CompSci 590.6

Understanding Data:
Theory and Applications

Lecture 3

Data Cube

Implementation and Selective Materialization

Instructor: Sudeepa Roy

Email: *sudeepa@cs.duke.edu*

Project Update

- Instructions in a day or two
- SQL Server is being set up on a new machine
  - needed for Data Cube
- Discussion on project ideas at the end of the class

Demo of Data Cube (SQL Server)

- In class
- With Natality Data
- Remove “order by”
- Remove “with cube”
  - Comparable time
- Marital Status, Smoking
  - 1: Yes 2: No
- Education
  - Integer value: #years
  - 1:0-8, 2: 9-11, 3: 12, 4:13-15, 5: >= 16
- Mother’s Age
  - Integer value: age
  - 1: < 15, 2: 15-19, 3: 20-24, 4: 25-29, 5: 30-34, 6: 35-39, 7: 40-44, 8: 45-49, 9: 50-54

Today's Paper

Harinarayan-Rajaraman-Ullman

Implementing data cubes efficiently

SIGMOD 1996

(1730 citation on Google Scholar)

Who are the authors?

Concepts

- Decision Support Systems (DSS)
  - used in businesses
  - get data from standard (operational) databases
  - compute aggregates to identify trends
- Data Warehouses
  - stores such historical information
  - large and grow over time
  - can slow down DSS – limits productivity
  - More about data warehouses in Lecture 4

This paper: Materialize Selected Views

- "Materialize" Views
  - pre-compute and store query answers
  - query answer can be obtained quickly
    - no computation needed at runtime
  - either, frequently asked queries
  - or, infrequently asked queries if that help to answer a number of other queries

- Challenge: how to select a good set of queries to materialize

Example: TPC-D Benchmark

- Models a business warehouse
- Three dimensions:
  - part (p), supplier (s), customer (c)
- A cell (p, s, c) represents the sale price (SP)
  - of part p that was bought from supplier s and sold to customer c
- Add "ALL" to denote consolidated sales
  - e.g. (p, ALL, c)
  - total sales of a given part p to a given customer c (across all suppliers)
  - similarly, (p, ALL, ALL): total sales of a given part p

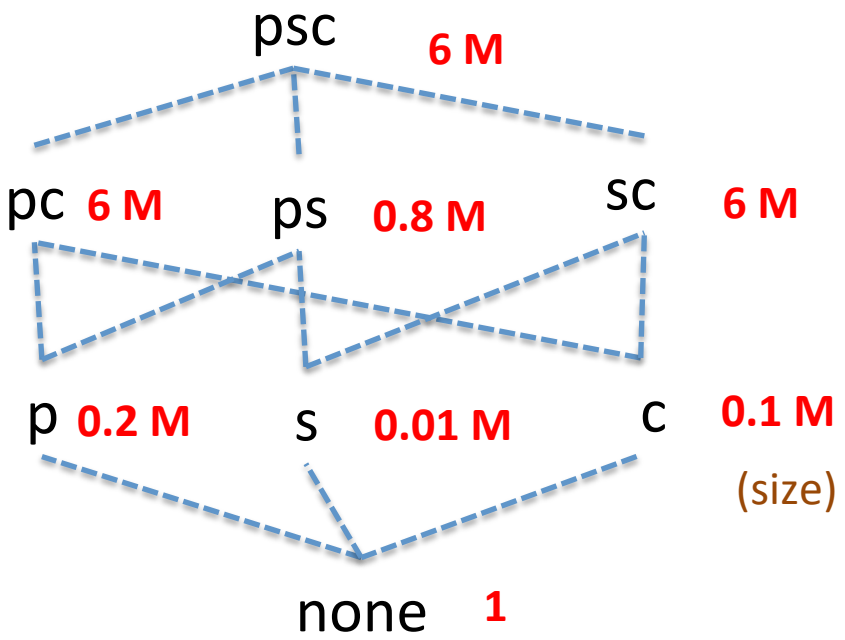
Lattice Structure of Data Cube on p, s, c

- Q1 ≤ Q2 if and only if Q1 can be answered using only the results of Q2

- (p) ≤ (c)? - No
- (c) ≤ (p)? - No
- (p) ≤ (pc)? – Yes

- ≤ Partial order
  - ancestor
  - descendent
  - next

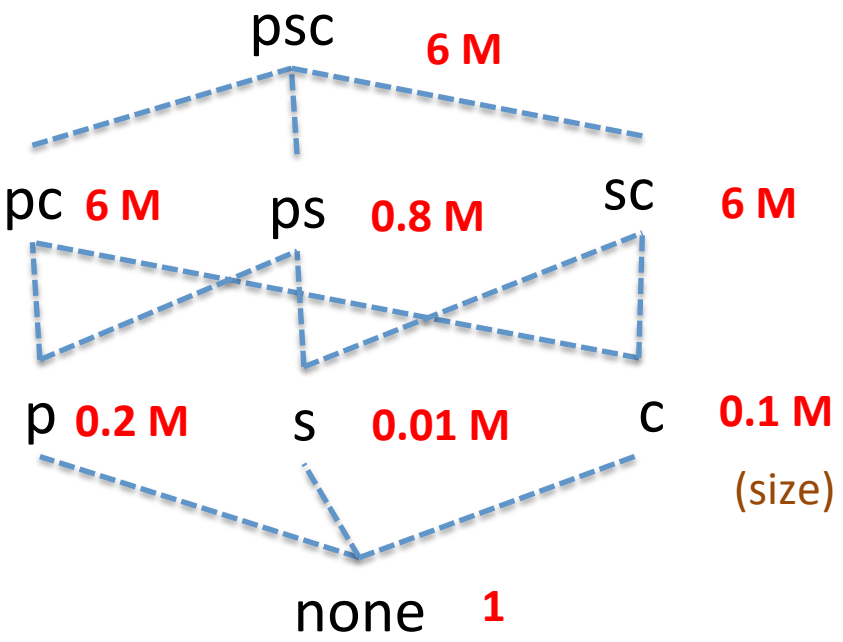
- We need a top-view, on which every view is dependent



