**The Data Cube**

- **Dimensions**
  - Gender: M, F
  - Age: Below 20, 20-30, 30-40, 40-50, 50-60, above 60
  - Education: Below High School, High School, College, Graduate School
  - Address: LA, NY, SD
- **Aggregation function (Measure)**
  - Average salary

**The Data**

<table>
<thead>
<tr>
<th>rid</th>
<th>gender</th>
<th>age</th>
<th>education</th>
<th>address</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>M</td>
<td>24</td>
<td>High school</td>
<td>LA,CA</td>
<td>100K</td>
</tr>
<tr>
<td>1002</td>
<td>F</td>
<td>25</td>
<td>College</td>
<td>LA,CA</td>
<td>60K</td>
</tr>
<tr>
<td>1003</td>
<td>M</td>
<td>36</td>
<td>College</td>
<td>NY,NY</td>
<td>65K</td>
</tr>
<tr>
<td>1004</td>
<td>M</td>
<td>61</td>
<td>Graduate school</td>
<td>NY,NY</td>
<td>120K</td>
</tr>
<tr>
<td>1005</td>
<td>F</td>
<td>18</td>
<td>College</td>
<td>NY,NY</td>
<td>40K</td>
</tr>
<tr>
<td>1006</td>
<td>F</td>
<td>29</td>
<td>Graduate school</td>
<td>NY,NY</td>
<td>50K</td>
</tr>
<tr>
<td>1007</td>
<td>F</td>
<td>55</td>
<td>High school</td>
<td>SD,CA</td>
<td>35K</td>
</tr>
<tr>
<td>1008</td>
<td>M</td>
<td>45</td>
<td>Middle school</td>
<td>SD,CA</td>
<td>30K</td>
</tr>
</tbody>
</table>

**About The Data Cube**

- **Data in a cuboid??**
  - E.g. cuboid(age,gender),
    cuboid(gender,age,address)
- **# of cuboids??**
- **# of cells??**

**Observations about Data Cubes ...**

- **How did a few tuples turn into so much data?**
  - Many cells contain no data (or 0)
    - E.g. (M, 60+, College, LA)
  - Many aggregation values are the same
    - E.g. (M, 20-30, HS, LA), (M, 20-30, *, LA), and (M, *, *, LA)

**... Observations about Data Cubes**

- **Observations**
  - Curse of Dimensionality
  - Sparsity
  - Closed coverage
- **Solutions**
  - Partial computation of data cube
    - Iceberg Cube
    - Shell Cube
  - Cube compression
A cell in a $n$-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: ($*,*,*,\text{LA},80\text{K}$)
- C2: (M,$*,*,\text{LA},100\text{K}$)
- C3: (M,20-30,HS,LA,100K)
- C4: (F,$*,*,\text{SD},35\text{K}$)
- C5: ($*,*,*,\text{SD},33\text{K}$)

---

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 $a_m \neq *$
- $a$ 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: ($*,*,*,\text{LA},80\text{K}$)
- C2: (M,$*,*,\text{LA},100\text{K}$)
- C3: (M,20-30,HS,LA,100K)
- C4: (F,$*,*,\text{SD},35\text{K}$)
- C5: ($*,*,*,\text{SD},33\text{K}$)

---

Closed Cell

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

---

Closed Cell Examples

- Which of the following are closed cells?
  - C1: ($*,*,*,\text{LA},80\text{K}$)
  - C2: (M,$*,*,\text{LA},100\text{K}$)
  - C3: (M,20-30,HS,LA,100K)
  - C4: (F,$*,*,\text{SD},35\text{K}$)
  - C5: ($*,*,*,\text{SD},33\text{K}$)
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>rid</th>
<th>gender</th>
<th>age</th>
<th>education</th>
<th>address</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>M</td>
<td>24</td>
<td>High school</td>
<td>LA,CA</td>
<td>100K</td>
</tr>
<tr>
<td>1002</td>
<td>F</td>
<td>25</td>
<td>College</td>
<td>LA,CA</td>
<td>60K</td>
</tr>
</tbody>
</table>

Query a Closed Cube

(*)*, (*)*, LA, ??

(*, *, College, SD, ??)

