Data Mining:

Concepts and Techniques

(3rd ed.)

— Chapter 5 —

Jianjun Cheng

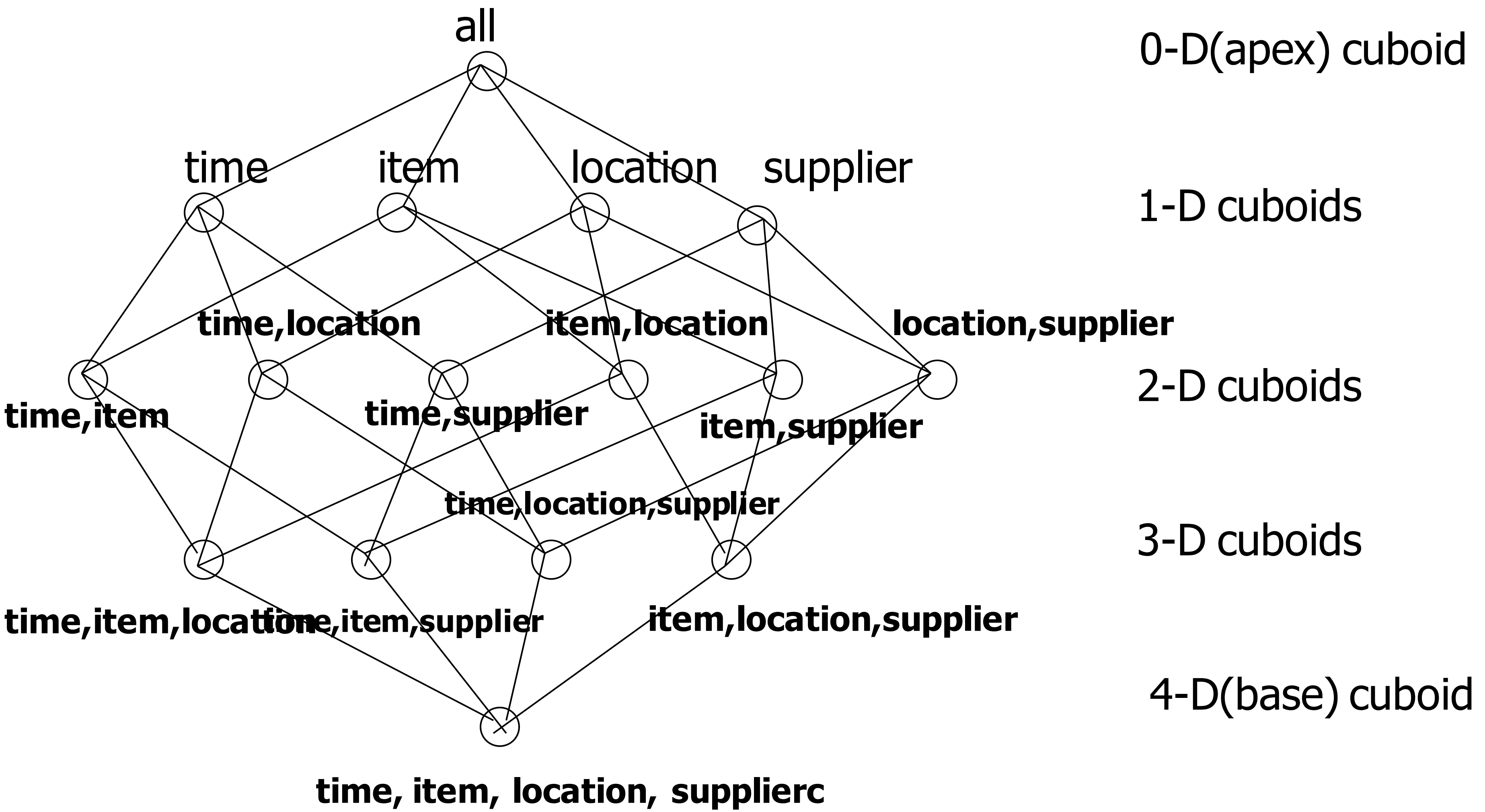
School of Information Science & Engineering