Note:
inverted lattice structure compared to Lecture 2

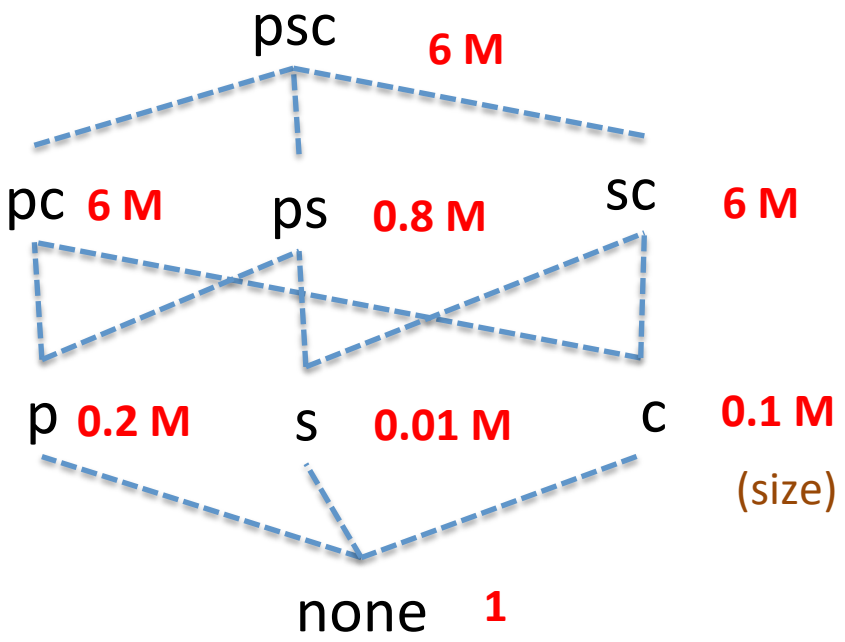
Option 1: Materialize everything?

- All views of the cube
- Here: about 19 M rows
- (+) Best query response time
  - no computation at runtime
- (-) Need to store every single cell of cube
  - too much space
  - too much time to pre-compute
  - impacts indexing



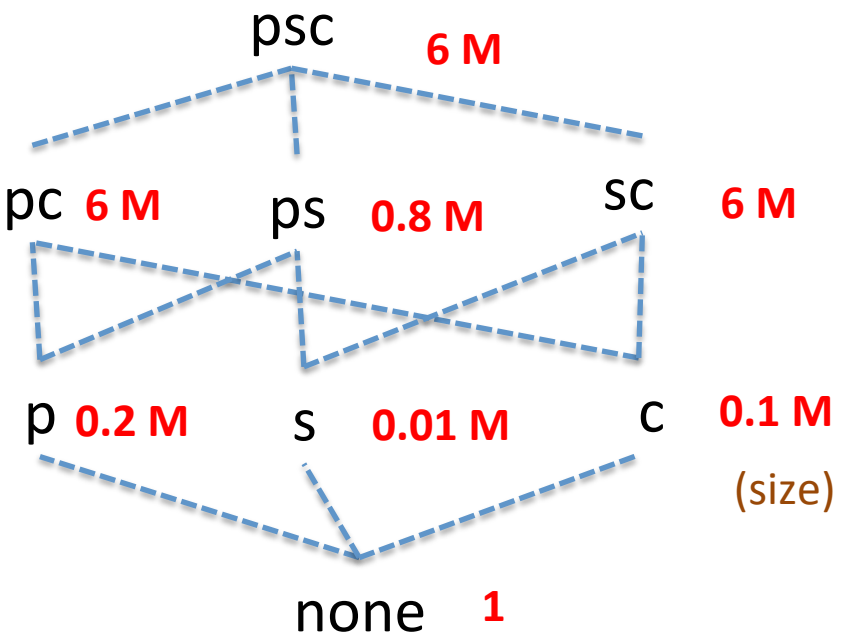
Option 2: Materialize nothing?

- Go to raw data and compute query on request
- (-) Bad query response time
- (+) No extra space
- (+) Scalable for large data



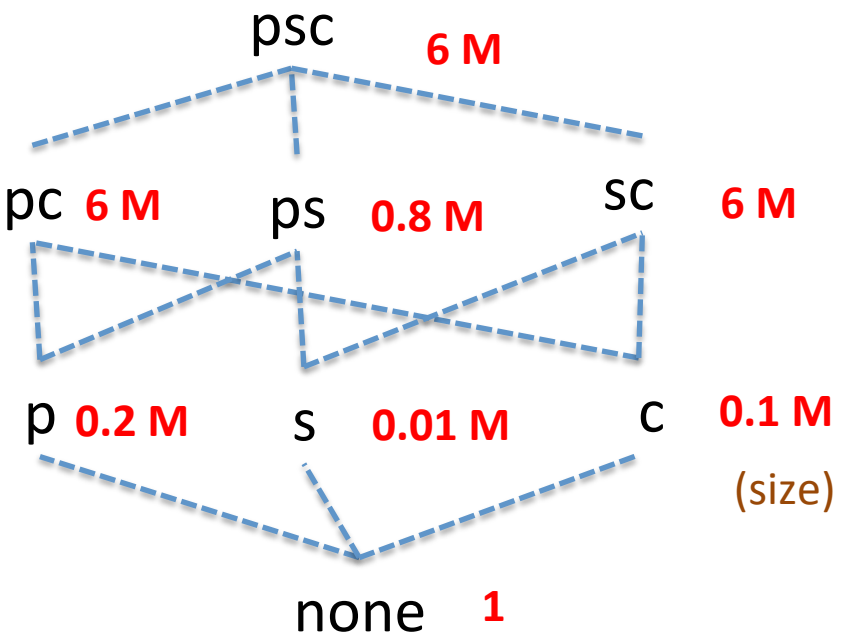
Option 3: Materialize something?