Full Cube Computation

Example – Data

<table>
<thead>
<tr>
<th>rid</th>
<th>gender</th>
<th>age</th>
<th>education</th>
<th>salary (k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>M</td>
<td>20-30</td>
<td>High school</td>
<td>100</td>
</tr>
<tr>
<td>1002</td>
<td>F</td>
<td>20-30</td>
<td>College</td>
<td>60</td>
</tr>
<tr>
<td>1003</td>
<td>M</td>
<td>30-40</td>
<td>College</td>
<td>65</td>
</tr>
<tr>
<td>1004</td>
<td>M</td>
<td>&gt;60</td>
<td>Graduate school</td>
<td>120</td>
</tr>
<tr>
<td>1005</td>
<td>F</td>
<td>&lt;30</td>
<td>College</td>
<td>40</td>
</tr>
<tr>
<td>1006</td>
<td>F</td>
<td>20-30</td>
<td>Graduate school</td>
<td>50</td>
</tr>
<tr>
<td>1007</td>
<td>F</td>
<td>50-60</td>
<td>High school</td>
<td>35</td>
</tr>
<tr>
<td>1008</td>
<td>M</td>
<td>40-50</td>
<td>&lt; High School</td>
<td>30</td>
</tr>
</tbody>
</table>

Full Cube Computation

Example – Dimensions

<table>
<thead>
<tr>
<th>Gender (g)</th>
<th>Age (a)</th>
<th>Education (e)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>M</td>
<td>&lt; High School</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>College</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>20-30</td>
</tr>
<tr>
<td>4</td>
<td>M</td>
<td>30-40</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>40-50</td>
</tr>
<tr>
<td>6</td>
<td>F</td>
<td>&gt;60</td>
</tr>
</tbody>
</table>

Cell examples:

(g₁,a₂,e₃,65), (g₁,*,,*), (*,a₂,*), ??

Full Cube Computation

Example – Data Cube

Approach 1: one group-by at a time
- 2ⁿ scans

Approach 2: single scan??
Single Scan – 3D→2D

\[(g,u,a_1^*,g_1^*,e_1^?,a_2^*,e_2^?,\ldots)\]

Order Matters

**2D Cells need to be kept in memory**
- Read unsorted: 12 + 8 + 24
- Read sorted: ??

\[
\begin{array}{cccc}
\text{rid} & g & a & e \\
1001 & 1 & 2 & 1 \\
1003 & 1 & 3 & 3 \\
1008 & 1 & 4 & 1 \\
1004 & 1 & 6 & 4 \\
1005 & 2 & 1 & 3 \\
1002 & 2 & 2 & 3 \\
1006 & 2 & 2 & 4 \\
1007 & 2 & 5 & 1 \\
\end{array}
\]

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

\[
\text{base cuboid} \rightarrow \text{A chunk}
\]

Order Matters (Again)

- Three dimensions
  - A: cardinality=40, partitions=4
  - B: cardinality=400, partitions=4
  - C: cardinality=4000, partitions=4
- Consider the following orders
  - \(a_1\cdot b_2\cdot c_3\cdot a_2\cdot b_3\cdot c_4\cdot a_3\cdot b_4\cdot c_5\cdot a_4\cdot b_5\cdot c_6\cdot a_5\cdot b_6\cdot c_7\cdot a_6\cdot b_7\cdot c_8\cdot a_7\cdot \)
  - \(a_1\cdot c_2\cdot b_3\cdot a_2\cdot c_4\cdot b_4\cdot c_5\cdot a_3\cdot c_6\cdot b_6\cdot c_7\cdot a_4\cdot c_8\cdot b_8\cdot c_9\cdot a_5\cdot \)
  - \(b_1\cdot a_2\cdot c_3\cdot b_2\cdot a_4\cdot c_5\cdot b_4\cdot a_6\cdot c_7\cdot b_6\cdot a_8\cdot c_9\cdot b_8\cdot \)

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
- Anti-monotonic aggregation functions
  - E.g. \(\text{count}, \text{sum}\)
- \(\text{Non-monotonic}\) aggregation functions??
BUC (Bottom-Up Construction)

BUC(input, dim)
aggregate(input); // place result in outputRec
if(input.count() == 1)
    WriteAncestors(input[0],dim); return;
endif
write outputRec
for(d=dim; d < numDims; ++d)
    C = cardinality[d]
    Partition(input, d, C, dataCount[d])
    k=0
    for(i=0; i < C; ++i)
        c = dataCount[d][i]
        if c >= min_sup
            outputRec.dim[d] = input[k].dim[d]
            BUC(input[k…k+c],d+1)
        endif
        k += c
    endfor
    outputRec.dim[d] = all
endfor

