

# CS522 Advanced Database Systems

## Data Cube Computation

Chengyu Sun  
California State University, Los Angeles

## The Data

### Fact (*Measure*)

- Sales

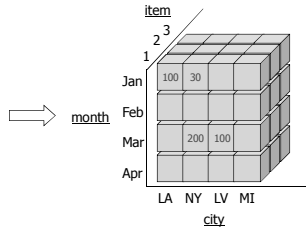
### Dimensions

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

item	month	city	sales
1	Jan	LA	100
2	Feb	LA	50
1	Jan	NY	30
1	Mar	NY	200
1	Mar	LV	100
3	Apr	MI	150

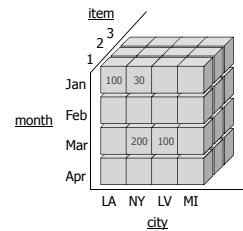
## The Multidimensional Model

item	month	city	sales
1	Jan	LA	100
2	Feb	LA	50
1	Jan	NY	30
1	Mar	NY	200
1	Mar	LV	100
3	Apr	MI	150



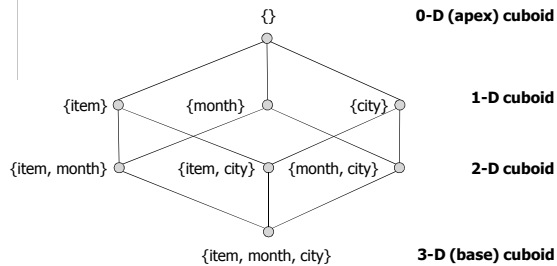
## A Cuboid

### 3-D cuboid {item, month, city}

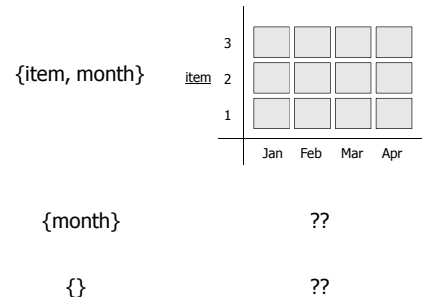


## Data Cube

### A lattice of cuboids



## More Cuboids



## 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 a  $n$ -dimensional cube:  
( $a_1, a_2, \dots, a_n, \text{measure}$ )
- ◆  $a_i$  is either a value or \*
- ◆ A cell is a  $m$ -dimensional cell if exactly  $m$  values in  $\{a_1, a_2, \dots, a_n\}$  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  $a = (a_1, a_2, \dots, a_n, \text{measure}_a)$  is an ancestor of a  $j$ -D cell  $b = (b_1, b_2, \dots, b_n, \text{measure}_b)$  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: (\*, \*, 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 descendent of  $c$  that has the same measure as  $c$

## 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)

## Closed Cube

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

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

item	month	city	sales
1	Jan	LA	100
2	Feb	LA	50

## Query a Closed Cube

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

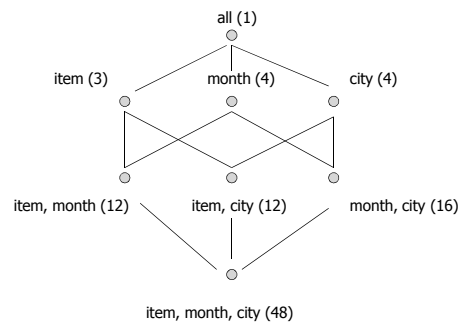
## Full Cube Computation Example – Dimensions

item (a)		month (b)		city (c)	
a1	1	b1	Jan	c1	LA
a2	2	b2	Feb	c2	NY
a3	3	b3	Mar	c3	LV
		b4	Apr	c4	MI

Cell examples:

( $a_1, b_1, c_1, 100$ ) ( $a_2, *, c_3, ??$ ) ( $*, b_2, *, ??$ )

## 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,\*), (\*,\*,c1)
  - (\*,\*,\*)

*Problem with naïve single scan??*

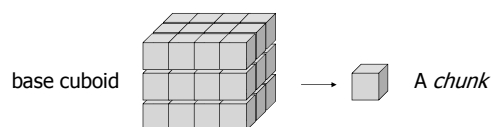
## Reduce Memory Requirement - Order Matters