- Approach in this paper
- Dependent cells
  - can be computed from other cells
  - e.g. (p, ALL, c) = Σ_s (p, s, c)
  - 70% cells in the adjacent cube are dependent
  - no ALL – not dependent (psc)
- Space limitations
- Carefully pick the right cells to materialize



Assumptions on cost

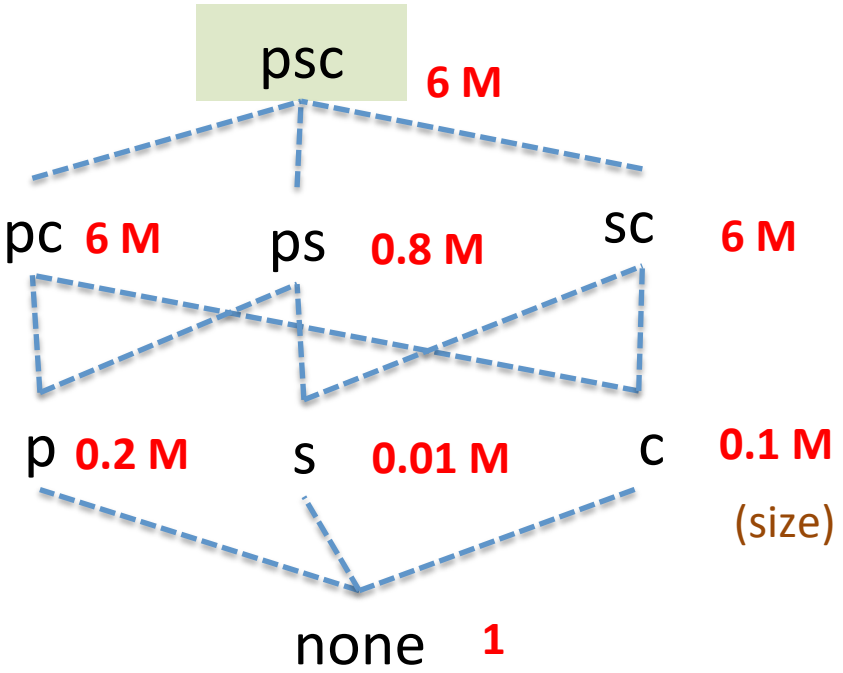
- The cost of answering a query is proportional to the number of row examined
- e.g. : Group by parts (p)
  - From materialized (p): 0.2 M
  - From materialized (pc): 6 M
- Assumes no index on views
  - Even with selection condn, say (p='widget'), same cost
  - or half cost on average



Example: Comparing choices

(psc) has to be materialized

to avoid visiting raw data

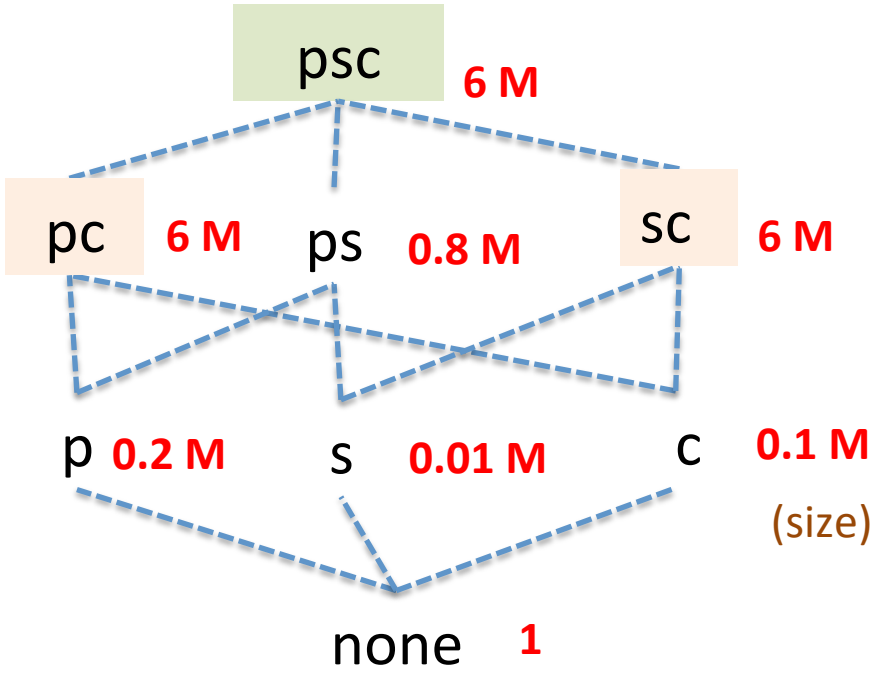


Example: Comparing choices

Do not materialize (pc) or (sc)

about same cost from (psc)

