Part, Customer, SUM(Sales) AS Sales
FROM Sales
GROUP BY Part, Customer
```

	Part	Customer	Sales
1	Laptop	James	11
2	Monitor	James	11
3	Mouse	James	10
4	Laptop	Joe	13
5	Monitor	Joe	10
6	Mouse	Joe	8
7	Laptop	Linda	20
8	Monitor	Linda	9
9	Mouse	Linda	20





Eight Views

3 dimensions give 8 possible groupings.

The corresponding views:

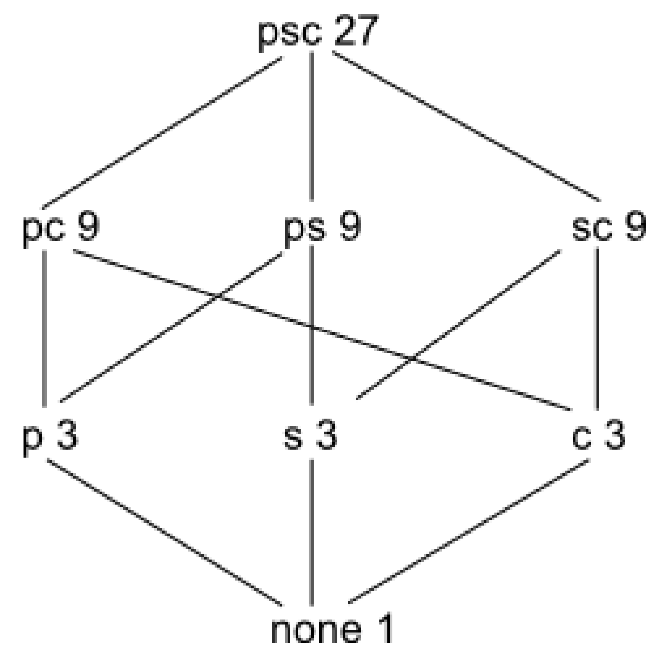
```
5.part, supplier, customer (27 rows)
6.part, customer (9)
7.part, supplier (9)
8.supplier, customer (9)
9.part (3)
10.supplier (3)
11.customer (3)
12.none (1)
```

Lattice Representation of Views

3 dimensions give 8 possible groupings.

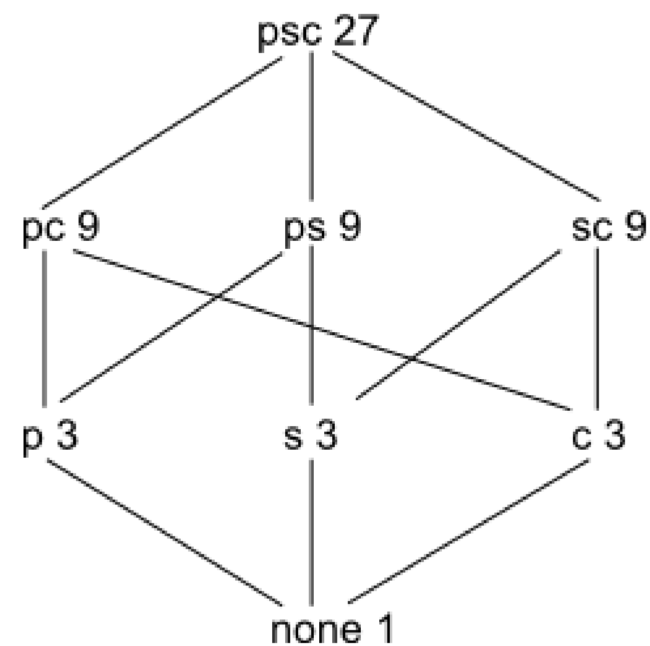
The corresponding views:

```
5.part, supplier, customer (27 rows)
6.part, customer (9)
7.part, supplier (9)
8.supplier, customer (9)
9.part (3)
10.supplier (3)
11.customer (3)
12.none (1)
```



The Dependence Relation $\preccurlyeq$

- Consider two queries Q_1 and Q_2.
