CS522 Advanced Database Systems
Data Cube Computation

Chengyu Sun
California State University, Los Angeles

The Data

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

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`

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

Cell

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

Cell Examples

- C1: (*,*,*,LA,80K)
- C2: (M,*,*,LA,100K)
- C3: (M,20-30,HS,LA,100K)
- C4: (F,*,*,SD,35K)
- C5: (*,*,*,SD,33K)

Ancestor and Descendent Cells

- An i-D cell $a=(a_1, a_2, \ldots, a_n, measure_a)$ is an ancestor of a j-D cell $b=(b_1, b_2, \ldots, b_n, measure_b)$ iff
  - i<j, and
  - For $1 \le m \le n$, $a_m = b_m$ 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: (*,*,*,LA,80K)
- C2: (M,*,*,LA,100K)
- C3: (M,20-30,HS,LA,100K)
- C4: (F,*,*,SD,35K)
- C5: (*,*,*,SD,33K)

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: (*,*,*,LA,80K)
  - C2: (M,*,*,LA,100K)
  - C3: (M,20-30,HS,LA,100K)
  - C4: (F,*,*,SD,35K)
  - C5: (*,*,*,SD,33K)

Closed Cube

- A closed cube is a data cube consisting of only *closed cells*

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

rid	gender	age	education	address	salary
1001	M	24	High school	LA,CA	100K
1002	F	25	College	LA,CA	60K

Query a Closed Cube

- (*,*,*,*,*??*)
- (*,*,*,LA,*??*)
- (M,*,*,LA,*??*)
- (*,*,College,SD,*??*)

Full Cube Computation Example – Data

rid	gender	age	education	salary (k)
1001	M	20-30	High school	100
1002	F	20-30	College	60
1003	M	30-40	College	65
1004	M	>60	Graduate school	120
1005	F	<20	College	40
1006	F	20-30	Graduate school	50
1007	F	50-60	High school	35
1008	M	40-50	< High School	30

Full Cube Computation Example – Dimensions

Gender (g)	
1	M
2	F

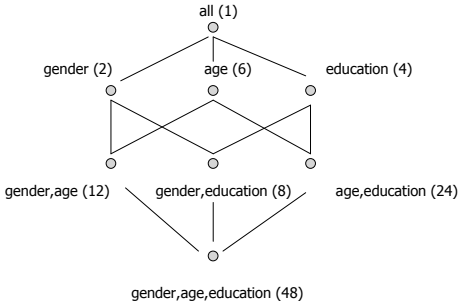
Age (a)	
1	<20
2	20-30
3	30-40
4	40-50
5	50-60
6	>60

Education (e)	
1	< High School
2	High school
3	College
4	Graduate school

Cell examples:

$(g_1,a_3,e_3,65)$ $(g_2,*,e_3,??)$ $(*,a_2,*,??)$

Full Cube Computation Example – Data Cube



Full Cube Computation

- Approach 1: one group-by at a time
  - 2^n scans
- Approach 2: single scan??

Single Scan – 3D→2D

$(g_1,a_1,*,?)$	$(g_1,*,e_1,?)$	$(*,a_1,e_1,?)$
$(g_1,a_2,*,?)$	$(g_1,*,e_2,?)$	$(*,a_1,e_2,?)$
⋮	⋮	⋮
$(g_1,a_6,*,?)$	$(g_1,*,e_4,?)$	$(*,a_1,e_4,?)$
$(g_2,a_1,*,?)$	$(g_2,*,e_1,?)$	$(*,a_2,e_1,?)$
⋮	⋮	⋮
$(g_2,a_6,*,?)$	$(g_2,*,e_4,?)$	$(*,a_6,e_4,?)$

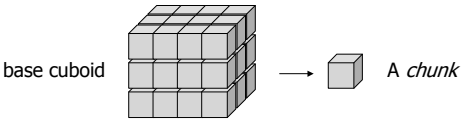
Order Matters

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

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

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



Order Matters (Again)

- Three dimensions
  - A: cardinality=40, partitions=4
  - B: cardinality=400, partitions=4
  - C: cardinality=4000, partitions=4
- Consider the following orders
  - $a_0b_0c_0,a_1b_0c_0,a_2b_0c_0,\ldots,a_0b_1c_0,\ldots,a_3b_3c_3$
  - $b_0a_0c_0,b_1a_0c_0,b_2a_0c_0,\ldots,b_0a_1c_0,\ldots,b_3a_3c_3$
  - $c_0b_0a_0,c_1b_0a_0,c_2b_0a_0,\ldots,c_0b_1a_0,\ldots,c_3b_3a_3$

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. `count`, `sum`
- *Non-monotonic* aggregation functions??