- ◆ Cells need to be kept in memory
  - Read unsorted: 52
  - Read sorted: ??

a	b	c	sales
a1	b1	c1	100
a1	b1	c2	30
a1	b3	c2	200
a1	b3	c3	100
a2	b2	c1	50
a3	b4	c4	150

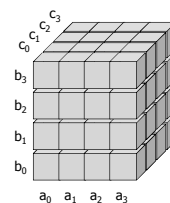
## Multiway Array Aggregation

- ◆ 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

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

- ◆ Compute an iceberg cube with `sum > 5`

## BUC Outline ...

- ◆ Aggregate all the input records

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

⇒ (\*,\*,\*,32)

### ... BUC Outline ...

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

A	B	C	Sum
a <sub>1</sub>	b <sub>1</sub>	c <sub>1</sub>	5
a <sub>2</sub>	b <sub>1</sub>	c <sub>2</sub>	10
a <sub>1</sub>	b <sub>2</sub>	c <sub>1</sub>	3
a <sub>2</sub>	b <sub>1</sub>	c <sub>1</sub>	6
a <sub>1</sub>	b <sub>2</sub>	c <sub>2</sub>	4
a <sub>2</sub>	b <sub>2</sub>	c <sub>3</sub>	4

a <sub>1</sub>	b <sub>1</sub>	c <sub>1</sub>	5
a <sub>1</sub>	b <sub>2</sub>	c <sub>2</sub>	4
a <sub>1</sub>	b <sub>2</sub>	c <sub>1</sub>	3

a <sub>2</sub>	b <sub>1</sub>	c <sub>1</sub>	6
a <sub>2</sub>	b <sub>1</sub>	c <sub>2</sub>	10
a <sub>2</sub>	b <sub>2</sub>	c <sub>3</sub>	4

### ... BUC Outline ...

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

a <sub>1</sub>	b <sub>1</sub>	c <sub>1</sub>	5
a <sub>1</sub>	b <sub>2</sub>	c <sub>2</sub>	4
a <sub>1</sub>	b <sub>2</sub>	c <sub>1</sub>	3

  
 $\Rightarrow (a_1, *, *, 12)$ 

### ... BUC Outline

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

a <sub>1</sub>	b <sub>1</sub>	c <sub>1</sub>	5
a <sub>1</sub>	b <sub>2</sub>	c <sub>2</sub>	4
a <sub>1</sub>	b <sub>2</sub>	c <sub>1</sub>	3

a <sub>1</sub>	b <sub>1</sub>	c <sub>1</sub>	5
a <sub>1</sub>	b <sub>2</sub>	c <sub>2</sub>	4
a <sub>1</sub>	b <sub>2</sub>	c <sub>1</sub>	3