- $Q_1 \preccurlyeq Q_2$ if Q_1 can be answered using only the results of Q_2
- Q_1 is *dependent* on Q_2
- There is a path downward from Q_2 to Q_1 iff $Q_1 \preccurlyeq Q_2$



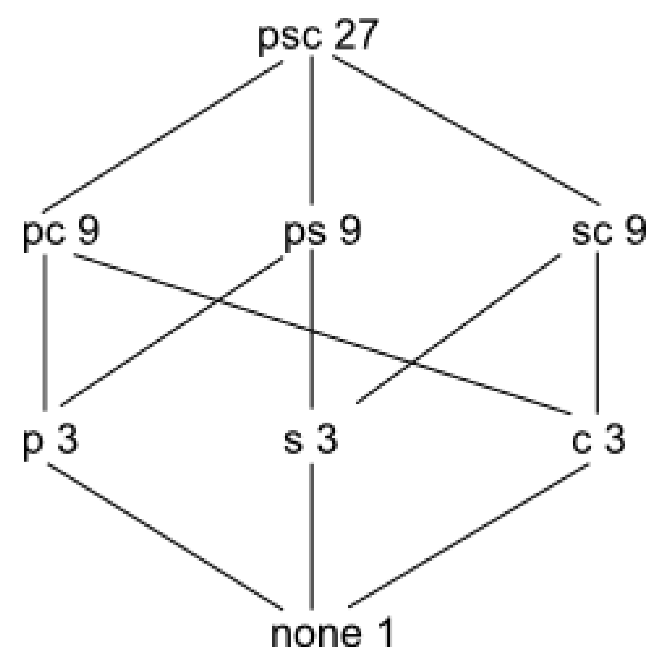
Examples:

- $(c) \not\preccurlyeq (pc)$
- $(c) \preccurlyeq (p)$

The Dependence Relation $\preccurlyeq$

$\preccurlyeq$ is a partial ordering

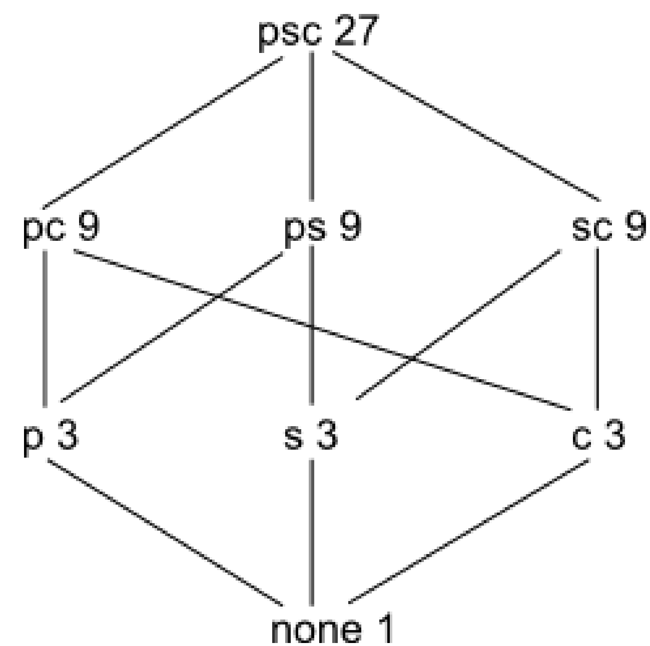
- **Reflexive:**
  $Q \preccurlyeq Q$
- **Antisymmetric:**
  $Q_1 \preccurlyeq Q_2 \wedge Q_2 \preccurlyeq Q_1 \Rightarrow Q_1 = Q_2$
- **Transitive:**
  $Q_1 \preccurlyeq Q_2 \wedge Q_2 \preccurlyeq Q_3 \Rightarrow Q_1 \preccurlyeq Q_3$



Let L be a set of views
(L, $\preccurlyeq$) is a partially ordered set