in general – compute from least-cost ancestor



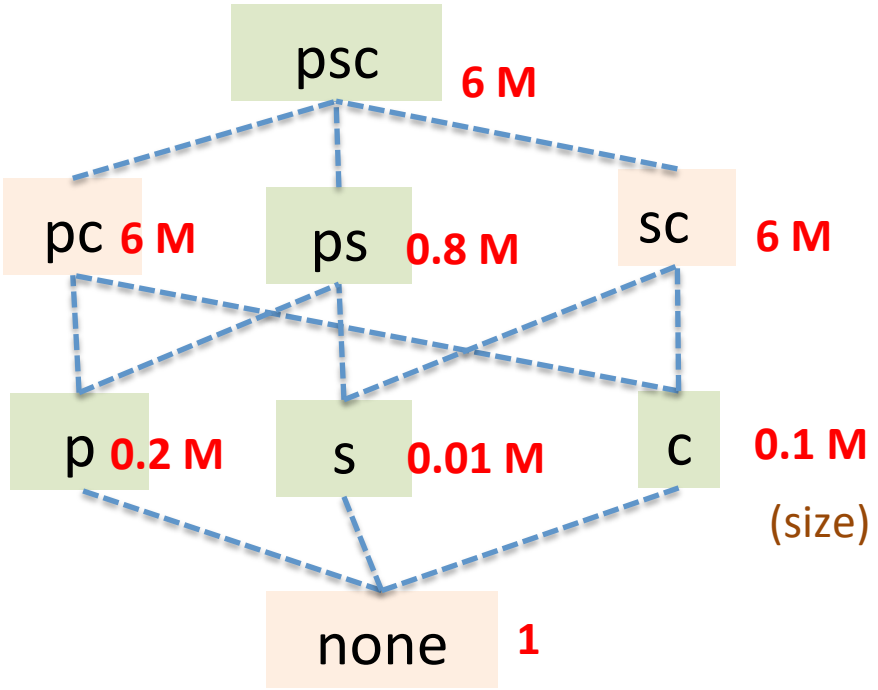
Example: Comparing choices

Materialize everything

- **space = 19.11 M**
- time ≈ 19.11 M

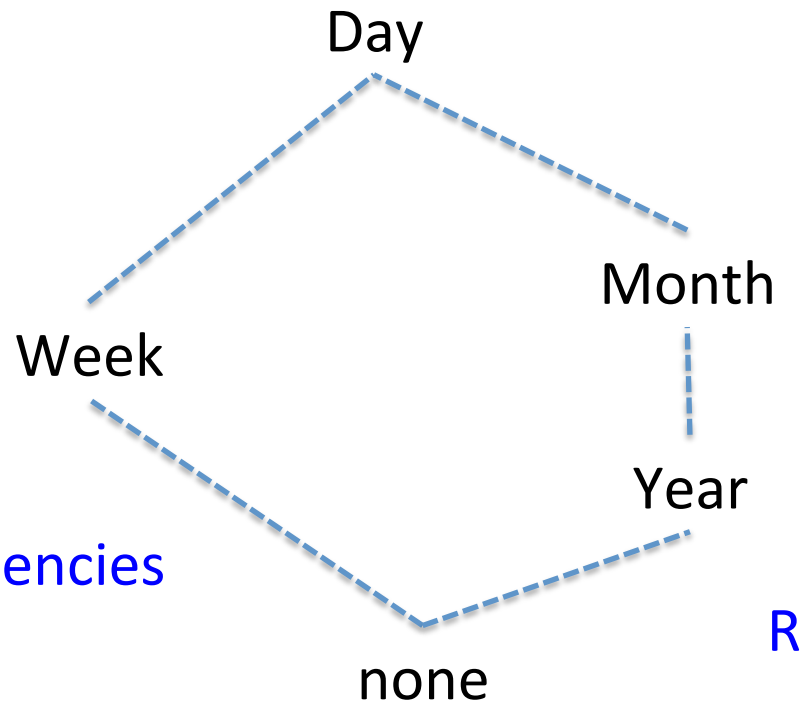
Materialize selectively

- **space = 7.11 M**
- time ≈ 19.11 M
- > 60% savings in space



Hierarchies

- Some dimensions (attributes) are organized in hierarchies
- Should be considered while deciding materialization of views



Functional dependencies
Month -> Year
(Jan'15) -> 2015

Roll-up

Drill-down

Hierarchy of time attributes

Composite Lattices for Multiple, Hierarchical Dimensions

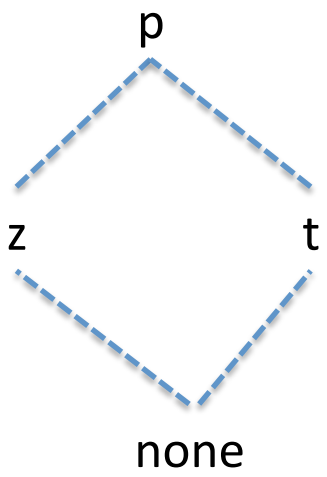
- Two types of query dependencies

1. Caused by the interaction of different dimensions
   - e.g, p, s, c
2. Caused by attribute hierarchies within a dimension

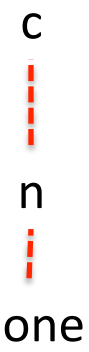
- n dimensions
- suppose arbitrary group by allowed for any/no member of the hierarchy of each dimension
- in each (a1, …, an) in the view, each ai is a point in the hierarchy

Combining Two Hierarchical Dimensions

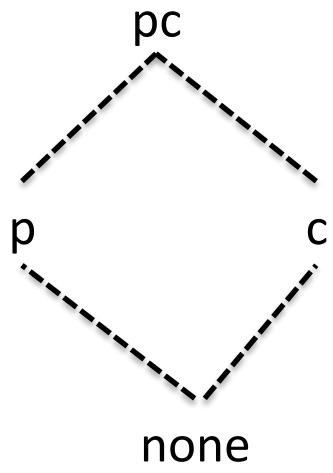
part:
size(z), type(t)



customer:
nation(n)

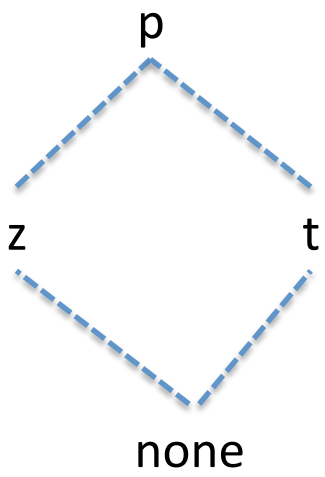


- lattice structure without hierarchy

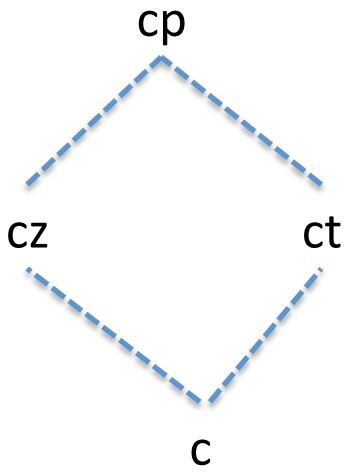


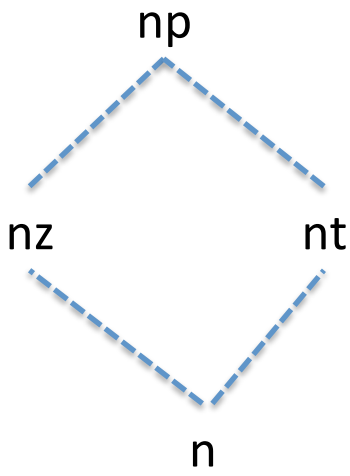
Combining Two Hierarchical Dimensions

part:
size(z), type(t)

