Multidimensional Databases
Cyrus Shahabi
Computer Science Department
University of Southern California
shahabi@usc.edu

Outline
- Definitions (from A.R. 20 and 21)
- Focus Application: OLAP
- Prefix-Sum (from A.R. 16)
- Dynamic Data Cube (from A.R. 17)
- Iterative Data Cube (from A.R. 18)
- Wavelet-based approaches
  - Compact Data Cube (from A.R. 19)
  - ProPolyne (from A.R. 22 and 23)
Definitions

- During the past decade, the *multidimensional data model* emerged for use when the objective is to analyze data rather than to perform online transactions.
- In contrast to previous technologies, these databases view data as multidimensional *cubes* that are particularly well suited for data analysis.
- Multidimensional data models have three important application areas within data analysis:
  - *Data warehouses* are large repositories that integrate data from several sources in an enterprise for analysis.
  - *Online analytical processing* (OLAP) systems provide fast answers for queries that aggregate large amounts of detail data to find overall trends.
  - *Data mining* applications seek to discover knowledge by searching semi-automatically for previously unknown patterns and relationships in multidimensional databases.

Definitions...

- Multidimensional databases view data as cubes that generalize spreadsheets to any number of dimensions.
- In addition, cubes support hierarchies in dimensions and formulas without duplicating their definitions.
- A collection of related cubes comprises a multidimensional database or data warehouse.
- Dimensions are used for selecting and aggregating data at the desired level of detail.
- A dimension is organized into a containment-like hierarchy composed of numerous levels, each representing a level of detail required by the desired analyses.
- Each instance of the dimension, or dimension value, belongs to a particular level.
Definitions …

- Facts represent the subject—the interesting pattern or event in the enterprise that must be analyzed.
- In most multidimensional data models, facts are implicitly defined by their combination of dimension values; a fact exists only if there is a nonempty cell for a particular combination of values.
- A measure consists of two components:
  - a fact's numerical property, such as the sales price or profit
  - a formula, usually a simple aggregation function such as sum, that can combine several measure values into one.
- In a multidimensional database, measures generally represent the properties of the fact that the user wants to optimize. Measures then take on different values for various dimension combinations.
- The property and formula are chosen to provide a meaningful value for all combinations of aggregation levels.

Focus Application:
On-Line Analytical Processing (OLAP)

Multidimensional data sets:
- Dimension attributes (e.g., Store, Product, Data)
- Measure attributes (e.g., Sale, Price)

Range-sum queries
- Average sale of shoes in CA in 2001
- Number of jackets sold in Seattle in Sep. 2001

Tougher queries:
- Covariance of sale and price of jackets in CA in 2001 (correlation)
- Variance of price of jackets in 2001 in Seattle
Range Queries in OLAP Data Cubes

Ching-Tien Ho  Rakesh Agrawal
Nimord Megiddo Rammakrishnan Srikant
IBM Almaden Research Center

Outline

- Introduction
- The Basic Range-Sum Algorithm
- The Blocked Range-Sum Algorithm
- The Batch-Update Algorithm for Range-Sum Queries
- Choosing Dimension, Cuboids and Block Sizes
- Conclusion
Introduction

Example for MDDB

- Assume the data cube has four functional attributes (dimensions): age, year, state and insurance type.
- The data cube will have age*year*state*type cells, with each cell containing the total revenue (the measure attribute) for the corresponding combination of these four attributes.

<table>
<thead>
<tr>
<th>Index</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>10/24</td>
<td>$300</td>
<td>$400</td>
<td>$700</td>
<td>$500</td>
<td>$250</td>
<td>$600</td>
<td>$150</td>
</tr>
<tr>
<td>10/25</td>
<td>$120</td>
<td>$130</td>
<td>$100</td>
<td>$240</td>
<td>$300</td>
<td>$100</td>
<td>$300</td>
</tr>
<tr>
<td>10/26</td>
<td>$110</td>
<td>$170</td>
<td>$200</td>
<td>$600</td>
<td>$500</td>
<td>$500</td>
<td>$300</td>
</tr>
<tr>
<td>10/27</td>
<td>$800</td>
<td>$550</td>
<td>$790</td>
<td>$800</td>
<td>$470</td>
<td>$500</td>
<td>$250</td>
</tr>
<tr>
<td>10/28</td>
<td>$250</td>
<td>$230</td>
<td>$600</td>
<td>$600</td>
<td>$250</td>
<td>$230</td>
<td>$170</td>
</tr>
<tr>
<td>10/29</td>
<td>$170</td>
<td>$390</td>
<td>$440</td>
<td>$500</td>
<td>$250</td>
<td>$360</td>
<td>$190</td>
</tr>
<tr>
<td>10/30</td>
<td>$370</td>
<td>$500</td>
<td>$800</td>
<td>$950</td>
<td>$560</td>
<td>$590</td>
<td>$390</td>
</tr>
</tbody>
</table>
Introduction

