Chapter 5, Data Cube Computation

Young-Rae Cho
Associate Professor
Department of Computer Science
Baylor University

A Roadmap for Data Cube Computation

- **Full Cube**
  - Full materialization
  - Materializing all the cells of all of the cuboids for a given data cube
  - Issues in time and space

- **Iceberg Cube**
  - Partial materialization
  - Materializing the cells of only interesting cuboids
  - Materializing only the cells in a cuboid whose measure value is above the minimum threshold

- **Closed Cube**
  - Materializing only closed cells
Typical Computation Process

- **Example**

![Diagram showing various combinations of time, item, location, and supplier]

- **Aggregating**
  - Aggregating from the smallest child cuboid

- **Caching**
  - Caching the result of a cuboid for the computation of other cuboids to reduce disk I/O

- **Sorting, Hashing and Grouping**
  - Sorting, hashing, and grouping operations are applied to a dimension in order to reorder and cluster

- **Pruning**
  - A priori pruning the cells with lower support than minimum threshold

Computation Techniques
Chapter 5, Data Cube Computation

- Multi-way Array Aggregation
- BUC: Iceberg Cube Computation
- Star-Cubing
- Shell Fragment Cube Computation

Multi-Way Array Aggregation

- **Features**
  - Array-based "bottom-up" approach
  - Uses multi-dimensional chunks
  - No direct tuple comparisons
  - Simultaneous aggregation on multiple dimensions
  - Intermediate aggregate values are re-used for computing ancestor cuboids
  - Full materialization (No iceberg optimization, No Apriori pruning)
Aggregation Strategy

- **Partitions array into chunks**
  - Chunk: a small sub-cube which fits in memory

- **Data addressing**
  - Uses chunk id and offset

- **Multi-way Aggregation**
  - Computes aggregates in multi-way
  - Visits chunks in the order
    - to minimize memory access
    - to minimize memory space

---

Aggregation Process

- **Example**
Aggregation Process

Example - Continued

Optimization of Memory Usage

What is the best traversing order?
- Depends on the memory space required

Example
- Suppose the data size on each dimension A, B and C is 40, 400 and 4000, respectively.
- Minimum memory required when traversing the order, 1,2,3,4,5,..., 64
- Total memory required is $100 \times 1000 + 40 \times 1000 + 40 \times 400$
Summary of Multi-Way

- **Method**
  - Cuboids should be sorted and computed according to the data size on each dimension.
  - Keeps the smallest plane in the main memory, fetches and computes only one chunk at a time for the largest plane.

- **Problems**
  - Full materialization → extremely large storage is required.
  - Computes well only for a small number of dimensions (high dimensional data → partial materialization).

- **Reference**

---

**Chapter 5, Data Cube Computation**

- Multi-way Array Aggregation
- BUC: Iceberg Cube Computation
- Star-Cubing
- Shell Fragment Cube Computation
Bottom-Up Computation (BUC)

- **Characteristics**
  - “Top-down” approach
  - Partial materialization (iceberg cube computation)
  - Divides dimensions into partitions and facilitates iceberg pruning
  - No simultaneous aggregation

Iceberg Pruning Process

- **Partitioning**
  - Sorts data values
  - Partitions into blocks that fit in memory

- **Apriori Pruning**
  - For each block
    - If it does not satisfy min_sup, its descendants are pruned
    - If it satisfies min_sup, materialization and a recursive call including the next dimension
Apriori Pruning Example

- **Description in SQL**
  - Iceberg cube computation

```sql
compute cube iceberg_cube as
select A, B, C, D, count(*)
from R
cube by A, B, C, D
having count(*) ≥ min_sup
```

- **Example**

<table>
<thead>
<tr>
<th></th>
<th>a1</th>
<th>b1</th>
<th>c1</th>
<th>d1</th>
<th>d2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a2</td>
<td>b3</td>
<td>c2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- (*, *, *, *) → 0-D cuboid
- (a1, *, *, *) → 1-D cuboid
- (a1, b1, *, *) → 2-D cuboid
- (a1, b1, c1, *) → 3-D cuboid
- (a1, b1, c1, d1) → pruning
- (a1, b1, c2, *) → 3-D cuboid
- (a1, b2, *, *) → pruning

Summary of BUC

- **Method**
  - Computation of sparse data cubes

- **Problems**
  - Sensitive to the order of dimensions
    - The most discriminating dimension should be used first
    - Dimensions should be in the order of decreasing cardinality
    - Dimensions should be in the order of increasing maximum number of duplicates