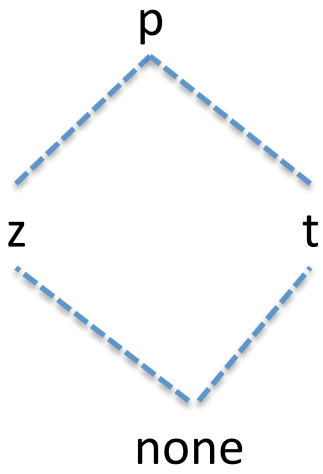


Direct Product Lattice



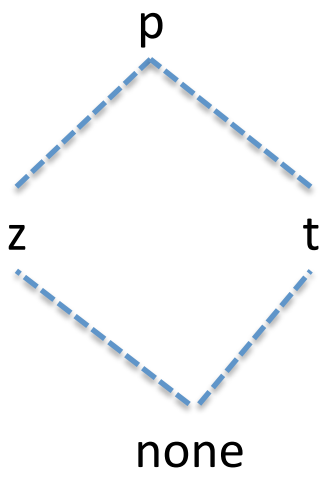


customer:
nation(n)



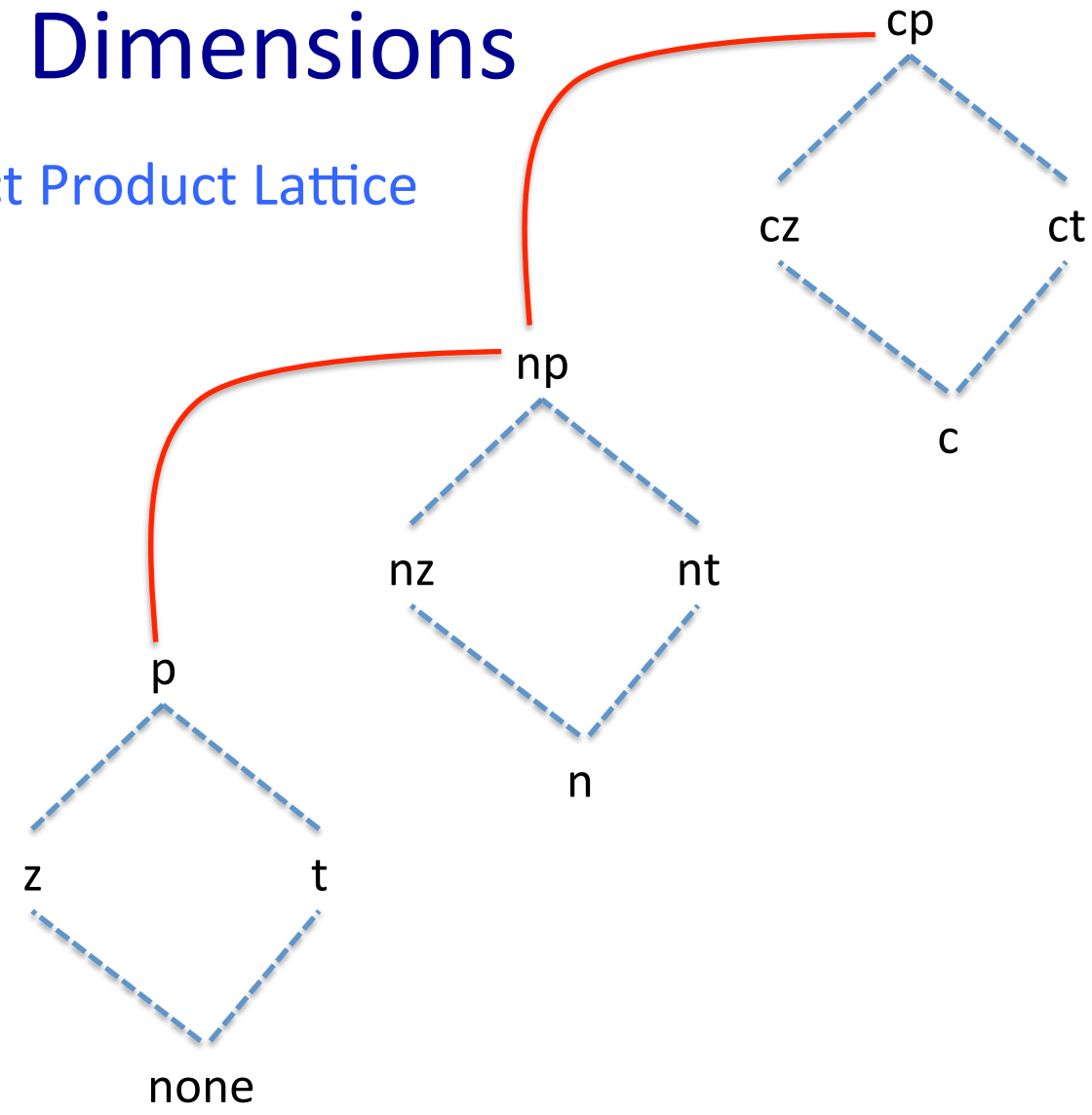
Combining Two Hierarchical Dimensions

part:
size(z), type(t)



