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 \(c\)
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 $(p, s, \text{ALL})$
**Aggregations (Example)**

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

<table>
<thead>
<tr>
<th>Supplier</th>
<th>All</th>
<th>Laptop Mouse</th>
<th>Monitor</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dell</td>
<td>18</td>
<td>19</td>
<td>12</td>
<td>49</td>
</tr>
<tr>
<td>Linda</td>
<td>8</td>
<td>9</td>
<td>4</td>
<td>21</td>
</tr>
<tr>
<td>Joe</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>James</td>
<td>7</td>
<td>8</td>
<td>3</td>
<td>18</td>
</tr>
</tbody>
</table>

(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.
The data cube can be represented with a simple table

The **Sales** Table:

<table>
<thead>
<tr>
<th>Part</th>
<th>Supplier</th>
<th>Customer</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laptop</td>
<td>Apple</td>
<td>James</td>
<td>2</td>
</tr>
<tr>
<td>Laptop</td>
<td>Apple</td>
<td>Joe</td>
<td>6</td>
</tr>
<tr>
<td>Laptop</td>
<td>Apple</td>
<td>Linda</td>
<td>5</td>
</tr>
<tr>
<td>Laptop</td>
<td>Dell</td>
<td>James</td>
<td>7</td>
</tr>
<tr>
<td>Laptop</td>
<td>Dell</td>
<td>Joe</td>
<td>3</td>
</tr>
<tr>
<td>Laptop</td>
<td>Dell</td>
<td>Linda</td>
<td>8</td>
</tr>
<tr>
<td>Laptop</td>
<td>IBM</td>
<td>James</td>
<td>2</td>
</tr>
<tr>
<td>Laptop</td>
<td>IBM</td>
<td>Joe</td>
<td>4</td>
</tr>
<tr>
<td>Laptop</td>
<td>IBM</td>
<td>Linda</td>
<td>7</td>
</tr>
<tr>
<td>Monitor</td>
<td>Apple</td>
<td>James</td>
<td>2</td>
</tr>
<tr>
<td>Monitor</td>
<td>Apple</td>
<td>Joe</td>
<td>2</td>
</tr>
<tr>
<td>Monitor</td>
<td>Apple</td>
<td>Linda</td>
<td>3</td>
</tr>
</tbody>
</table>

Only independent cells are stored in the table

27 rows ...
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
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Supplier</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laptop</td>
<td>Dell</td>
<td>18</td>
</tr>
</tbody>
</table>
Overview of the Presentation

- Introduction and Motivating Example
- The Lattice Framework
- Query–Cost Model
- The Greedy Algorithm
- Performance Guarantee
- Conclusion
- Paper Evaluation
Cells are organized into sets based on the positions of ALL in their addresses.

For example, all cells with address \((p, s, c) = (_, \text{ALL}, _)\) are placed in the same set.

Each set corresponds to an SQL query result.

A set of cells \(\equiv\) a query result \(\equiv\) a **view**.
Cell Organization (Example)

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

**SELECT** Part, Customer, SUM(Sales) AS Sales  
**FROM** Sales  
**GROUP BY** Part, Customer
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 $\preceq$

- Consider two queries $Q_1$ and $Q_2$.
- $Q_1 \preceq Q_2$ 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 $Q_1 \preceq Q_2$

Examples:
- $(c) \preceq (pc)$
- $(c) \preceq (p)$
The Dependence Relation \( \preceq \)

\( \preceq \) is a partial ordering

- **Reflexive:**
  \[ Q \preceq Q \]

- **Antisymmetric:**
  \[ Q_1 \preceq Q_2 \land Q_2 \preceq Q_1 \Rightarrow Q_1 = Q_2 \]

- **Transitive:**
  \[ Q_1 \preceq Q_2 \land Q_2 \preceq Q_3 \Rightarrow Q_1 \preceq Q_3 \]

Let \( L \) be a set of views

\( (L, \preceq) \) is a partially ordered set
(L, ≼) is a lattice because every pair of views has a least upper bound and greatest lower bound.