- **Reference**
Chapter 5, Data Cube Computation

- Multi-way Array Aggregation
- BUC: Iceberg Cube Computation
- Star-Cubing
- Shell Fragment Cube Computation

Characteristics of Star-Cubing

- **Characteristics**
  - Integrated method of “top-down” and “bottom-up” cube computation
  - Explores both multidimensional aggregation (as Multi-way) and apriori pruning (as BUC)
  - Explores **shared dimensions**
    - e.g., dimension A is a shared dimension of ACD and AD
    - e.g., ABD/AB means ABD has the shared dimension AB
Iceberg Pruning Strategy

**Iceberg Pruning in a Shared Dimension**
- If AB is a shared dimension of ABD, then the cuboid AB is computed simultaneously with ABD
- If the aggregate on a shared dimension does not satisfy the iceberg condition ($min\_sup$), then all the cells extended from this shared dimension cannot satisfy the condition either
- If we can compute the shared dimensions before the actual cuboid, we can apply Apriori pruning
  → Pruning while aggregating simultaneously on multiple dimensions

Cuboid Trees

**Cuboid Trees**
- Tree structure to represent cuboids
- Base cuboid tree, 3-D cuboid trees, 2-D cuboid trees, ...
- Each level represents a dimension
- Each node represents an attribute
- Each node includes the attribute value, aggregate value, descendant(s)
- The path from the root to a leaf represents a tuple
- Example
  - $\text{count}(a1,b1,*,*) = 10$
  - $\text{count}(a1,b1,c1,*) = 5$
Star Nodes

- **Star Nodes**
  - If the single dimensional aggregate does not satisfy min_sup,
    no need to consider the node in the iceberg cube computation
  - The nodes are replaced by *
  - Example (min_sup = 2)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>*</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>a1</td>
<td>b1</td>
<td>*</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>a1</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>a2</td>
<td>*</td>
<td>c3</td>
<td>d4</td>
<td>1</td>
</tr>
<tr>
<td>a2</td>
<td>*</td>
<td>c3</td>
<td>d4</td>
<td>1</td>
</tr>
<tr>
<td>a1</td>
<td>*</td>
<td>c3</td>
<td>d4</td>
<td>2</td>
</tr>
<tr>
<td>a1</td>
<td>*</td>
<td>c3</td>
<td>d4</td>
<td>2</td>
</tr>
</tbody>
</table>

A B C D count

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>*</td>
<td>*</td>
<td>2</td>
</tr>
<tr>
<td>a1</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>a2</td>
<td>*</td>
<td>c3</td>
<td>d4</td>
<td>2</td>
</tr>
</tbody>
</table>

Star Tree Structure

- **Star Tree**
  - A cuboid tree that is compressed using star nodes

- **Star Tree Construction**
  - Uses the compressed table
  - Keeps the star table for lookup of star nodes
  - Lossless compression from the original cuboid tree

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>b2</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b3</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b4</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c1</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c2</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c3</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c4</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d1</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d2</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d3</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d4</td>
<td>*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Star Table
Multi-Way Star-Tree Aggregation

- **Aggregation**
  - DFS (depth-first-search) from the root of a star tree
  - Creates star trees for the cuboids on the next level

- **Pruning**
  - Prunes if the aggregates do not satisfy min_sup
  - Prunes if all the nodes in the generated tree are star nodes

Implementation of Star Cubing

- **Method**
  - Multi-way aggregation & iceberg pruning
Summary of Star Cubing

- **Problems**
  - Sensitive to the order of dimensions
    - The order of decreasing cardinality

- **Reference**
  - Xin, D., Han, J., Li, X. and Wah, B.W., “Star-Cubing: Computing Iceberg Cubes by Top-Down and Bottom-Up Integration”, In Proceedings of VLDB (2003)

---

**Chapter 5, Data Cube Computation**

- Multi-way Array Aggregation
- BUC: Iceberg Cube Computation
- Star-Cubing
- Shell Fragment Cube Computation
Shell Fragment Cube Computation

- **Motivations**
  - The computation and storage of iceberg cube are still costly for high dimensional data ("the curse of dimensionality")
  - Hard to determine an appropriate iceberg threshold
  - No update in an iceberg cube

- **Features**
  - Reduces a high dimensional cube into a set of lower dimensional cubes
  - Lossless reduction
  - Online re-construction of high-dimensional data cube