The Dependence Relation $\preccurlyeq$

$(L, \preccurlyeq)$ is a lattice because every pair of views has a least upper bound and greatest lower bound



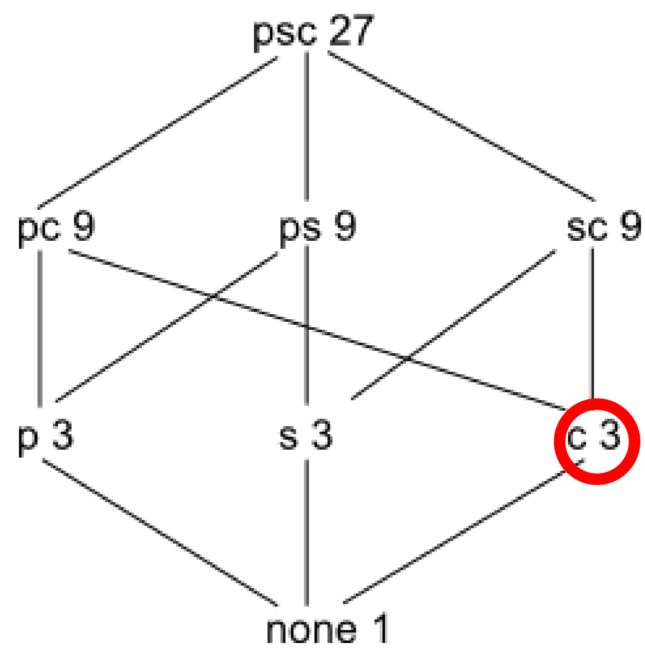
We only need these assumptions:

- $\preccurlyeq$ is a partial ordering
- There is a top element upon which every view is dependent

Answering a Query using Another View

```
SELECT Customer, SUM(Sales) AS Sales
FROM Part_Customer
GROUP BY Customer
```

	Customer	Sales
1	James	32
2	Joe	31
3	Linda	49

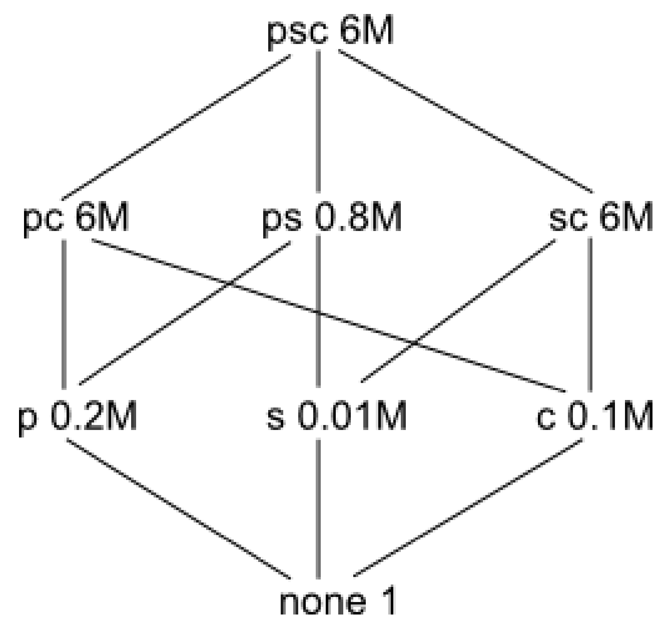


c can be answered using *pc* (or *sc*)

A More Realistic Example

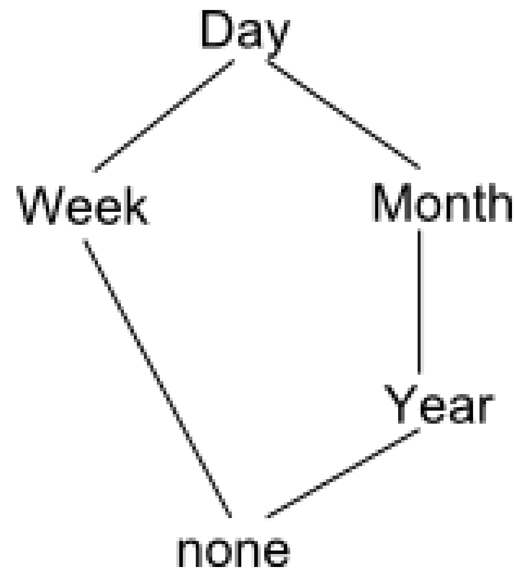
Which views to materialize?