We only need these assumptions:
- ≼ is a partial ordering
- There is a top element upon which every view is dependent
SELECT Customer, SUM(Sales) AS Sales
FROM Part_Customer
GROUP BY Customer

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
- $(a_1, a_2, \ldots, a_n) \preceq (b_1, b_2, \ldots, b_n)$ iff $a_i \preceq b_i$ for all $i$
Composite Lattice Example

Customer dimension
\( c = \text{customer} \)
\( n = \text{nation} \)

Part dimension
\( p = \text{part} \)
\( s = \text{size} \)
\( t = \text{type} \)
Overview of the Presentation

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

- \(C(\nu) = \text{cost of view } \nu\)
- \(S = \text{set of selected views}\)
- \(B(\nu, S) = \text{benefit of view } \nu \text{ relative to } S\), as follows:
  1. For each \(w \preceq \nu\), define quantity \(B_w\) by:

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

- Define \(B(\nu, s) = \sum_{w \preceq b} B_w\).
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 $C(v) < C(u) \iff 50 < 100$ ?
  - 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 $w \preceq v$, define quantity $B_w$ by:
   - (a) Let $u$ be the view of least cost in $S$ such that $w \preceq u$.
   - (b) If $C(v) < C(u)$, then $B_w = C(u) - C(v)$. Otherwise, $B_w = 0$.

   Define $B(v, s) = \sum_{w \preceq b} B_w$. 
Purpose: Select a set of $k$ views to materialize in addition to the top view

$S = \{\text{top view}\}$;
for $i = 1$ to $k$ do begin
    select view $v \notin S$ such that $B(v, S)$ is maximized;
    $S = S \cup \{v\}$;
end;
resulting $S$ is the greedy selection;
## The Greedy Algorithm (Example)

### k = 3

<table>
<thead>
<tr>
<th></th>
<th>Choice 1 (b)</th>
<th>Choice 2 (f)</th>
<th>Choice 3 (d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>50 x 5 = 250</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>25 x 5 = 125</td>
<td>25 x 2 = 50</td>
<td>25 x 1 = 25</td>
</tr>
<tr>
<td>d</td>
<td>80 x 2 = 160</td>
<td>30 x 2 = 60</td>
<td><strong>30 x 2 = 60</strong></td>
</tr>
<tr>
<td>e</td>
<td>70 x 3 = 210</td>
<td>20 x 3 = 60</td>
<td>2 x 20 + 10 = 50</td>
</tr>
<tr>
<td>f</td>
<td>60 x 2 = 120</td>
<td><strong>60 + 10 = 70</strong></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>99 x 1 = 99</td>
<td>49 x 1 = 49</td>
<td>49 x 1 = 49</td>
</tr>
<tr>
<td>h</td>
<td>90 x 1 = 90</td>
<td>40 x 1 = 40</td>
<td>30 x 1 = 30</td>
</tr>
</tbody>
</table>

Result: \( S = \{ a, b, d, f \} \)
# Greedy Algorithm Experiment

<table>
<thead>
<tr>
<th># Views</th>
<th>Selection</th>
<th>Benefit (million rows)</th>
<th>Total time (million)</th>
<th>Total space (million rows)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cp</td>
<td>infinite</td>
<td>72</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>ns</td>
<td>24</td>
<td>48</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>nt</td>
<td>12</td>
<td>36</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>5.9</td>
<td>30.1</td>
<td>6.1</td>
</tr>
<tr>
<td>5</td>
<td>p</td>
<td>5.8</td>
<td>24.3</td>
<td>6.3</td>
</tr>
<tr>
<td>6</td>
<td>cs</td>
<td>1</td>
<td>23.3</td>
<td>11.3</td>
</tr>
<tr>
<td>7</td>
<td>np</td>
<td>1</td>
<td>22.3</td>
<td>16.3</td>
</tr>
<tr>
<td>8</td>
<td>ct</td>
<td>0.01</td>
<td>22.3</td>
<td>22.3</td>
</tr>
<tr>
<td>9</td>
<td>t</td>
<td>small</td>
<td>22.3</td>
<td>22.3</td>
</tr>
<tr>
<td>10</td>
<td>n</td>
<td>small</td>
<td>22.3</td>
<td>22.3</td>
</tr>
<tr>
<td>11</td>
<td>s</td>
<td>small</td>
<td>22.3</td>
<td>22.3</td>
</tr>
<tr>
<td>12</td>
<td>none</td>
<td>small</td>
<td>22.3</td>
<td>22.3</td>
</tr>
</tbody>
</table>
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: \[ B_{\text{greedy}} / B_{\text{opt}} \geq 1 - \left( \frac{k - 1}{k} \right)^k \]

where \( B_{\text{greedy}} \) is the benefit of \( k \) views chosen by the greedy algorithm, and \( B_{\text{opt}} \) 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 \( B_{\text{greedy}} / B_{\text{opt}} \geq 1 - 1/e \approx 0.63 \)
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
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
1. For each \( w \leq v \), define the quantity \( B_w \) by:

   (a) Let \( u \) be the view of least cost in \( S \) such that \( w \leq u \). 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 \( B(v, S) = \sum_{w \leq v} B_w \).

\( C(v) - C(u) \) should be \( C(u) - C(v) \).
Thank you for your attention

Any questions?