Fragmentation Strategy

- **Observation**
  - OLAP occurs only on a small subset of dimensions at a time

- **Fragmentation**
  - Partitions the set of dimensions into shell fragments
  - Computes data cubes for each shell fragment
  - 20 3-D data cube computation is much better than 1 60-D data cube.
  - Retains inverted indices or value-list indices

- **Semi-Online Computation**
  - Given the pre-computed fragment cubes, dynamically compute cube cells of the high-dimensional cube online
Inverted Index

- **Example**
  - Divides the 5 dimensions into 2 shell fragments, (A, B, C) and (D, E)
  - Builds the inverted index (1-D inverted index)

<table>
<thead>
<tr>
<th>TID</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
<td>e1</td>
</tr>
<tr>
<td>2</td>
<td>a1</td>
<td>b2</td>
<td>c1</td>
<td>d2</td>
<td>e1</td>
</tr>
<tr>
<td>3</td>
<td>a1</td>
<td>b2</td>
<td>c1</td>
<td>d1</td>
<td>e2</td>
</tr>
<tr>
<td>4</td>
<td>a2</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
<td>e2</td>
</tr>
<tr>
<td>5</td>
<td>a2</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
<td>e3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>TID List</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>1 2 3</td>
<td>3</td>
</tr>
<tr>
<td>a2</td>
<td>4 5</td>
<td>2</td>
</tr>
<tr>
<td>b1</td>
<td>1 4 5</td>
<td>3</td>
</tr>
<tr>
<td>b2</td>
<td>2 3</td>
<td>2</td>
</tr>
<tr>
<td>c1</td>
<td>1 2 3 4 5</td>
<td>5</td>
</tr>
<tr>
<td>d1</td>
<td>1 3 4 5</td>
<td>4</td>
</tr>
<tr>
<td>d2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>e1</td>
<td>1 2</td>
<td>2</td>
</tr>
<tr>
<td>e2</td>
<td>3 4</td>
<td>2</td>
</tr>
<tr>
<td>e3</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

Cube Computation

- **Process**
  - Generalizes the 1-D inverted indices to multi-dimensional inverted indices
  - Computes cuboids using the inverted indices

- **Example**
  - Shell fragment cube ABC contains 7 cuboids, A, B, C, AB, AC, BC, ABC
  - The cuboid AB is computed using the 2-D inverted index below

<table>
<thead>
<tr>
<th>Cell</th>
<th>Intersection</th>
<th>TID List</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a1,b1)</td>
<td>(1,2,3)∩(1,4,5)</td>
<td>{1}</td>
<td>1</td>
</tr>
<tr>
<td>(a1,b2)</td>
<td>(1,2,3)∩(2,3)</td>
<td>{2,3}</td>
<td>2</td>
</tr>
<tr>
<td>(a2,b1)</td>
<td>(4,5)∩(1,4,5)</td>
<td>{4,5}</td>
<td>2</td>
</tr>
<tr>
<td>(a2,b2)</td>
<td>(4,5)∩(2,3)</td>
<td>{}</td>
<td>0</td>
</tr>
</tbody>
</table>
Online Query Computation

- **Process**
  - Given shell fragment cubes
  - Divides the query into shell fragments
  - Fetches the corresponding TID list for each fragment from the cubes
  - Intersects the TID lists from each fragment to construct instantiated base table
  - Computes the **online cube** using the base table with any data cube computation algorithm
  - Computes the query

Shell Fragment Cube Process

Dimensions

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>...</th>
</tr>
</thead>
</table>

ABC Cube

DEF Cube

<table>
<thead>
<tr>
<th>Cell</th>
<th>T-ID List</th>
</tr>
</thead>
<tbody>
<tr>
<td>d1 e1</td>
<td>(1, 3, 8, 9)</td>
</tr>
<tr>
<td>d1 e2</td>
<td>(2, 4, 6, 7)</td>
</tr>
<tr>
<td>d2 e1</td>
<td>(5, 10)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Online Query Process

- ABCDEF
- GHIJKL
- MN...

Instantiated Base Table → Online Cube

Summary of Shell Fragment Cube

- **Advantages**
  - Significant increase of the speed of data cube computation
  - Significant decrease of the storage usage
  - Various applications in very high dimensional data

- **Problems**
  - Tradeoffs between the preprocessing time to construct a data cube and the online computation time for queries
  - Tradeoffs between the storage for a data cube and the memory for online computation

- **Reference**
Questions?

- Lecture Slides are found on the Course Website,
  www.ecs.baylor.edu/faculty/cho/4352