- Range Queries
  - That apply a given aggregation operation over selected cells where the selection is specified as continuous ranges in the domains of some of the attributes.
  - Example: Sum, Max, Count and Average

- The Basic Range-Sum Algorithm
- The Blocked Range-Sum Algorithm
- The Batch-Update Algorithm for Range-Sum Queries
- Conclusion
The Basic Range-Sum Algorithm

- Pre-computed Prefix-Sum array $P$
  - Let $P$ be a $d$-dimensional array of size
  - $N = n_1 \times n_2 \times \ldots \times n_d$

$$P[x, y] = Sum(0 : x, 0 : y) = \sum_{i=0}^{x} \sum_{j=0}^{y} A[i, j]$$

Example

<table>
<thead>
<tr>
<th>Array $A$ (original array)</th>
<th>Array $P$ (Prefix-Sum array)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

$P(1, 1) = A(0, 0) + A(0, 1) + A(1, 0) + A(1, 1)$
The Basic Range-Sum Algorithm

- **Theorem 1:**
  Proves how any range-sum of A can be computed from up to $2^d$ appropriate elements of $P$.

---

**Question:** How can we get the sum of this area?

<table>
<thead>
<tr>
<th>Array A (original array)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>
The Basic Range-Sum Algorithm

- When \( d = 2 \), the range-sum \( \text{Sum}(l_1:h_1, l_2:h_2) \) can be obtained in three computation steps as

\[
\text{Sum}(l_1:h_1, l_2:h_2) = \text{P}[h_1,h_2] - \text{P}[h_1,l_2-1] - \text{P}[l_1-1,h_2] + \text{P}[l_1-1,l_2-1]
\]

\( 2^2 - 1 = 3 \) steps
The Basic Range-Sum Algorithm

For instance:
The range-sum $\text{Sum}(2:3,1:2) = P[3,2] - P[3,0] - P[1,2] + P[1,0]$
$= 40 - 11 - 24 + 8 = 13$

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>8</td>
<td>9</td>
<td>11</td>
<td>13</td>
<td>16</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>18</td>
<td>21</td>
<td>29</td>
<td>39</td>
<td>44</td>
</tr>
<tr>
<td>1</td>
<td>12</td>
<td>24</td>
<td>29</td>
<td>40</td>
<td>53</td>
<td>63</td>
</tr>
</tbody>
</table>

- Introduction
- The Basic Range-Sum Algorithm
- The Blocked Range-Sum Algorithm
- The Batch-Update Algorithm for Range-Sum Queries
- Conclusion
The Blocked Range-Sum Algorithm

- To save space as a trade-off to time is to keep Prefix-Sums at a coarser-grained (block) level.
- Store the Prefix-Sum only when every index is either one less than some multiple of b or the last index.
- If the blocked algorithm is used, the original array A cannot be dropped.

For Example, in the two-dimensional case, only $P_{[b-1,b-1]}, P_{[b-1,2b-1]}, \ldots, P_{[b-1,n^2-1]}, P_{[2b-1,b-1]}, P_{[2b-1,2b-1]}$, etc.

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>18</td>
<td>-</td>
<td>29</td>
<td>-</td>
<td>44</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>24</td>
<td>-</td>
<td>40</td>
<td>-</td>
<td>63</td>
</tr>
</tbody>
</table>

Example of the blocked Prefix-Sum array $P$ with $b=2$
The Blocked Range-Sum Algorithm

- How to compute $\text{Sum}(l_1:h_1,l_2:h_2)$.
  Intuitively, decompose the range in each dimension into $3^d$ disjoining sub-ranges where the middles sub-range is properly aligned with the block structure.

For example, we want to query Region(50:350,50:350)
The Blocked Range-Sum Algorithm

This diagram shows the $3^2=9$ decomposed regions for this query. A5 is an **internal region** and all other 8 regions are **boundary regions**.

Since the internal region is properly aligned with the block structure, its range-sum can be computed in up to $2^d-1$ step based on the Prefix-Sum array $P$.

For any boundary region, there are two possible methods for computation:

1. sum up all elements of $A$ corresponding to the boundary regions.

