### The Data

**Fact (Measure)**
- Sales

**Dimensions**
- **Month:** 1, 2, 3, 4
- **City:** LA, NY, LV, MI
- **Item:** 1, 2, 3

<table>
<thead>
<tr>
<th>Item</th>
<th>Month</th>
<th>City</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jan</td>
<td>LA</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>Feb</td>
<td>LA</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>Jan</td>
<td>NY</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>Mar</td>
<td>NY</td>
<td>200</td>
</tr>
<tr>
<td>1</td>
<td>Mar</td>
<td>LV</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>Apr</td>
<td>MI</td>
<td>150</td>
</tr>
</tbody>
</table>

### The Multidimensional Model

<table>
<thead>
<tr>
<th>Item</th>
<th>Month</th>
<th>City</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jan</td>
<td>LA</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>Feb</td>
<td>LA</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>Jan</td>
<td>NY</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>Mar</td>
<td>NY</td>
<td>200</td>
</tr>
<tr>
<td>1</td>
<td>Mar</td>
<td>LV</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>Apr</td>
<td>MI</td>
<td>150</td>
</tr>
</tbody>
</table>

### A Cuboid

**3-D cuboid** \{item, month, city\}

<table>
<thead>
<tr>
<th>Item</th>
<th>Month</th>
<th>City</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jan</td>
<td>LA</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>Feb</td>
<td>LA</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>Jan</td>
<td>NY</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>Mar</td>
<td>NY</td>
<td>200</td>
</tr>
<tr>
<td>1</td>
<td>Mar</td>
<td>LV</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>Apr</td>
<td>MI</td>
<td>150</td>
</tr>
</tbody>
</table>

### Data Cube

**A lattice of cuboids**

- **0-D (apex) cuboid**
- **1-D cuboid**
- **2-D cuboid**
- **3-D (base) cuboid**
About the Data Cube

- # of cuboids??
- # of cells in each cuboid??
- How do a few records turn into so much data??

Observations and Solutions

- Observations
  - Curse of Dimensionality
  - Sparsity
  - Closed coverage

- Solutions
  - Partial computation of data cube
    - Iceberg Cube
    - Cube Shell / shell fragments
  - Cube compression
  - Closed Cube

Cell

- A cell in an n-dimensional cube: 
  \((a_1, a_2, ..., a_n, \text{measure})\)
- \(a_i\) is either a value or *
- A cell is a \(m\)-dimensional cell if exactly \(m\) values in \((a_1, a_2, ..., a_n)\) are not *
- Base cell: \(m=n\)
- Aggregate cell: \(m<n\)

Cell Examples

- C1: (*,*,LA,150)
- C2: (2,* ,LA,50)
- C3: (1,Jan,LA,100)
- C4: (1,*,NY,230)
- C5: (*,*,NY,230)

Ancestor and Descendent Cells

- An \(i\)-D cell \(a=(a_1, a_2, ..., a_n, \text{measure}_a)\) is an ancestor of a \(j\)-D cell
  \(b=(b_1, b_2, ..., b_n, \text{measure}_b)\) iff
    - \(i<j\), and
    - For \(1 \leq m \leq n\), \(a_m=b_m\) whenever \(a_m\neq\ast\)
- \(a\) is a parent of \(b\) (and \(b\) a child of \(a\))
  - \(a\) is an ancestor of \(b\), and
  - \(j=i+1\)

Ancestor and Descendent Examples

- C1: (*,*,LA,150)
- C2: (2,* ,LA,50)
- C3: (1,Jan,LA,100)
- C4: (1,*,NY,230)
- C5: (*,*,NY,230)
Closed Cell

A cell $c$ is a closed cell if there is no descendant of $c$ that has the same measure as $c$.

Closed Cube

A closed cube is a data cube consisting of only closed cells.

What's the closed cube of the following data??

<table>
<thead>
<tr>
<th>item</th>
<th>month</th>
<th>city</th>
<th>sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jan</td>
<td>LA</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>Feb</td>
<td>LA</td>
<td>50</td>
</tr>
</tbody>
</table>

Closed Cell Examples

Which of the following are closed cells?

- C1: (*,*,LA,150)
- C2: (2,*,LA,50)
- C3: (1,Jan,LA,100)
- C4: (1,*,NY,230)
- C5: (*,*,NY,230)

Full Cube Computation Example – Dimensions

<table>
<thead>
<tr>
<th>item (a)</th>
<th>month (b)</th>
<th>city (c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td>c2</td>
</tr>
<tr>
<td>a3</td>
<td>b3</td>
<td>c3</td>
</tr>
<tr>
<td>a4</td>
<td>b4</td>
<td>c4</td>
</tr>
</tbody>
</table>

Cell examples:

- $(a_2,b_1,c_1,100)$
- $(a_2,*,c_3,??)$
- $(*,b_4,*,??)$

Query a Closed Cube

- (1,Jan,LA,??)
- (1,*,LA,??)
- (1,*,NY,??)
- (2,*,*,??)
- (*,*,*,??)