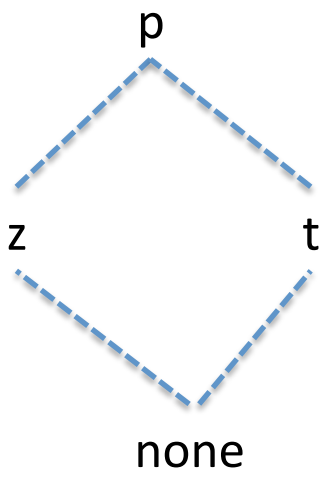
customer:
nation(n)

Direct Product Lattice

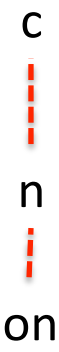


Combining Two Hierarchical Dimensions

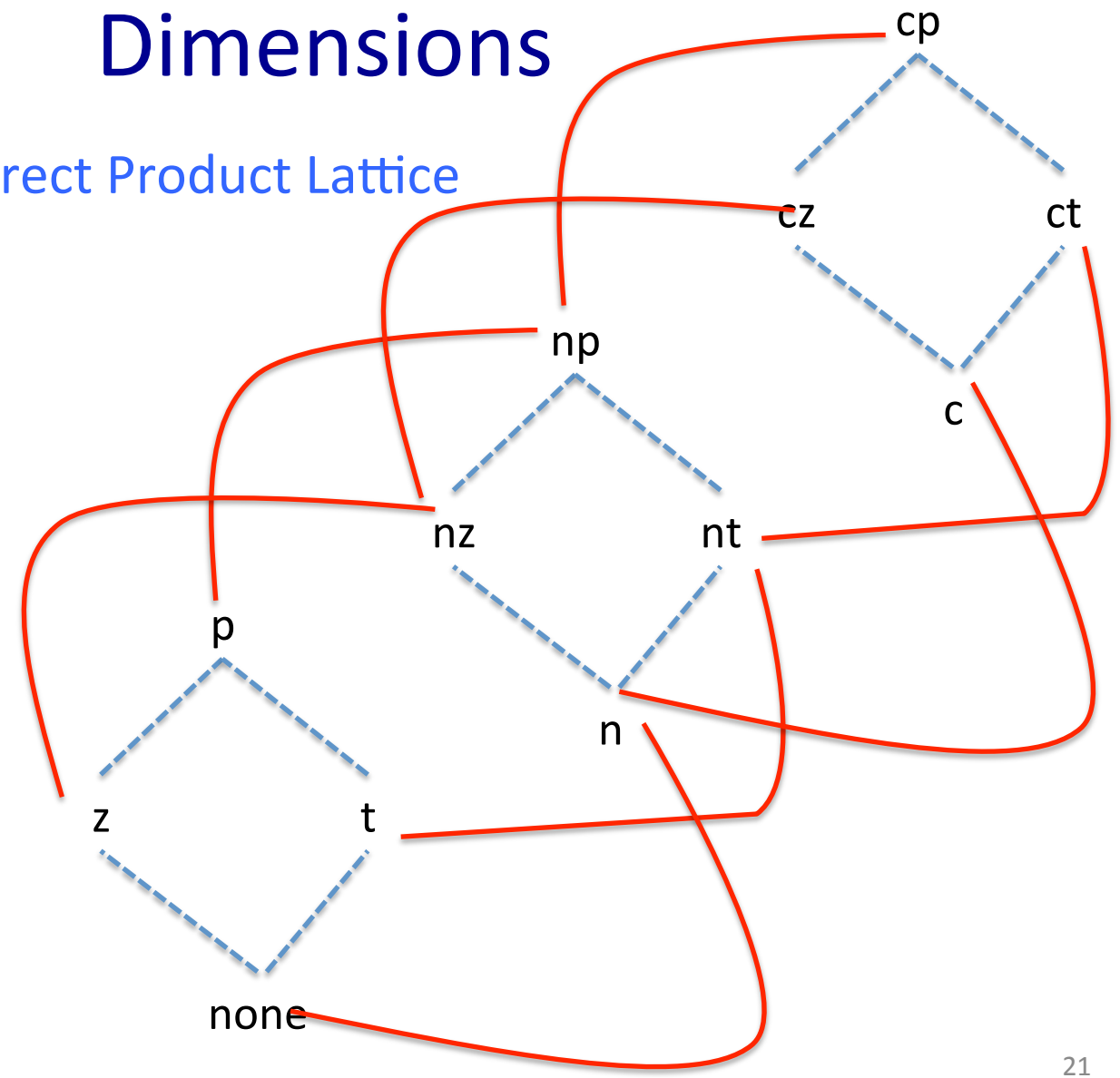
part:
size(z), type(t)



customer:
nation(n)