### 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
    endif
    outputRec.dim[d] = all
endifor
    
```

### BUC Example

A	B	C	Sum
a <sub>1</sub>	b <sub>1</sub>	c <sub>1</sub>	5
a <sub>2</sub>	b <sub>1</sub>	c <sub>2</sub>	10
a <sub>1</sub>	b <sub>2</sub>	c <sub>1</sub>	3
a <sub>2</sub>	b <sub>1</sub>	c <sub>1</sub>	6
a <sub>1</sub>	b <sub>2</sub>	c <sub>2</sub>	4
a <sub>2</sub>	b <sub>2</sub>	c <sub>3</sub>	4

#### Aggregates

- (\*,\*,\*,32)
  - (a<sub>1</sub>\*,\*,12)
    - (a<sub>1</sub>b<sub>1</sub>\*,5)
    - (a<sub>1</sub>b<sub>2</sub>\*,7)
    - (a<sub>1</sub>\*,c<sub>1</sub>,8)
    - (a<sub>1</sub>\*,c<sub>2</sub>,4)
  - (a<sub>2</sub>\*,\*,20)
    - ...
  - (\*,b<sub>1</sub>\*,21)
    - ...
  - (\*,\*,c<sub>1</sub>,14)
    - ...

- Construct an Iceberg cube with  $sum > 5$

### A Few Optimizations in BUC

- Apriori pruning
- Dimension ordering
- Single record partition

## Problems of Iceberg Cubes

- ◆ May still be too large
- ◆ Minimum support is hard to determine
- ◆ Incremental updates require re-computation 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

tid	a	b	c	d	e
1	a <sub>1</sub>	b <sub>1</sub>	c <sub>1</sub>	d <sub>1</sub>	e <sub>1</sub>
2	a <sub>1</sub>	b <sub>2</sub>	c <sub>1</sub>	d <sub>2</sub>	e <sub>1</sub>
3	a <sub>1</sub>	b <sub>2</sub>	c <sub>1</sub>	d <sub>1</sub>	e <sub>2</sub>
4	a <sub>2</sub>	b <sub>1</sub>	c <sub>1</sub>	d <sub>1</sub>	e <sub>2</sub>
5	a <sub>2</sub>	b <sub>1</sub>	c <sub>1</sub>	d <sub>1</sub>	e <sub>3</sub>

## Shell Fragments Construction (1)

- ◆ Partition the dimension into *non-overlapping* groups – fragments

(a,b,c,d,e) → (a,b,c) and (d,e)

## Shell Fragments Construction (2)

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

Attribute value	TID list	List size
a <sub>1</sub>	{1,2,3}	3
a <sub>2</sub>	{4,5}	2
b <sub>1</sub>	{1,4,5}	3
b <sub>2</sub>	{2,3}	2
c <sub>1</sub>	{1,2,3,4,5}	5
d <sub>1</sub>	{1,3,4,5}	4
d <sub>2</sub>	{2}	1
e <sub>1</sub>	{1,2}	2
e <sub>2</sub>	{3,4}	2
e <sub>3</sub>	{5}	1

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

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

**ab** cuboid

Cell	Intersection	TID List	List Size
(a <sub>1</sub> ,b <sub>1</sub> )	{1,2,3} ∩ {1,4,5}	{1}	1
(a <sub>1</sub> ,b <sub>2</sub> )	{1,2,3} ∩ {2,3}	{2,3}	2
(a <sub>2</sub> ,b <sub>1</sub> )	{4,5} ∩ {1,4,5}	{4,5}	2
(a <sub>2</sub> ,b <sub>2</sub> )	{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 Construction (3)

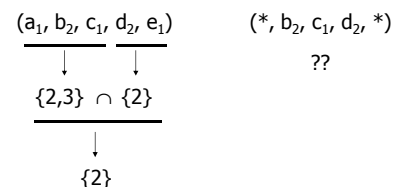
- Using an *ID\_measure* array instead of the original database table

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<sub>1</sub>,b<sub>2</sub>,c<sub>1</sub>,d<sub>2</sub>,e<sub>1</sub>,??)
  - (a<sub>1</sub>,b<sub>2</sub>,c<sub>1</sub>,d<sub>2</sub>,\*,??)
  - (\* ,b<sub>2</sub>,c<sub>1</sub>,d<sub>2</sub> ,\*,??)

## Answering Point Queries

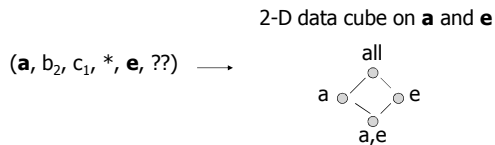




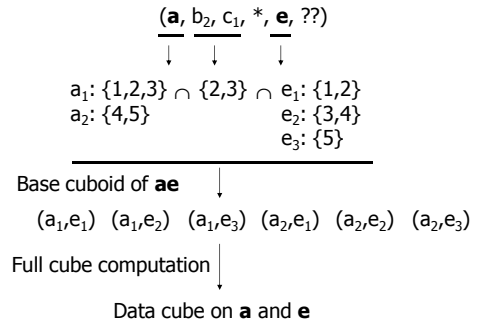
## Query Cube Fragments – Subcube Query

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

◆ Example:



## Answering Subcube Queries



## OLAP Storage Types

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

## A ROLAP Data Store

◆ Summary fact tables

RID	Item	Day	Month	Quarter	Year	Sales
1001	TV	15	10	Q4	2003	250
1002	TV	23	10	Q4	2003	175
...						
5001	TV	<b>all</b>	10	Q4	2003	45,786

## 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