2. One can sum up all elements of $A$ corresponding to the complement region, then subtract the sum from the range-sum for the super-block region. The range-sum for a super-block region can be computed in $2^d-1$ step using Prefix-Sum array $P$. 
The Blocked Range-Sum Algorithm

The boundary region = super-block region – complement region

To minimize the time complexity, the algorithm will choose the first method when the volume of the target region is smaller than or equal to “the volume of its complement region plus 2d-1”; and will choose the second method, otherwise.
The Blocked Range-Sum Algorithm

For example: consider the query Sum(75:374,100:357)

- Method 1: Compute the boundary directly.
- Method 2: Compute the super block and complement block, then subtract to find the boundary block.

Introduction
- The Basic Range-Sum Algorithm
- The Blocked Range-Sum Algorithm
- The Batch-Update Algorithm for Range-Sum Queries
- Conclusion
The Batch-Update Algorithm for Range-Sum Queries

- **Why Batch-Update?**
  - In a typical OLAP environment, updates to the data cube are accumulated over a period of time, then these updates are performed together as a batch at the end of each period.
  - Assume a model that $k$ update queries are issued successively before the next read-only query is issued.
  - Batch all the updates together and perform a “combined” update to array $P$.

The basic algorithm (block size=1)
When a query updates an array element $A[x_1, \ldots, x_d]$, all $P[y_1, \ldots, y_d]$, where $y_j > x_j$ will be “affected”.
The Batch-Update Algorithm for Range-Sum Queries

- For each update in the form of (location of the element of A, value-to-add), update the corresponding A element right away
- For d=1, updates in the form (u_i, v_i)
- queue an update form to be used later for a combined update of P
The Batch-Update Algorithm for Range-Sum Queries

- Two elements are in the same update-class if they are affected by the same subset of k updates.
- For example

```
A  B  C
1  2
```

The same update class

The goal is to group all “affected” elements of array P into minimum number of disjoint regions

- Property 1: All elements of P in the same region are in the same update-class.
- Property 2: Each region has a shape of a d-dimensional rectangular cube.
The Batch-Update Algorithm for Range-Sum Queries

Example of partitioned region for $k=3$ and $d=2$

Example of partitioned region for $k=3$ and $d=2$

- Assume $d=1$
- Sort the indices of $k$ updates in the ascending order: $u_1, u_2, ..., u_k$
- Partition the index space of $P$ into $k+1$ adjoining regions according to the sorted $k$ indices:

  $\begin{array}{ccc}
  0 & u_1 & u_2 & ... & u_k & n_1 \\
  \text{Reg 0} & \text{Reg 1} & \text{Reg k} \\
  \end{array}$

- All elements of $P$ in the same region are in the same update-class.
- Except region 0, all other regions are affected.
- Perform combined updates one region at a time starting from region 1.
The Batch-Update Algorithm for Range-Sum Queries

- For $d > 1$, recursively call a number of batch-update algorithms for dimensions $d-1$
- Theorem: The batch-update algorithm will group all affected elements of $P$ into up to:
  
  $d - 1 \prod_{j=0}^{d-1} \left( \frac{n + j}{d!} \right)$

Thus, the index space of $A$ has been contracted by a factor of $b$ in every dimension

The Batch-Update Algorithm for Range-Sum Queries

1. For every $d$-dimensional block of from $b^*b^*...b$, sum up all values-of-add for all updates of $A$ in the block
   - Thus, the index space of $A$ has been contracted by a factor of $b$ in every dimension
2. apply the batch-update algorithm
   - Treating each block of $A$ as one element of $A$
   - Treating the combined values-to-add for each block as a value-to-add for each element of $A$
   - Treating the blocked Prefix-Sum array $P$ as the basic Prefix-Sum array $P$
The Batch-Update Algorithm for Range-Sum Queries

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>18</td>
<td>-</td>
<td>29</td>
<td>-</td>
<td>44</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>24</td>
<td>-</td>
<td>40</td>
<td>-</td>
<td>63</td>
</tr>
</tbody>
</table>

**Array P**

<table>
<thead>
<tr>
<th>Index</th>
<th>1</th>
<th>3</th>
<th>5</th>
<th>7</th>
<th>9</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18</td>
<td>29</td>
<td>44</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>24</td>
<td>40</td>
<td>63</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- **Introduction**
- **The Basic Range-Sum Algorithm**
- **The Blocked Range-Sum Algorithm**
- **The Batch-Update Algorithm for Range-Sum Queries**
- **Conclusion**
Conclusion

- For speeding up range sum queries: pre-compute multidimensional Prefix-Sums of the data cube. Then, any range-sum query can be answered by accessing 2\textsuperscript{d} appropriate Prefix-Sums.