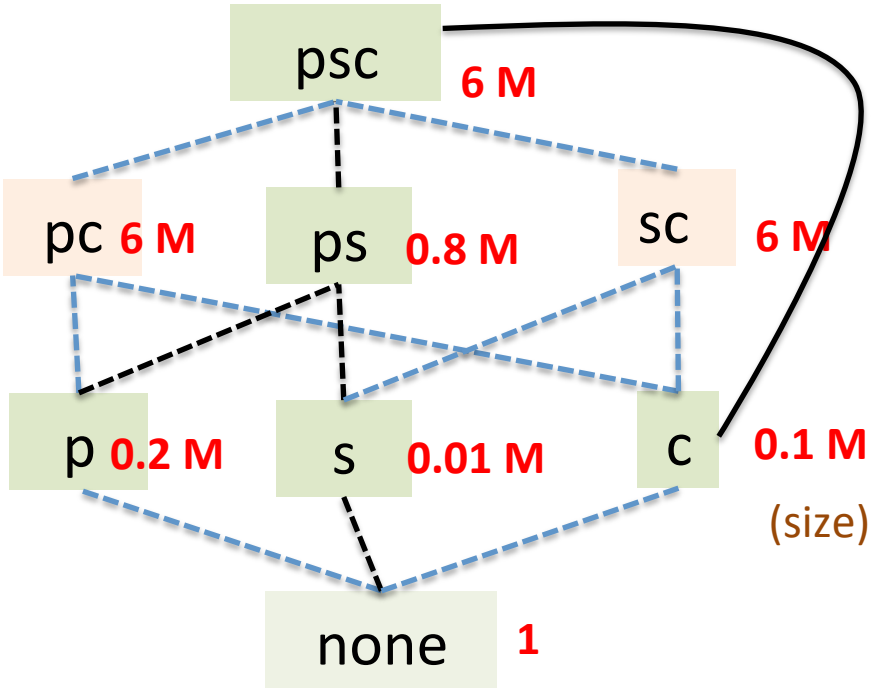


Direct Product Lattice



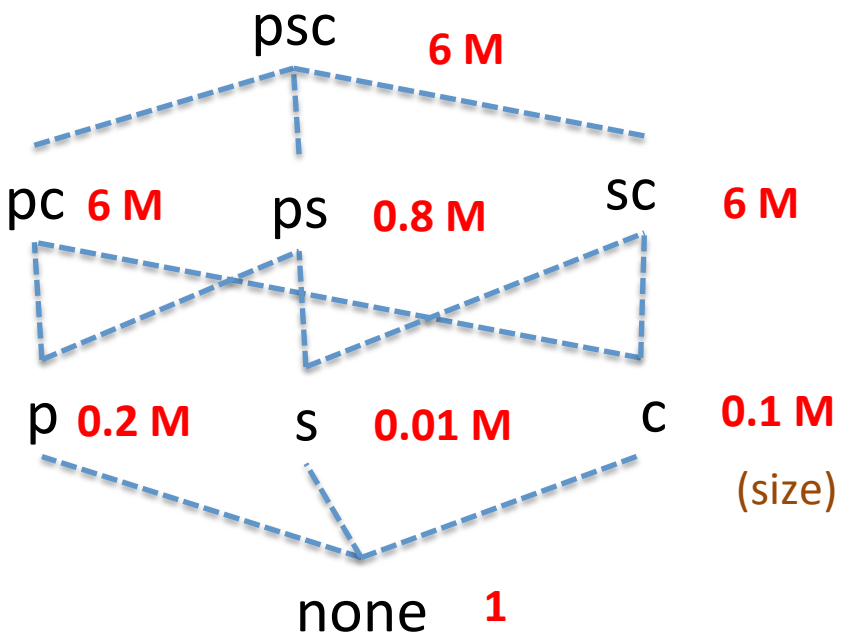
Advantages of Lattice Framework

- To reason with dimension hierarchy
  - Not always a "hypercube"
- Can model the common queries better
  - Users typically go along the edges
  - Drill-down (going up) and Roll-up (going down) along a path
- The order of materializing views
  - Suppose a set S of views has to be materialized
  - We do not need to go to raw data to materialize every view
  - Topological order sort in S
  - Materialize from the smallest ancestor



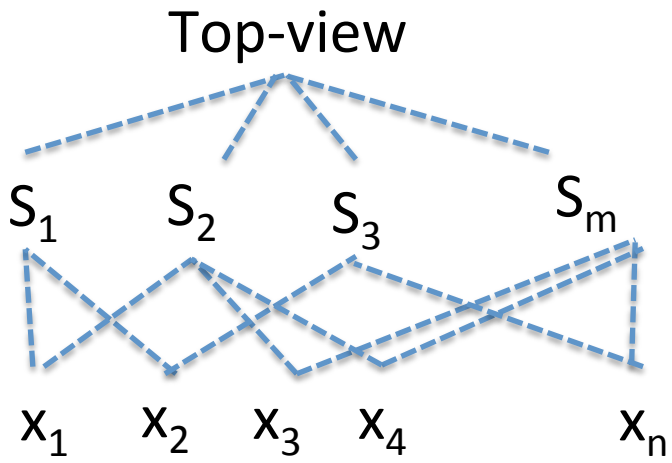
Optimization Problem

1. Minimize the time taken to evaluate views in an arbitrary lattice
   - not necessarily full hypercube lattice
2. Constrained to materialize a fixed number of views
   - regardless of the space they use



NP-Hardness

- Reduction from k-cover
  - Sets: {S_1, …, S_m} on n elements {x_1, …, x_n}
  - Include at most k sets to cover as many elements as possible

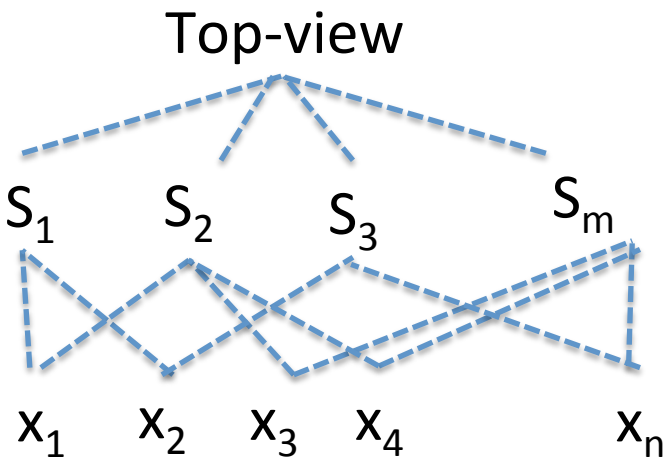


- The lattice structure for the optimization problem is shown in the figure
  - Bound on #views = k (apart from top-view)
  - Top-view costs = M > 1 (covers all views)
  - Each set S_i costs= 1 (covers itself and its elements)
  - Each element x_i costs = 1 (covers itself)