Lanzhou University

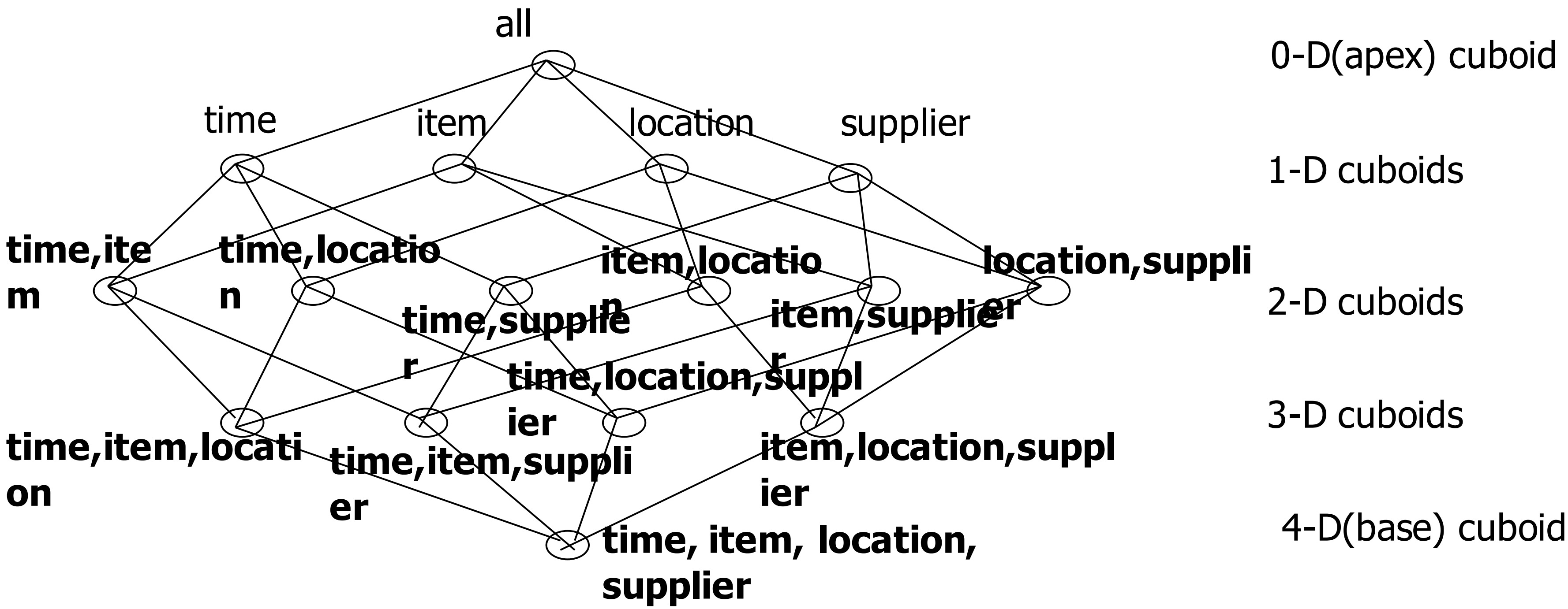
Chapter 5: Data Cube Technology

- Data Cube Computation: Preliminary Concepts
- Data Cube Computation Methods
- Processing Advanced Queries by Exploring Data Cube Technology
- Multidimensional Data Analysis in Cube Space
- Summary

Data Cube: A Lattice of Cuboids



Data Cube: A Lattice of Cuboids



- Base vs. aggregate cells; ancestor vs. descendant cells; parent vs. child cells
  1. (9/15, milk, Urbana, Dairy_land)
  2. (9/15, milk, Urbana, *)
  3. (*, milk, Urbana, *)
  4. (*, milk, Urbana, *)
  5. (*, milk, Chicago, *)
  6. (*, milk, *, *)

Cube Materialization

- Cube、Cuboid、basic cuboid、apex cuboid
- Basic cell、aggregate cell

  A cell in the base cuboid is a base cell, a cell from a nonbase cuboid is an aggregate cell.
- An aggregate cell aggregates over one or more dimensions, where each aggregated dimension is indicated by a "*" in the cell notation.
- Let $a = (a_1 , a_2 , . . . , a_n , measures)$ be a cell from one of the cuboids making up the data cube. We say that a is an m-dimensional cell if exactly m ($m \le n$) values among $\{a_1 , a_2 , . . . , a_n \}$ are not "*".
- If $m = n$, then a is a base cell; otherwise, it is an aggregated cell (i.e., where $m < n$).

