Data Cube: View Materialization

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong
A major objective of data warehouses is to provide fast support to group-by queries. In this lecture, we will discuss a view materialization technique for achieving the purpose. Our discussion will eventually lead us to a beautiful result on greedy algorithms.
Recall:

- A fact table $T$ has $d$ dimension attributes $A_1, \ldots, A_d$, and a numeric attribute $B$.
- A group-by query is parameterized by a subset $G$ of $\{A_1, \ldots, A_d\}$. Its result, denoted as $T_G$, is referred to as a cuboid.
- The data cube of $T$ is the set of results of all the $2^d$ group-by queries.
We will use the following fact table as our running example:

<table>
<thead>
<tr>
<th>product</th>
<th>location</th>
<th>time</th>
<th>sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>tv</td>
<td>HK</td>
<td>Jan</td>
<td>5</td>
</tr>
<tr>
<td>tv</td>
<td>NY</td>
<td>Jan</td>
<td>6</td>
</tr>
<tr>
<td>tv</td>
<td>HK</td>
<td>Feb</td>
<td>4</td>
</tr>
<tr>
<td>tv</td>
<td>SH</td>
<td>Feb</td>
<td>8</td>
</tr>
<tr>
<td>tv</td>
<td>SH</td>
<td>Mar</td>
<td>2</td>
</tr>
<tr>
<td>dvd</td>
<td>NY</td>
<td>Jan</td>
<td>3</td>
</tr>
<tr>
<td>dvd</td>
<td>SH</td>
<td>Jan</td>
<td>7</td>
</tr>
<tr>
<td>dvd</td>
<td>SH</td>
<td>Feb</td>
<td>1</td>
</tr>
<tr>
<td>laptop</td>
<td>NY</td>
<td>Feb</td>
<td>4</td>
</tr>
<tr>
<td>laptop</td>
<td>SH</td>
<td>Feb</td>
<td>9</td>
</tr>
</tbody>
</table>
As a baseline approach, we can answer a group-by query with dimension set $G$ by sorting the fact table $T$ according to $G$.

- **Example**: To answer a group-by query with $G = \{\text{product, location}\}$ from the fact table $T$ in the previous slide, we can sort the tuples of $T$ first by product, and then by location (think: and then what?).

In practice, the amount of storage space we can afford is often larger than the size of $T$. This motivates us to **precompute** some information to reduce the cost of group-by queries.
Example 2.

In our running example, suppose that we have precomputed the cuboid $T_{G_1}$ of $G_1 = \{\text{product, location}\}$. We can now answer a group-by query with dimension set $G_2 = \{\text{product}\}$ directly from $T_{G_1}$, instead of from the fact table $T$.

Notice that the size of $T_{G_1}$ is at most that of $T$. And yet, $T_{G_1}$ includes all the information needed to answer the query with $G_2$.

Think: What other group-by queries can benefit from a precomputed $T_{G_1}$?
A naive solution to minimizing the cost of all group-by queries is to precompute and materialize (which is the data warehouse jargon for “storing”) all the $2^d$ cuboids.

However, $d$ in practice can be very large, e.g., at the order of 100. Consequently, materializing all the $2^d$ cuboids requires prohibitive space. This naturally leads to the following question: which cuboids should we choose to materialize?

In the rest of this lecture, we will take a principled approach to answer this question by formalizing it as an optimization problem. Before doing so, however, we will need to understand better how queries are answered from a set of cuboids.
All the subsets of \( \{A_1, ..., A_d\} \) form a lattice \( L \), where a subset \( G \) is a parent of \( G' \) if and only if \( G' \subset G \) and \( |G| = |G'| + 1 \).

**Example 3.**

The lattice for the fact table \( T \) in Slide 4.
Fact 4.

A materialized $G$-cuboid $T_G$ can be used to answer a query with group-by dimension set $Q$ only if $Q \subseteq G$, namely, $G$ is an ancestor of $Q$ in $L$.