BUC (Bottom-Up Construction)

```
BUC( input, dim )
   aggregate(input) // place result in outputRec
   if( input.count() == 1 ) the
      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
```

BUC Example

A	B	C	Sum
a_1	b_1	c_1	5
a_2	b_1	c_2	10
a_1	b_2	c_1	3
a_2	b_1	c_1	6
a_1	b_2	c_2	4
a_2	b_2	c_3	4

- Construct an Iceberg cube with `sum>5`

- Aggregates
  - (*,*,*,32)
    - (a_1,*,*,12)
      - ~~(a_1,b_1,*,5)~~
      - (a_1,b_2,*,7)
        - ~~(a_1,b_2,c_1,3)~~
        - ~~(a_1,b_2,c_2,4)~~
    - (a_2,*,*,20)
      - …
    - (*,b_1,*21)
      - …
    - (*,*,c_1,14)
      - …

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
- `count`≥k and `avg`≥x ➔ $\text{avg}^k(c)\geq x$
  - What if we remove the `count`≥k 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-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 the pre-computed data

Shell Fragment Example

tid	a	b	c	d	e
1	a_1	b_1	c_1	d_1	e_1
2	a_1	b_2	c_1	d_2	e_1
3	a_1	b_2	c_1	d_1	e_2
4	a_2	b_1	c_1	d_1	e_2
5	a_2	b_1	c_1	d_1	e_3

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

Attribute value	TID list	List size
a_1	{1,2,3}	3
a_2	{4,5}	2
b_1	{1,4,5}	3
b_2	{2,3}	2
c_1	{1,2,3,4,5}	5
d_1	{1,3,4,5}	4
d_2	{2}	1
e_1	{1,2}	2
e_2	{3,4}	2
e_3	{5}	1

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) ➔ a, b, c, ab, ac, bc, abc
(d,e) ➔ d, e, de

… Shell Fragment Computation (3) …

ab cuboid

Cell	Intersection	TID List	List Size
(a_1,b_1)	{1,2,3} ∩ {1,4,5}	{1}	1
(a_1,b_2)	{1,2,3} ∩ {2,3}	{2,3}	2
(a_2,b_1)	{4,5} ∩ {1,4,5}	{4,5}	2
(a_2,b_2)	{4,5} ∩ {2,3}	{}	0

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

… Shell Fragment Computation (3)

- Using an *ID_measure* array instead of the original database table

TID	Item_count	sum
1	5	70
2	3	10
3	8	20
4	5	40
5	2	30

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

$(a_1, b_2, c_1, d_2, e_1)$

$\{2,3\} \cap \{2\}$

$\{2\}$

$(*, b_2, c_1, d_2, *)$

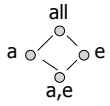
??

Query Cube Fragments – Subcube Query

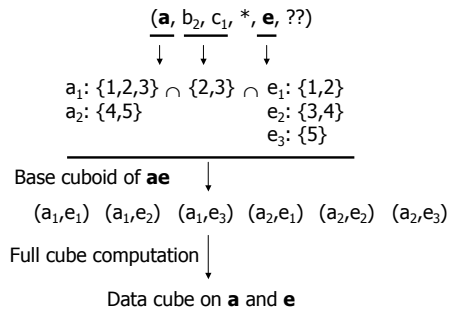
- Subcube query: at least one of the dimensions is *inquired* (i.e. a group-by attribute)
- Example:

(**a**, b_2, c_1, *, **e**, ??) ⟶

2-D data cube on **a** and **e**



Answering Subcube Queries



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, descendents, and siblings