- *psc* is obligatory



Hierarchies

- Dimensions may have hierarchies of attributes



Drill-down (more detail):

- Sales per year → sales per month → sales on a given day

Roll-up (less detail):

- Sales on a given day → sales in that month → sales in that year

Composite Lattices

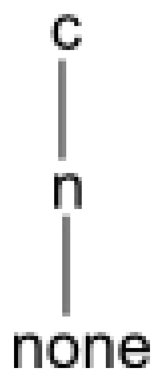
Two types of query dependencies:

- Dependencies caused by interaction of dimensions
- Dependencies within a dimension caused by attribute hierarchies

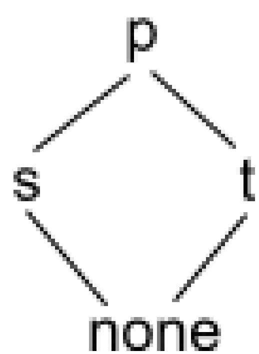
- A view is represented by an *n*–tuple $(a_1, a_2, \ldots, a_n)$, where each a_i is a point in the hierarchy for the *i*th dimension
- $(a_1, a_2, \ldots, a_n) \preccurlyeq (b_1, b_2, \ldots, b_n)$ iff $a_i \preccurlyeq b_i$ for all i

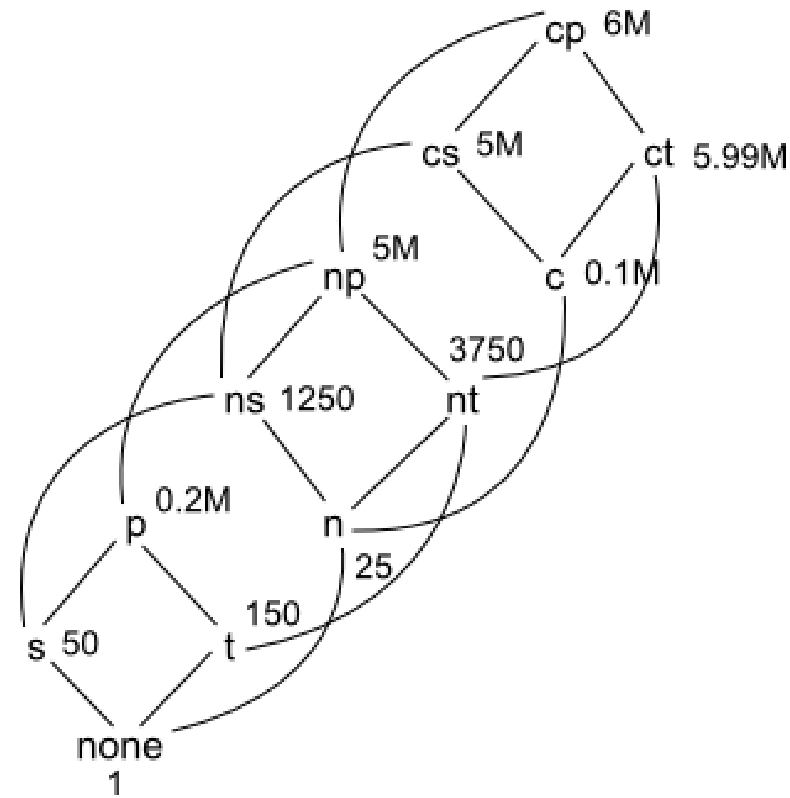
Composite Lattice Example

Customer dimension
c = customer
n = nation



Part dimension
p = part
s = size
t = type





Overview of the Presentation

- Introduction and Motivating Example
- The Lattice Framework
- Query-Cost Model
- The Greedy Algorithm
- Performance Guarantee
- Conclusion
- Paper Evaluation

Linear Cost Model

To answer query Q:

- Choose an ancestor Q_A that has been materialized
- Process the table corresponding to Q_A
- Cost of answering Q is the number of rows in the table for query Q_A.