For example, in the example of the previous slide, the group-by query of $G = \{L\}$ can be answered from $T_{\{PL\}}$, $T_{\{LT\}}$, and $T$, but not from $T_{\{PT\}}$. 
Let $\text{cost}(Q, G)$ be the cost of answering a group-by query with dimension set $Q$, from the $G$-cuboid $T_G$. From our earlier discussion, let us define this function as follows

$$
\text{cost}(Q, T_G) = \begin{cases} 
\infty & \text{if } Q \not\subseteq G \\
\text{SORT}(T_G) & \text{otherwise}
\end{cases}
$$

where $\text{SORT}(T_G)$ is the cost of sorting $T_G$. 
Now suppose that we have materialized a set $S$ of cuboids
\{ $T_{G_1}, \ldots, T_{G_k}$ \}. We will consider that $S$ always includes the fact table $T$ (why is this reasonable?).

The previous slide tells us that, to answer a group-by query with dimension set $Q$, we should use the cuboid chosen as follows: from all the $T_G \in S$ satisfying $Q \subseteq G$, choose the one with the smallest $\text{SORT}(T_G)$.

If $T_G$ is the selected cuboid, we define $\text{cost}_S(Q) = \text{SORT}(T_G)$. In other words, $\text{cost}_S(Q)$ is the lowest cost of answering the query from $S$. 
The total cost of answering all possible group-by queries from $S$ is thus:

$$allcost(S) = \sum_{\forall Q \subseteq \{A_1, \ldots, A_t\}} cost_S(Q)$$

We will use the following function to measure the quality of $S$:

$$benefit(S) = allcost(\{T\}) - allcost(S)$$

Recall that any $S$ will have to include $T$. Thus, $benefit(S) \geq 0$ (think: why?).
Example 5.

The figure on the right shows the sorting cost on each cuboid. Suppose $S = \{ T_{PLT}, T_{LT} \}$. The following table gives the cost of answering each group-by query:

<table>
<thead>
<tr>
<th>$Q$</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>${ PLT }$</td>
<td>10000</td>
</tr>
<tr>
<td>${ PL }$</td>
<td>10000</td>
</tr>
<tr>
<td>${ PT }$</td>
<td>10000</td>
</tr>
<tr>
<td>${ LT }$</td>
<td>1000</td>
</tr>
<tr>
<td>${ P }$</td>
<td>10000</td>
</tr>
<tr>
<td>${ L }$</td>
<td>1000</td>
</tr>
<tr>
<td>${ T }$</td>
<td>1000</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>1000</td>
</tr>
</tbody>
</table>

Therefore, $\text{allcost}(S) = 44000$. Since $\text{allcost}(\{ T \}) = 80000$, we know that $\text{benefit}(S) = 80000 - 44000 = 36000$. 
Now we are ready to formalize the question asked in Slide 7 into an optimization problem:

**Problem 6 (View Materialization Problem).**

Given an integer $k$, find a set $S$ of $k$ cuboids with the largest $\text{benefit}(S)$.

This problem is known to be NP-hard. We will therefore turn to good approximate solutions. Let $S^*$ be an optimal solution to the problem. Then, a set $S$ of $k$ cuboids is said to be $\rho$-approximate where

$$\rho = \frac{\text{benefit}(S)}{\text{benefit}(S^*)}$$

Next we will learn a greedy algorithm that guarantees a solution with approximation ratio at least $1 - 1/e \approx 0.632$. 

Y Tao  Data Cube: View Materialization
**algorithm** Greedy($k$)

/* return a set of $k \geq 1$ cuboids */

1. initialize a set $S$ with only one cuboid: the fact table $T$
2. while $|S| < k$
3. $T_X \leftarrow$ a cuboid $T_G$ maximizing benefit($S \cup \{T_G\}$)
   among all the cuboids $T_G \notin S$
4. add $T_X$ to $S$
5. return $S$
Example 7.