none

NP-Hardness

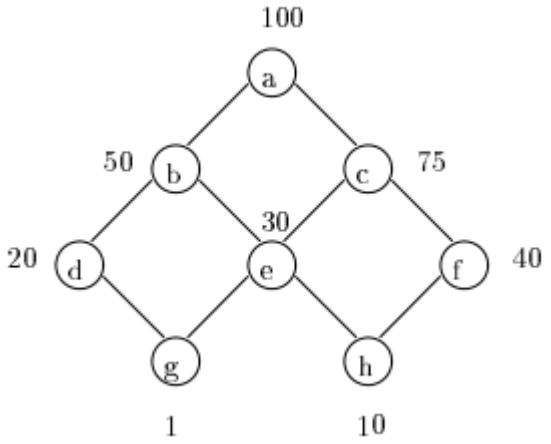
- Reduction from k-cover (contd.)
  - WLOG always set-views S_i are chosen. Why?
  - WLOG exactly k set-views are chosen. Why?
  - Suppose a set of sets O, |O| = k, covers N elements
  - Gives a solution for views with total cost =
    - M (for top-view)
    - + k + (m-k)M (for sets views S_i)
    - + N + (n-N)M (for element views x_i)
  - = k + mM + nM – (M-1)N
  - Also vice versa
  - i.e. N is maximized if the cost is minimized



none

Greedy Algorithm

- Input: Data cube lattice
  - space cost C(v) for view v
  - limit k on #views (in addition to the top view)
- Suppose we have selected a set S, |S| < k+1
  - top view is in S
- The benefit of v w.r.t. S, B(v, S) is computed as follows
  - for all w ≤ v,
  - Let u be the view with least cost in S such that w ≤ u (at least top-view)
  - If C(v) < C(u), B_w = C(v) – C(u), else 0
- B(v, S) = $\Sigma_{w \leq v} B_w$
  - Includes itself
- Choose v not in S yet with max B(v, S) k times
  - Here k = 2 (apart from top-view)
  - S = {a} initially
- Here optimal, but not always



	First Choice	Second Choice	Third Choice
b	$50 \times 5 = 250$		
c	$25 \times 5 = 125$	$25 \times 2 = 50$	$25 \times 1 = 25$
d	$80 \times 2 = 160$	$30 \times 2 = 60$	$30 \times 2 = 60$
e	$70 \times 3 = 210$	$20 \times 3 = 60$	$20 + 20 + 10 = 50$
f	$60 \times 2 = 120$	$60 + 10 = 70$	
g	$99 \times 1 = 99$	$49 \times 1 = 49$	$49 \times 1 = 49$
h	$90 \times 1 = 90$	$40 \times 1 = 40$	$30 \times 1 = 30$

When Greedy is not optimal

- K = 2
- Greedy: {c, b} or {c, d}
  - C: 101 * 41 = 4141, b or d: 100 * 41 = 4100
  - b or d: 100 * 21 = 2100
  - Total benefit = 6241
- Optimal: {b, d}
  - Total benefit = 8200
- Ratio
  - 6241/8200 = about ¾
- As bad as it can get for k = 2
- Actual ratio 1 – (k-1/k)^k >= 1 – 1/e
- Extension to space limit
  - Benefit per unit space
  - Some small view can exclude large views
  - Still can get a bound

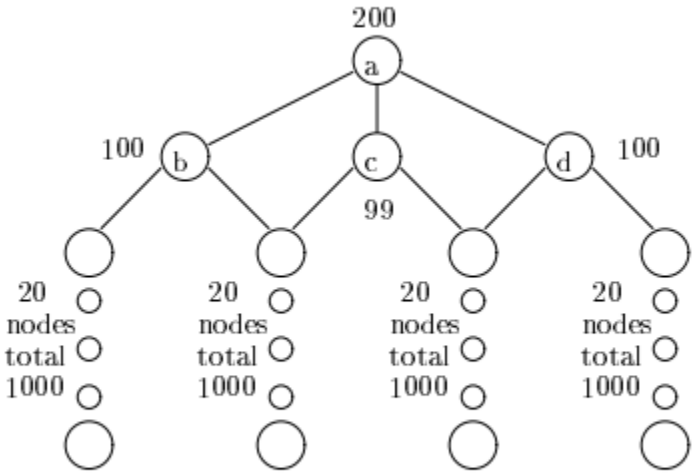


Figure 9: A lattice where the greedy algorithm does poorly

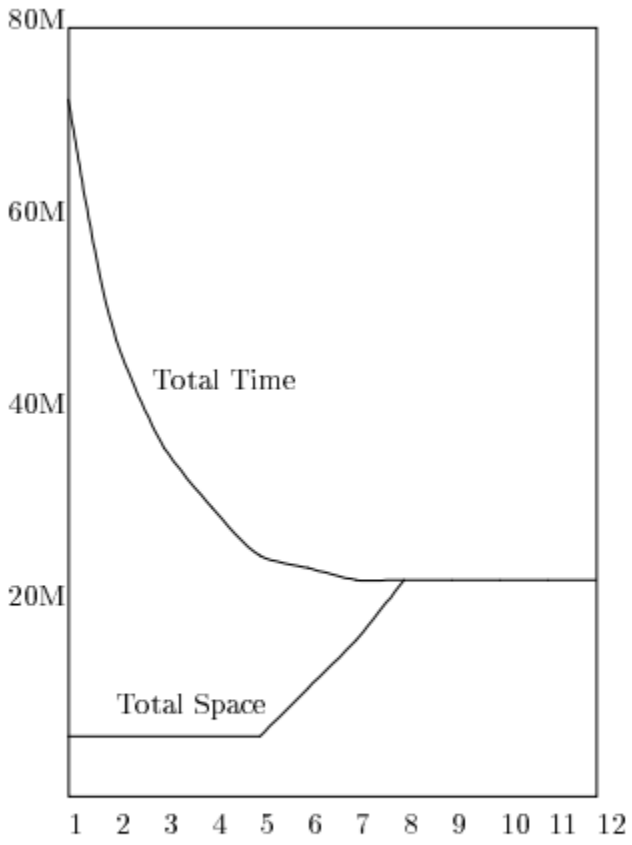


Figure 11: Time and Space for the greedy view selection for the TPC-D-based example.

Project ideas on Cube

- in class