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
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 $\leq$ Q2 if and only if Q1 can be answered using only the results of Q2
- (p) $\leq$ (c)? - No
- (c) $\leq$ (p)? - No
- (p) $\leq$ (pc)? – Yes
- $\leq$ 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. \((p, \text{ALL}, c) = \Sigma_s (p, s, c)\)
  - 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

Hierarchy of time attributes

Day

Month

Year

Week

Month -> Year
(Jan’15) -> 2015

Roll-up

Drill-down

Functional dependencies
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 \((a_1, \ldots, a_n)\) in the view, each \(a_i\) is a point in the hierarchy
Combining Two Hierarchical Dimensions

- lattice structure without hierarchy
Combining Two Hierarchical Dimensions

part:
  size(z), type(t)

None

customer:
  nation(n)

None

Direct Product Lattice
Combining Two Hierarchical Dimensions

part:
  size(z), type(t)

Direct Product Lattice

customer:
nation(n)
  c
  n
  none

none

p

z

t

none

np

n

t

nz

nt

cz

cp

c

ct

none

z

t

none
Combining Two Hierarchical Dimensions

part:
size(z), type(t)

none

Direct Product Lattice

customer:
nation(n)

c
n
none

none
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, \ldots, S_m\} on n elements \{x_1, \ldots, 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)
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
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 $w \leq v$,
  - Let $u$ be the view with least cost in $S$ such that $w \leq u$ (at least top-view)
  - If $C(v) < C(u)$, $B_w = C(v) - C(u)$, else 0

- $B(v, S) = \sum_{w \leq v} B_w$
  - 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
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**
  - \(6241/8200 = \text{about } \frac{3}{4}\)
- **As bad as it can get for } K = 2**
- **Actual ratio** \(1 - (k-1/k)^k \geq 1 - 1/e\)
- **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