Let us run the algorithm with $k = 2$ on the lattice shown on the right. Initially, $S = \{ T_{PLT} \}$.

<table>
<thead>
<tr>
<th>$G$</th>
<th>$benefit(S \cup { T_G })$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${PL}$</td>
<td>20000</td>
</tr>
<tr>
<td>${PT}$</td>
<td>8000</td>
</tr>
<tr>
<td>${LT}$</td>
<td>36000</td>
</tr>
<tr>
<td>${P}$</td>
<td>17000</td>
</tr>
<tr>
<td>${L}$</td>
<td>19400</td>
</tr>
<tr>
<td>${T}$</td>
<td>18400</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>9999</td>
</tr>
</tbody>
</table>

Therefore, the algorithm adds $T_{LT}$ to $S$. 
Theorem 8.
The greedy algorithm returns a solution with approximation ratio at least $1 - 1/e$.

We will prove the above using a well-known result that applies in a more general scenario.
Let us consider the following problem. Let $U$ be a finite set (the items of the set are irrelevant). For any subset $S$ of $U$, function $f(S)$ returns a non-negative value as the quality of $S$.

We require that function $f$ is submodular. This means that, for any subsets $S \subset M \subset U$ and any $x \in U \setminus M$, it holds that

$$f(S \cup \{x\}) - f(S) \geq f(M \cup \{x\}) - f(M).$$

Intuitively, this means that it is always a good idea to place an item $x$ into a smaller set, if the goal is to increase the quality.

Given an integer $k$, the goal of the problem is to find a subset $S$ of size $k$ with the highest quality $f(S)$. Let us refer to the problem as the subset selection problem.
The problem is NP-hard but admits a standard greedy algorithm guaranteed to return a solution that is at least 63% as good as the optimal solution.

**algorithm** Greedy\((k)\)

1. initialize an empty set \(S = \emptyset\)
2. while \(|S| < k\)
3. \(x \leftarrow \) an element maximizing \(f(S \cup \{y\})\) among all the elements \(y \notin S\)
4. add \(x\) to \(S\)
5. return \(S\)

**Theorem 9.**

Let \(S^*\) be the optimal solution to the subset selection problem. Then, the set \(S\) returned by Greedy satisfies

\[
\frac{f(S)}{f(S^*)} \geq 1 - \frac{1}{e}.
\]
Let us now return to proving Theorem 8. It suffices to establish:

**Lemma 10.**

Function $\text{benefit}(S)$ is submodular.

**Proof**

Consider two sets of cuboids $S$ and $M$ such that $S \subseteq M$. Consider any cuboid $T_G$ not in $M$. We will prove

$$\text{benefit}(S \cup \{T_G\}) - \text{benefit}(S) \geq \text{benefit}(M \cup \{T_G\}) - \text{benefit}(M).$$

Let $\Gamma$ be the set of queries $Q$ satisfying (i) $Q \subseteq G$, and (ii) $\text{cost}_S(Q) \geq \text{SORT}(T_G)$. Namely, $\Gamma$ contains those queries that will be answered with $T_G$ after this cuboid is added to $S$. Similarly, let $\Lambda$ be the set of queries $Q$ satisfying (i) $Q \subseteq G$, and (ii) $\text{cost}_M(Q) \geq \text{SORT}(T_G)$.

From $S \subseteq M$, we know $\Lambda \subseteq \Gamma$. 
Proof (Cont.)

\[
\begin{align*}
\text{benefit}(S \cup \{T_G\}) - \text{benefit}(S) &= \sum_{Q \in \Gamma} (\text{cost}_S(Q) - \text{SORT}(T_G)) \\
&\geq \sum_{Q \in \Lambda} (\text{cost}_S(Q) - \text{SORT}(T_G)) \\
&\geq \sum_{Q \in \Lambda} (\text{cost}_M(Q) - \text{SORT}(T_G)) \\
&= \text{benefit}(M \cup \{T_G\}) - \text{benefit}(M).
\end{align*}
\]