Full Cube Computation Example – Data Cube
Full Cube Computation
- Approach 1: one cuboid (i.e. group-by) at a time
  - 2^n-1 scans
- Approach 2: single scan??

Naïve Single Scan ...
- Create all cube cells in memory and initialize them to 0
- Read in each record and update corresponding cells

... Naïve Single Scan
- For example, after reading (a1,b1,c1,100), the following cells will be updated:
  - (a1,b1,*), (a1,*,c1), (*,b1,c1)
  - (a1,*,*), (*,b1,*)
  - (*,*,*)

Reduce Memory Requirement - Order Matters
- Cells need to be kept in memory
  - Read unsorted: 52
  - Read sorted: ??

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>100</td>
</tr>
<tr>
<td>a1</td>
<td>b1</td>
<td>c2</td>
<td>30</td>
</tr>
<tr>
<td>a1</td>
<td>b3</td>
<td>c2</td>
<td>200</td>
</tr>
<tr>
<td>a1</td>
<td>b3</td>
<td>c3</td>
<td>100</td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td>c1</td>
<td>50</td>
</tr>
<tr>
<td>a3</td>
<td>b4</td>
<td>c4</td>
<td>150</td>
</tr>
</tbody>
</table>

Multiway Array Aggregation
- Use a multidimensional array store the base cuboid
- Partition the array into chunks such that each chunk can fit into the memory
- Read in each chunk in certain order to compute the aggregates

MAA Example – Data
- Three dimensions
  - A: cardinality=40, partitions=4
  - B: cardinality=400, partitions=4
  - C: cardinality=4000, partitions=4
MAA Example – 3D to 2D

- To compute all the 2D cells, which of the ordering of the chunks is the best?
  - Sort by $a$, $b$, $c$
  - Sort by $b$, $a$, $c$
  - Sort by $c$, $b$, $a$

Iceberg Cubes

- Data cubes that contain only cells with aggregates greater than a minimum threshold (minimum threshold support, or minimum support)

The Apriori Property

- If a cell does not satisfy minimum support, then no descendant of the cell can satisfy the minimum support
- Antimonotonic aggregation functions
  - E.g. count, sum
- Non-antimonotonic aggregation functions
  - E.g. avg

BUC

- Bottom-Up Construction
- An algorithm to compute iceberg cubes with antimonotonic measures
- It’s actually top-down in our view of the lattice of cuboids

BUC Example

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>$b_1$</td>
<td>$c_1$</td>
<td>5</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_1$</td>
<td>$c_1$</td>
<td>10</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$b_2$</td>
<td>$c_1$</td>
<td>3</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_1$</td>
<td>$c_1$</td>
<td>6</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$b_2$</td>
<td>$c_2$</td>
<td>4</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_1$</td>
<td>$c_2$</td>
<td>4</td>
</tr>
</tbody>
</table>

- Compute an iceberg cube with $\text{sum} > 5$

BUC Outline ...

- Aggregate all the input records

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>$b_1$</td>
<td>$c_1$</td>
<td>5</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_1$</td>
<td>$c_1$</td>
<td>10</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$b_2$</td>
<td>$c_1$</td>
<td>3</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_1$</td>
<td>$c_1$</td>
<td>6</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$b_2$</td>
<td>$c_2$</td>
<td>4</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_2$</td>
<td>$c_2$</td>
<td>4</td>
</tr>
</tbody>
</table>

$\Rightarrow (\ast,\ast,\ast,32)$
… BUC Outline …

- Partition the input records on the distinct values of the next dimension

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
<td>b₁</td>
<td>c₁</td>
<td>5</td>
</tr>
<tr>
<td>a₂</td>
<td>b₁</td>
<td>c₂</td>
<td>10</td>
</tr>
<tr>
<td>a₁</td>
<td>b₂</td>
<td>c₁</td>
<td>3</td>
</tr>
<tr>
<td>a₂</td>
<td>b₁</td>
<td>c₁</td>
<td>6</td>
</tr>
<tr>
<td>a₁</td>
<td>b₂</td>
<td>c₂</td>
<td>4</td>
</tr>
<tr>
<td>a₂</td>
<td>b₂</td>
<td>c₁</td>
<td>4</td>
</tr>
</tbody>
</table>

=⇒

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
<td>b₁</td>
<td>c₁</td>
<td>5</td>
</tr>
<tr>
<td>a₂</td>
<td>b₂</td>
<td>c₂</td>
<td>4</td>
</tr>
<tr>
<td>a₁</td>
<td>b₂</td>
<td>c₁</td>
<td>3</td>
</tr>
</tbody>
</table>