Simple, but realistic, cost model

Overview of the Presentation

- Introduction and Motivating Example
- The Lattice Framework
- Query-Cost Model
- The Greedy Algorithm
- Performance Guarantee
- Conclusion
- Paper Evaluation

Optimizing Data-Cube Lattices

Which views to materialize?

- Minimize time taken to evaluate the set of queries identical to the views
- Constrained to materialize a fixed number of views (regardless of space)

- Optimization problem is NP-complete.

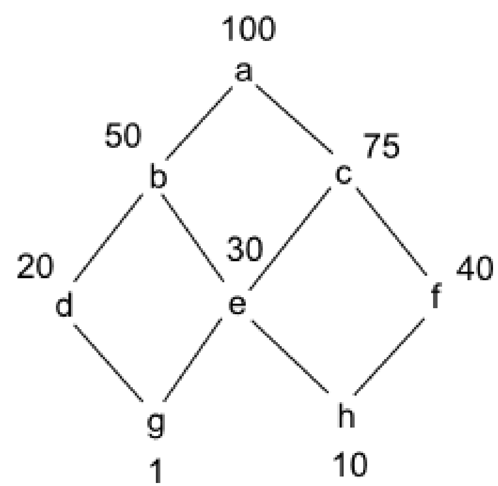
The Benefit of a View

- $C(v)$ = cost of view v
- S = set of selected views
- $B(v,S)$ = benefit of view v relative to S, as follows:

1. For each $w \preccurlyeq v$, define quantity B_w by:

   (a) Let u be the view of least cost in S such that $w \preccurlyeq u$.
   (b) If $C(v) < C(u)$, then $B_w = C(u) - C(v)$. Otherwise, $B_w = 0$.

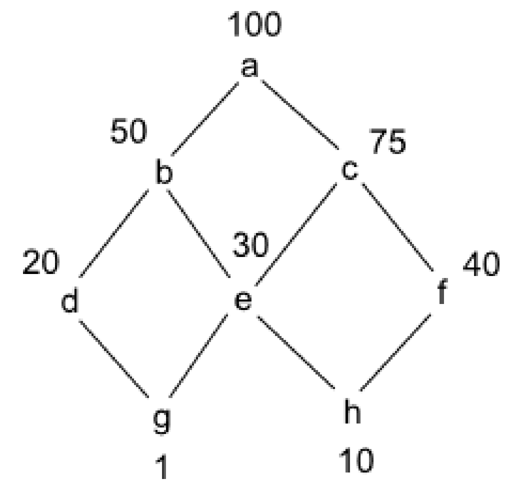
- Define $B(v,s) = \sum_{w \preccurlyeq b} B_w$.



The Benefit of a View (Example)

- Compute $B(v, S)$ where $v = b$ and $S = \{a\}$
- First compute B_w where $w = b$
- $u = a$
- Is $C(v) < C(u) \Leftrightarrow 50 < 100$?
- Yes, so

  $B_w = C(u) - C(v) = 100 - 50 = 50$
- Repeat for views d, e, g, and h
- $B(v,S) = 50 \times 5 = 250$



1. For each $w \preccurlyeq v$, define quantity B_w by:

   (a) Let u be the view of least cost in S such that $w \preccurlyeq u$.
   (b) If $C(v) < C(u)$, then $B_w = C(u) - C(v)$. Otherwise, $B_w = 0$.

- Define $B(v,s) = \Sigma_{w \preccurlyeq b} B_w$.