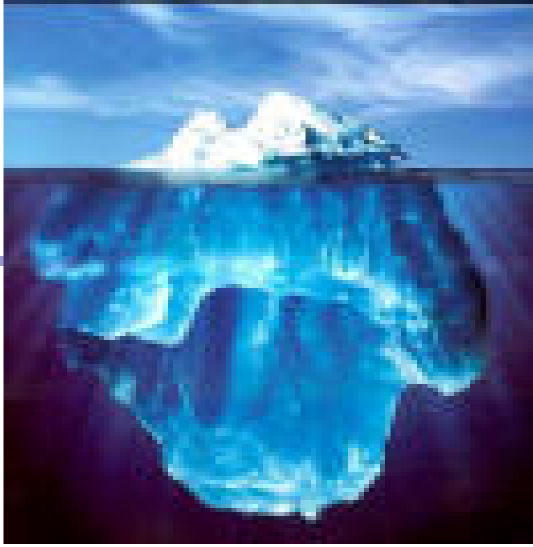
Cube Materialization

- An ancestor-descendant relationship may exist between cells.
- In an n-dimensional data cube, an i-D cell $a = (a_1, a_2, \ldots, a_n, measures_a)$ is an ancestor of a j-D cell $b = (b_1, b_2, \ldots, b_n, measures_b)$, and b is a descendant of a, if and only if

  (1) $i < j$

  (2) for $1 \le m \le n$, $a_m = b_m$ whenever $a_m \neq$ "*".
- In particular, cell a is called a parent of cell b, and b is a child of a, if and only if $j = i+1$ and b is a descendant of a.

Cube Materialization: Full Cube vs. Iceberg Cube



- Full cube vs. iceberg cube

  compute cube sales iceberg as
  select month, city, customer group, count(*)
  from salesInfo
  cube by month, city, customer group
  having count(*) >= min_support

  iceberg condition

- Computing *only* the cuboid cells whose measure satisfies the iceberg condition
- Only a small portion of cells may be "above the water'' in a sparse cube
- Avoid explosive growth: A cube with 100 dimensions
  - 2 base cells: (a1, a2, …., a100), (b1, b2, …, b100)
  - How many aggregate cells if "having count >= 1"?
  - What about "having count >= 2"?

Iceberg Cube, Closed Cube & Cube Shell

- Is iceberg cube good enough?
  - 2 base cells: $\{(a_1, a_2, a_3 \ldots, a_{100}):10, (a_1, a_2, b_3, \ldots, b_{100}):10\}$
  - How many cells will the iceberg cube have if having count(*) >= 10? Hint: A huge but tricky number!
- Close cube:
  - Closed cell c: if there exists no cell d, s.t. d is a descendant of c, and d has the same measure value as c.
  - Closed cube: a cube consisting of only closed cells
  - What is the closed cube of the above base cuboid? Hint: only 3 cells
- Cube Shell
  - Precompute only the cuboids involving a small # of dimensions, e.g., 3

    For $(A_1, A_2, \ldots A_{10})$, how many combinations to compute?
  - More dimension combinations will need to be computed on the fly

Roadmap for Efficient Computation

- General cube computation heuristics (Agarwal et al.'96)
- Computing full/iceberg cubes: 3 methodologies
  - Bottom-Up: Multi-Way array aggregation (Zhao, Deshpande & Naughton, SIGMOD'97)
  - Top-down:
    - BUC (Beyer & Ramarkrishnan, SIGMOD'99)
    - H-cubing technique (Han, Pei, Dong & Wang: SIGMOD'01)
  - Integrating Top-Down and Bottom-Up:
    - Star-cubing algorithm (Xin, Han, Li & Wah: VLDB'03)
- High-dimensional OLAP: A Minimal Cubing Approach (Li, et al. VLDB'04)
- Computing alternative kinds of cubes:
  - Partial cube, closed cube, approximate cube, etc.