=⇒ \((a₄, *, *, 12)\)

… BUC Outline …

- If a partition satisfy the iceberg condition, recursively call BUC using this partition as input

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
<td>b₁</td>
<td>c₁</td>
<td>5</td>
</tr>
<tr>
<td>a₂</td>
<td>b₂</td>
<td>c₁</td>
<td>4</td>
</tr>
<tr>
<td>a₁</td>
<td>b₂</td>
<td>c₁</td>
<td>3</td>
</tr>
</tbody>
</table>

=⇒

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
<td>b₁</td>
<td>c₁</td>
<td>5</td>
</tr>
<tr>
<td>a₂</td>
<td>b₂</td>
<td>c₁</td>
<td>4</td>
</tr>
</tbody>
</table>

=⇒ \((a₄, *, *, 12)\)

BUC (Bottom-Up Construction)

```c
BUC(input, dim)
    aggregate(input) // place result in outputRec
    if input.count() == 1 then
        write outputRec;
        return;
    end

    if C = Carinality(d)
        Partition(input, C, dataCount(d));
        k = 0;
        for d=dim; d<numDims; ++d)
            C = dataCount(d)
            if C >= minSup
                outputRec.dim(d) = input[x].dim(d)
                BUC(input[x], x+1, d+1)
            end
        end
        k = C
    end

    outputRec.dim(d) = all
endfor
```

A Few Optimizations in BUC

- Apriori pruning
- Dimension ordering
- Single record partition

BUC Example

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
<td>b₁</td>
<td>c₁</td>
<td>5</td>
</tr>
<tr>
<td>a₂</td>
<td>b₁</td>
<td>c₂</td>
<td>10</td>
</tr>
<tr>
<td>a₁</td>
<td>b₂</td>
<td>c₁</td>
<td>3</td>
</tr>
<tr>
<td>a₂</td>
<td>b₂</td>
<td>c₁</td>
<td>6</td>
</tr>
<tr>
<td>a₁</td>
<td>b₂</td>
<td>c₂</td>
<td>4</td>
</tr>
<tr>
<td>a₂</td>
<td>b₂</td>
<td>c₁</td>
<td>4</td>
</tr>
</tbody>
</table>

- Aggregates
  * \((*, *, *, 32)\)
  * \((b₉, *, *, 12)\)
  * \((a₅, b₉, *, 7)\)
  * \((a₄, *, c₁, 14)\)
  * \((a₅, b₉, *, 20)\)
  * -
  * \((*, *, *, 21)\)
  * -
  * \((*, *, *, 14)\)
  * -

- Construct an Iceberg cube with \(\text{sum}>5\)
Problems of Iceberg Cubes

- May still be too large
- Minimum support is hard to determine
- Incremental updates require recomputation of the whole cube

Cube Shells

- Observation: most OLAP operations are performed on a small number of dimensions at a time
- A cube shell of a data cube consists of the cuboids up to a certain dimension
  - E.g. all cuboids with 3 dimensions or less in a 60-dimension data cube

Problems with Cube Shells

- They may still be too large
  - E.g. how many cuboids in a 3-D shell of a 60-D data cube?
- They can’t be used to answer queries like
  \[(location, product\_type, supplier, 2004, ?)\]

Shell Fragments

- Compute only parts of a cube shell — shell fragments
- Answer queries using pre-computed or dynamically computed data

Shell Fragment Example

<table>
<thead>
<tr>
<th>std</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a_1</td>
<td>b_1</td>
<td>c_1</td>
<td>d_1</td>
<td>e_1</td>
</tr>
<tr>
<td>2</td>
<td>a_2</td>
<td>b_2</td>
<td>c_1</td>
<td>d_1</td>
<td>e_1</td>
</tr>
<tr>
<td>3</td>
<td>a_3</td>
<td>b_1</td>
<td>c_2</td>
<td>d_1</td>
<td>e_2</td>
</tr>
<tr>
<td>4</td>
<td>a_4</td>
<td>b_1</td>
<td>c_1</td>
<td>d_2</td>
<td>e_2</td>
</tr>
<tr>
<td>5</td>
<td>a_5</td>
<td>b_1</td>
<td>c_1</td>
<td>d_1</td>
<td>e_3</td>
</tr>
</tbody>
</table>

Shell Fragments Construction (1)

- Partition the dimension into non-overlapping groups — fragments
  \[(a,b,c,d,e) \rightarrow (a,b,c) \text{ and } (d,e)\]
Shell Fragments Construction (2)