The Greedy Algorithm

- **Purpose:** Select a set of *k* views to materialize in addition to the top view

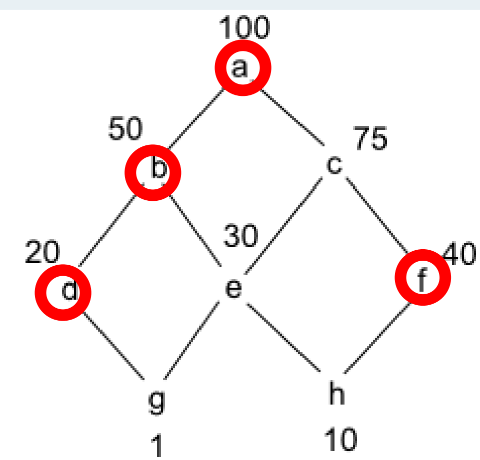
```
S = {top view};
for i=1 to k do begin
       select view v ∉ S such that B(v,S) is maximized;
       S = S ∪ {v};
end;
resulting S is the greedy selection;
```

The Greedy Algorithm (Example)

k = 3

	Choice 1 (b)	Choice 2 (f)	Choice 3 (d)
a			
b	50 x 5 = 250		
c	25 x 5 = 125	25 x 2 = 50	25 x 1 = 25
d	80 x 2 = 160	30 x 2 = 60	30 x 2 = 60
e	70 x 3 = 210	20 x 3 = 60	2 x 20 + 10 = 50
f	60 x 2 = 120	60 + 10 = 70	
g	99 x 1 = 99	49 x 1 = 49	49 x 1 = 49
h	90 x 1 = 90	40 x 1 = 40	30 x 1 = 30

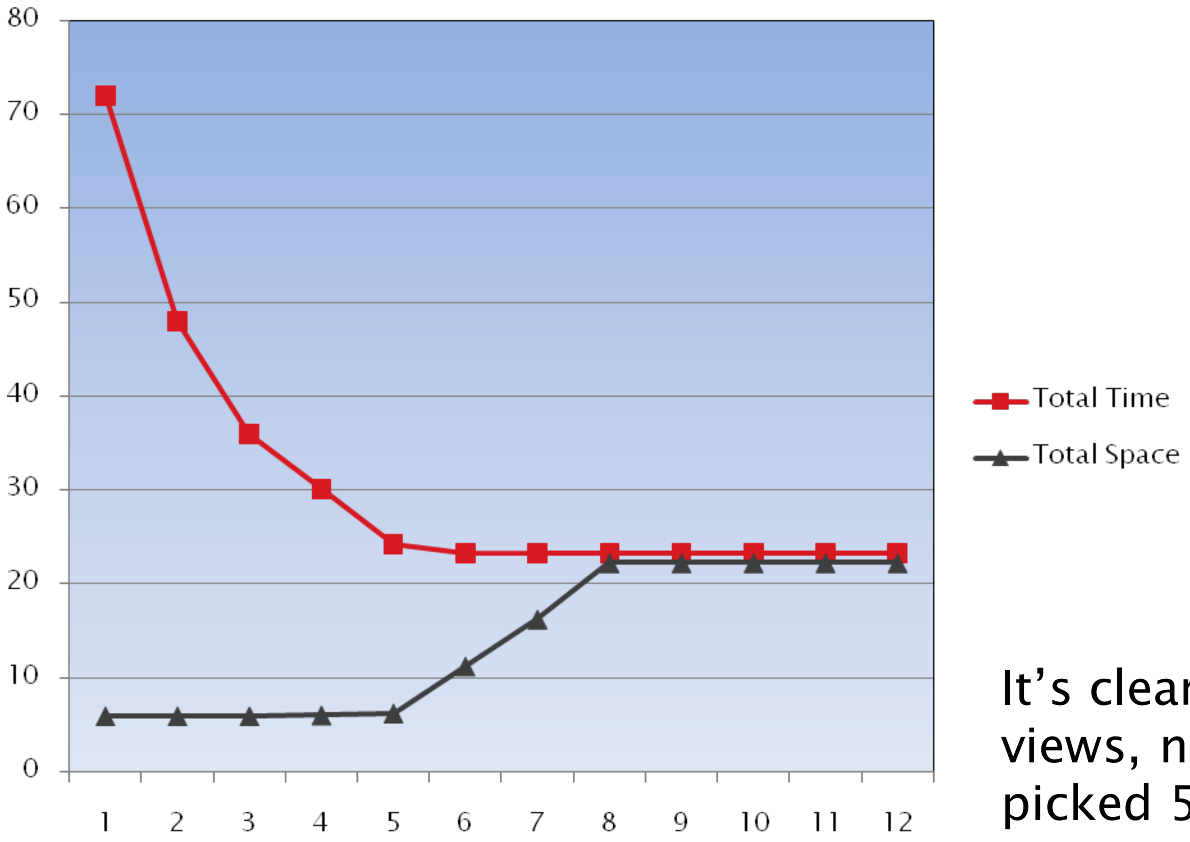
Result: $S = \{ a, b, d, f \}$



Greedy Algorithm Experiment

# Views	Selection	Benefit (million rows)	Total time (million	Total space (million rows)
1	cp	infinite	72	6
2	ns	24	48	6
3	nt	12	36	6
4	c	5.9	30.1	6.1
5	p	5.8	24.3	6.3
6	cs	1	23.3	11.3
7	np	1	22.3	16.3
8	ct	0.01	22.3	22.3
9	t	small	22.3	22.3
10	n	small	22.3	22.3
11	s	small	22.3	22.3
12	none	small	22.3	22.3

Experiment Results in Graphics



It's clear when to stop picking views, namely when we have picked 5 views including the top view, i.e., when $k = 4$

Overview of the Presentation

- Introduction and Motivating Example
- The Lattice Framework
- Query-Cost Model
- The Greedy Algorithm
- Performance Guarantee
- Conclusion
- Paper Evaluation

Performance Guarantee

For *no* lattice does the greedy algorithm give a benefit less than 63% of the optimal benefit.

It can be shown that: $$B_{greedy} / B_{opt} \geq 1 - \left(\frac{k-1}{k} \right)^k$$

where B_{greedy} is the benefit of k views chosen by the greedy algorithm, and B_{opt} is the benefit of an optimal set of k views.

