Implementing Data Cubes Efficiently

Presented by:
• Kiran Ali
• Américo Pereira
Introduction

Decision Support Systems:
- Allow business to get data
- Users interested in identifying trends
- Delay unacceptable

Usually is used Data Warehouses instead of Operational Databases for DDS.
Introduction

Operational Databases:
- Maintain state information

Data Warehouses:
- Typically maintain historical information
- Very large and grow over time
- Used for identifying trends
The Data Cube

• Data are presented as multidimensional data cubes
• Users explore the cubes and discover information
• Data cubes as a conceptual model of multidimensional, aggregated data
Example 1

- Relation Sales
  - Dimensions: part, supplier, customer
  - Measure: sales
  - Aggregation: sum

- General Example
  - What is the total sales of a given part p a given customer c?
  - Add an additional value “ALL” to the domain of each dimension.
  - Look up value in cell (p,ALL,c)
Example 2

We can compute the value of cell \((p, \text{ALL}, c)\) as the sum of the values of cells of 
\((p,s_1,c),(p,s_2,c),...,(p,s_N,c)\)

\(N\) - number of the supplier
Data Warehousing

Query performance in data warehouses:
- Queries are very complex
- Make heavy use of aggregation
- Take very long to complete
- Limit Productivity

Solution:
- Materialize query results, pre-compute 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 from raw data.
Three Alternatives

• Materialize only part of the data cube
  - Trade-off between space required and query response time

- Which cells should be materialized?
- How many views must we materialize to get reasonable performance?
Lattice Framework

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 $Q_1 \preceq Q_2$
Lattice Framework

\preceq \text{ is a partial ordering}

- \preceq \text{ is Reflexive:}
  \[ Q \preceq Q \]
- \preceq \text{ is Antisymmetric:}
  \[ Q_1 \preceq Q_2 \land Q_2 \preceq Q_1 \Rightarrow Q_1 = Q_2 \]
- \preceq \text{ is Transitive:}
  \[ Q_1 \preceq Q_2 \land Q_2 \preceq Q_3 \Rightarrow Q_1 \preceq Q_3 \]
Example 3

(C) ≼ (PC)
(C) not ≼ (P)
PSC is obligatory to materialize
Hierarchies

Dimensions may have hierarchies of attributes

• **Drill-Down:**
  Sales per year => sales per month => sales on a given day

• **Roll-Up**
  Sales on a given day => sales in that month => sales in that year
Hierarchies example

(year) ≤ (month) ≤ day

(week) ≤ (day)

(month) not ≤ (week)
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
- \((a_1, a_2, \ldots, a_N) \preceq (b_1, b_2, \ldots, b_N)\) if \(a_i \preceq b_i\) for all \(i\)
Composite Lattices Example

(a) Customer

(b) Part
Linear Cost Model

To answer query Q:
- Choose an ancestor Qa that has been materialized
- Process the table corresponding to Qa
- Cost of answering Q is the number of rows in the table for query Qa
Optimizing data-cube Lattices

Which views to materialize?
- Minimize time taken to evaluate the set of queries identical the views
- Constrained to materialize a fixed number of views
- optimization problem is NP-complete
The benefit of a view

- \( C(v) = \text{cost of view } v \)
- \( S = \text{set of selected views} \)
- \( B(v,S) = \text{benefit of view } v \text{ relative to } S, \text{ as follows:} \)
  - 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 \( C(v) < C(u) \), \( B_w = C(u) - C(v) \), otherwise \( B_w = 0 \)
  - Define \( B(v,S) = \sum_{w \preceq b} B_w \).
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 $C(v) < C(u) \iff 50$ less then 100?
yes, $B_w = C(u) - C(v) = 100 - 50 = 50$
Repeat for views d, e, g and h
$B(v,S) = 50 \times 5 = 250$
The Greedy algorithm

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

\[
S = \{\text{top view}\};
\]
\[
\text{for } i=1 \text{ to } k \text{ do begin}
\]
\[
\quad \text{select that view } v \text{ not in } S \text{ such that } B(v,S) \text{ is maximized};
\]
\[
\quad S = S \text{ union } \{v\};
\]
\[
\text{end;}
\]
\[
\text{resulting } S \text{ is the greedy selection;}
\]
The Greedy algorithm Example

We always pick the maximum value.
For choice 2: if we choose f, we found that h be effectively calculated by using f instead of b
Benefit = (100 - 40) + (50 - 40) = 70
Performance Guarantee

The benefit of the Greedy algorithm is at least 63% of the benefit of the optimal algorithm.

If all the costs are equal then greedy gives optimal solution.

Chekuri has shown using a result of Feige that unless P = NP, there is no deterministic polynomial-time algorithm that guarantee a better bound than the greedy.
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
Questions?