- Scan the base cuboid and construct an inverted index for each attribute

<table>
<thead>
<tr>
<th>Attribute value</th>
<th>TID list</th>
<th>List size</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_1</td>
<td>(1,2,3)</td>
<td>3</td>
</tr>
<tr>
<td>b_1</td>
<td>(4,5)</td>
<td>2</td>
</tr>
<tr>
<td>b_2</td>
<td>(1,4,5)</td>
<td>3</td>
</tr>
<tr>
<td>b_3</td>
<td>(2,3)</td>
<td>2</td>
</tr>
<tr>
<td>c_1</td>
<td>(1,2,3,4,5)</td>
<td>5</td>
</tr>
<tr>
<td>d_1</td>
<td>(1,3,4,5)</td>
<td>4</td>
</tr>
<tr>
<td>d_2</td>
<td>(2)</td>
<td>1</td>
</tr>
<tr>
<td>e_1</td>
<td>(1,2)</td>
<td>2</td>
</tr>
<tr>
<td>e_2</td>
<td>(3,4)</td>
<td>2</td>
</tr>
<tr>
<td>e_3</td>
<td>(3)</td>
<td>1</td>
</tr>
</tbody>
</table>

Shell Fragments Construction (3) ...

- Compute the full local data cube (except the local apex cuboid) for each fragment
  - Vs. Cube shell??
- Record an inverted index for each cell in the cuboids
  
- \( (a,b,c) \rightarrow a, b, c, ab, ac, bc, abc \)
- \( (d,e) \rightarrow d, e, de \)

... Shell Fragment Construction (3) ...

<table>
<thead>
<tr>
<th>ab cuboid</th>
</tr>
</thead>
</table>

- **Cell**
  - \((a_1,b_2)\)
  - \((a_2,b_1)\)
  - \((a_2,b_2)\)
  - \((a_2,b_3)\)

- **Intersection**
  - \((1,2,3)\) \(\cap\) \((1,4,5)\)
  - \((1,2,3)\) \(\cap\) \((2,3)\)
  - \((4,5)\) \(\cap\) \((1,4,5)\)
  - \((4,5)\) \(\cap\) \((2,3)\)

- **TID List**
  - (1)
  - (2,3)
  - (4,5)
  - (4,5)

- **List Size**
  - 1
  - 2
  - 2
  - 0

- Inverted indexes are built as the cell aggregates are computed
- Apriori property can be used to prune some computation

Query Cube Fragments – Point Query

- Point query: all dimensions are instantiated with either a value or *
- Examples:
  - \((a_1,b_2,c_1,d_2,e_1)\)
  - \((a_1,b_2,c_1,d_2,*\)
  - \((*,b_2,c_1,d_2,*\)

Answering Point Queries

- \((a_1,b_2,c_1,d_2,e_1)\)
  - \(\downarrow\)
  - \(\{2,3\} \cap \{2\}\)
  - \(\downarrow\)
  - \(\{2\}\)
Query Cube Fragments – Subcube Query

- Subcube query: at least one of the dimensions is *inquired* (i.e. a group-by attribute)
- Example:

  2-D data cube on \( a \) and \( e \)

\[(a, b, c, *, e, ?) \rightarrow \text{all} \]

Answering Subcube Queries

\[(a, b, c, *, e, ?) \]

\[a_1: \{1,2,3\} \cap \{2,3\} \cap e_1: \{1,2\}\]

\[a_2: \{4,5\}\]

\[e_2: \{3,4\}\]

Base cuboid of \( ae \)

\[(a_1, e_1) (a_2, e_3) (a_2, e_4) (a_3, e_5) (a_2, e_5)\]

Full cube computation

Data cube on \( a \) and \( e \)

OLAP Storage Types

- Relational OLAP (ROLAP)
- Multidimensional OLAP (MOLAP)
- Hybrid OLAP (HOLAP)

A ROLAP Data Store

**Summary fact tables**

<table>
<thead>
<tr>
<th>RID</th>
<th>Item</th>
<th>Day</th>
<th>Month</th>
<th>Quarter</th>
<th>Year</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>TV</td>
<td>15</td>
<td>10</td>
<td>Q4</td>
<td>2003</td>
<td>250</td>
</tr>
<tr>
<td>1002</td>
<td>TV</td>
<td>23</td>
<td>10</td>
<td>Q4</td>
<td>2003</td>
<td>175</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5001</td>
<td>TV</td>
<td>all</td>
<td>10</td>
<td>Q4</td>
<td>2003</td>
<td>45,786</td>
</tr>
</tbody>
</table>

Summary

- Data cube
  - Cuboid
  - Closed cube
  - Full cube computation
    - Multiway Array Aggregation
- Iceberg cube
  - BUC
- Cube shell fragments
  - Construction
  - Query
- OLAP storage types

Readings

- Textbook Chapter 5.1 and 5.2 except 5.2.3