As $k \to \infty$, $\left(\frac{k-1}{k} \right)^k$ approaches $1/e$,
so $B_{greedy}/B_{opt} \geq 1 - 1/e \cong 0.63$

Performance Guarantee

- Chekuri has shown using a result of Feige that unless $P = NP$ there is no polynomial-time algorithm that can guarantee a better bound than the greedy

Overview of the Presentation

- Introduction and Motivating Example
- The Lattice Framework
- Query-Cost Model
- The Greedy Algorithm
- Performance Guarantee
- Conclusion
- Paper Evaluation

Conclusion

- Materialization of views is an essential query optimization strategy
- The right selection of views to materialize is critical
- It is important to materialize some but not all views
- The greedy algorithm performs this selection
- No polynomial-time algorithm can perform better than the greedy.

Overview of the Presentation

- Introduction and Motivating Example
- The Lattice Framework
- Query-Cost Model
- The Greedy Algorithm
- Performance Guarantee
- Conclusion
- Paper Evaluation

Paper Evaluation

Good things

- Well written
- Well structured
- Refers to a more detailed version of the paper

Things that could be better:

- A figure of an actual cube would have been nice
- There were some mistakes, including a quite critical one on page 212

Paper Evaluation

1. For each $w \preceq v$, define the quantity B_w by:

   (a) Let u be the view of least cost in S such that $w \preceq u$. Note that since the top view is in S, there must be at least one such view in S.

   (b) If $C(v) < C(u)$, then $B_w = C(v) - C(u)$. Otherwise, $B_w = 0$.

2. Define $B(v, S) = \sum_{w \preceq v} B_w$.

C(*v*) – *C*(*u*) should be *C*(*u*) – *C*(*v*)

Thank you for your attention

Any questions?