Aggregates

(*,*,*,32)
(rhombus4)
(a1,*,*,12)
(square6)
(a1,b1,*,5)
(square6)
(a1,b2,*,7)
(square6)
(a1,b2,c1,3)
(square6)
(a1,b2,c2,4)
(rhombus4)
(*,b1,*,21)
(square6)
(*,c1,14)

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>c1</td>
<td>10</td>
</tr>
<tr>
<td>a1</td>
<td>b2</td>
<td>c1</td>
<td>3</td>
</tr>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>6</td>
</tr>
<tr>
<td>a1</td>
<td>b2</td>
<td>c3</td>
<td>4</td>
</tr>
<tr>
<td>a1</td>
<td>b2</td>
<td>c4</td>
<td>4</td>
</tr>
</tbody>
</table>

Construct an Iceberg cube with sum>5

About BUC

◆ It is actually Top-Down
◆ Dimensions should be processed in order of decreasing cardinality
◆ Take advantage of the Apriori property
◆ Does not share computation costs between parent and child group-bys (unlike Star-Cubing)

Dealing with Non-Monotonic Measures

◆ Example: Compute an Iceberg cube of count(*) ≥ 2 and average(salary) > 40k

Transform Non-monotonic Measures

◆ Cell $c$ covers $n$ non-empty base cells
◆ $\text{avg}^k(c)$: the average of top $k$ base cells covered by $c$
◆ $\text{count}_{\geq 2k}$ and $\text{avg}_{\geq 2x}$ $\Rightarrow$ $\text{avg}^k(c)_{\geq 2x}$
   ◆ What if we remove the $\text{count}_{\geq 2k}$ condition??

Problems of Iceberg Cubes

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

Shell Fragments Computation (1)

- Partition the dimension into non-overlapping groups – fragments
  - (a, b, c, d, e) → (a, b, c) and (d, e)

Shell Fragments Computation (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>a_2</td>
<td>(4,5)</td>
<td>2</td>
</tr>
<tr>
<td>b_1</td>
<td>(1,4,5)</td>
<td>3</td>
</tr>
<tr>
<td>b_2</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>(2,4)</td>
<td>2</td>
</tr>
<tr>
<td>e_3</td>
<td>(5)</td>
<td>1</td>
</tr>
</tbody>
</table>

Shell Fragment Example

<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>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_1</td>
<td>b_1</td>
<td>c_1</td>
<td>d_1</td>
<td>e_1</td>
</tr>
<tr>
<td>3</td>
<td>a_1</td>
<td>b_1</td>
<td>c_1</td>
<td>d_1</td>
<td>e_1</td>
</tr>
<tr>
<td>4</td>
<td>a_1</td>
<td>b_1</td>
<td>c_1</td>
<td>d_1</td>
<td>e_1</td>
</tr>
<tr>
<td>5</td>
<td>a_1</td>
<td>b_1</td>
<td>c_1</td>
<td>d_1</td>
<td>e_1</td>
</tr>
</tbody>
</table>
Shell Fragments Computation (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
  \[(a,b,c) \rightarrow a, b, c, ab, bc, abc\]
  \[(d,e) \rightarrow d, e, de\]

- 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

\[(a_2, b_2, c_1, d_2, e_1) \rightarrow \langle 2,3 \rangle \cap \langle 2 \rangle \]
\[\langle 2 \rangle \rightarrow \langle * \rangle, b_2, c_1, d_2, * \]

---

Query Cube Fragments – Subcube Query

- Subcube query: at least one of the dimensions is inquired (i.e. a group-by attribute)
- Example:
  \[(a, b_2, c_1, *, e, ?) \rightarrow \langle 2 \rangle\]
Answering Subcube Queries

(a, b, c, *, e, ??)

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

\[ a_2 \cap \{4,5\} \cap e_2 \cap \{3,4\} \]

\[ e_3 \cap \{5\} \]

Base cuboid of ae

\[ (a_1, e_1) \cap (a_2, e_2) \cap (a_1, e_1) \cap (a_2, e_2) \cap (a_2, e_3) \]

Full cube computation

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

Summary

- **Closed cube**
- **Full cube computation**
  - Multiway Array Aggregation
- **Iceberg cube**
  - BUC
- **Cube shell fragments**
  - Computation and query

Further Issues in OLAP

- Detect exceptions
- Data visualization and exploration
- Complex aggregations
  - E.g. total sales of highest-priced items group by month and region
- Gradient analysis
  - Changes between probe cells and its ancestors, descendants, and siblings