Preliminary Tricks (Agarwal et al. VLDB'96)

- Sorting, hashing, and grouping operations are applied to the dimension attributes in order to reorder and cluster related tuples
  - **Share-sorts:** sharing sorting costs cross multiple cuboids when sort-based method is used
  - **Share-partitions:** sharing the partitioning cost across multiple cuboids when hash-based algorithms are used
- Aggregates may be computed from previously computed aggregates, rather than from the base fact table
  - **Cache-results:** caching results of a cuboid from which other cuboids are computed to reduce disk I/Os
  - **Amortize-scans:** computing as many as possible cuboids at the same time to amortize disk reads
  - **Smallest-child:** computing a cuboid from the smallest, previously computed cuboid
  - **Apriori pruning method:** If a given cell does not satisfy minimum support, then no descendant of the cell will satisfy minimum support either.

Chapter 5: Data Cube Technology

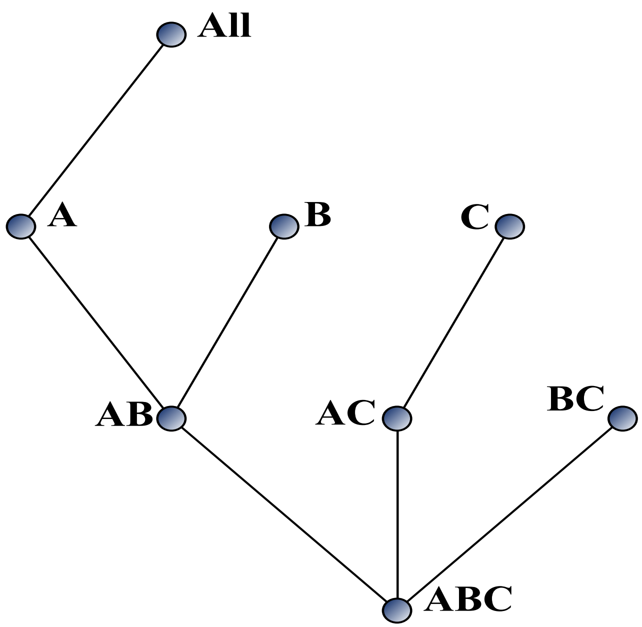
- Data Cube Computation: Preliminary Concepts
- Data Cube Computation Methods
- Processing Advanced Queries by Exploring Data Cube Technology
- Multidimensional Data Analysis in Cube Space
- Summary

Data Cube Computation Methods

- Multi-Way Array Aggregation
- BUC
- Star-Cubing
- High-Dimensional OLAP

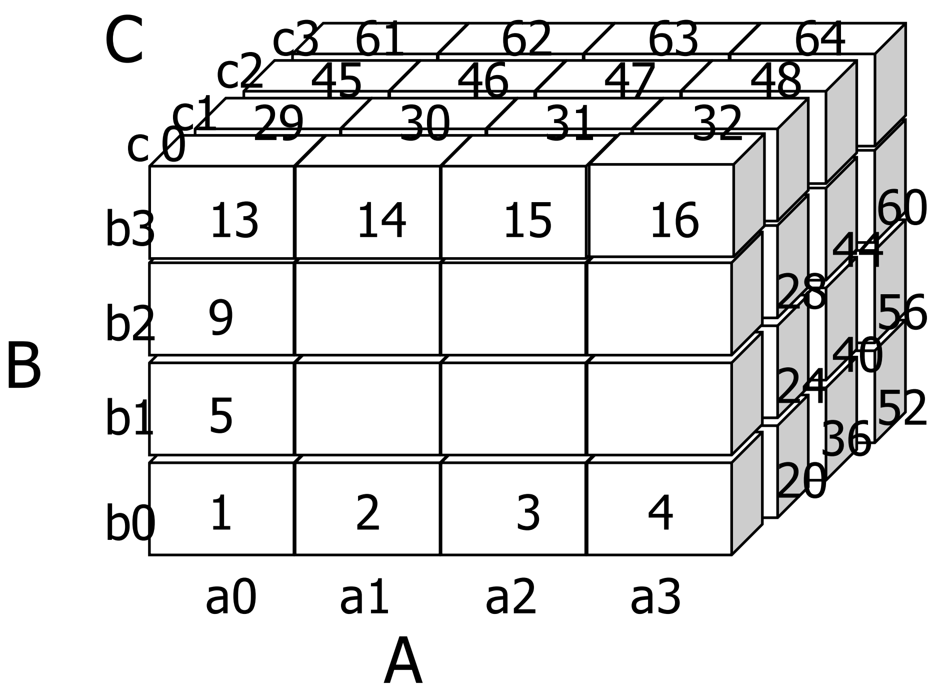
Multi-Way Array Aggregation

- Array-based "bottom-up" algorithm
- Using multi-dimensional chunks
- No direct tuple comparisons
- Simultaneous aggregation on multiple dimensions
- Intermediate aggregate values are re-used for computing ancestor cuboids
- Cannot do *Apriori* pruning: No iceberg optimization



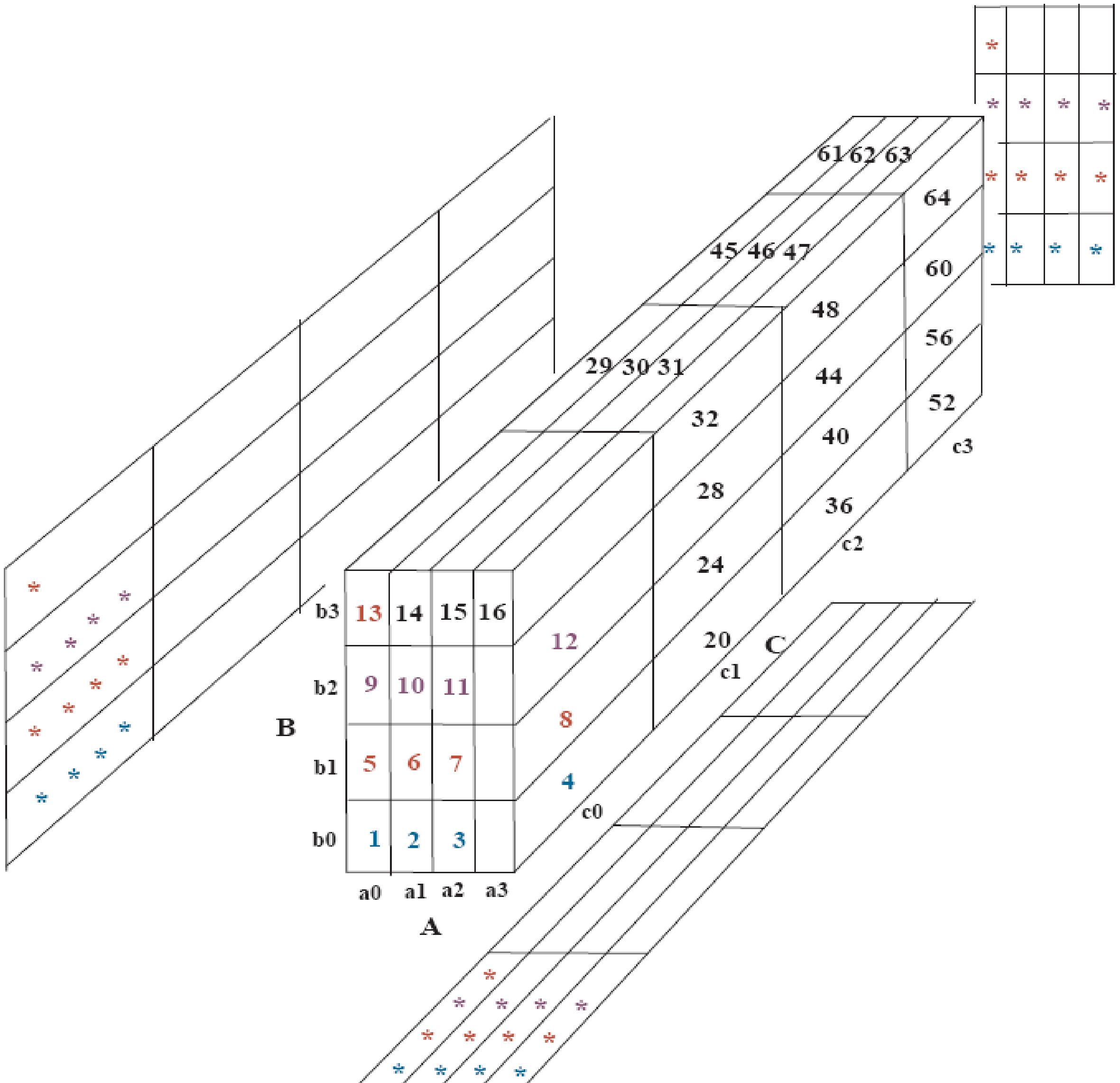
Multi-way Array Aggregation for Cube Computation (MOLAP)

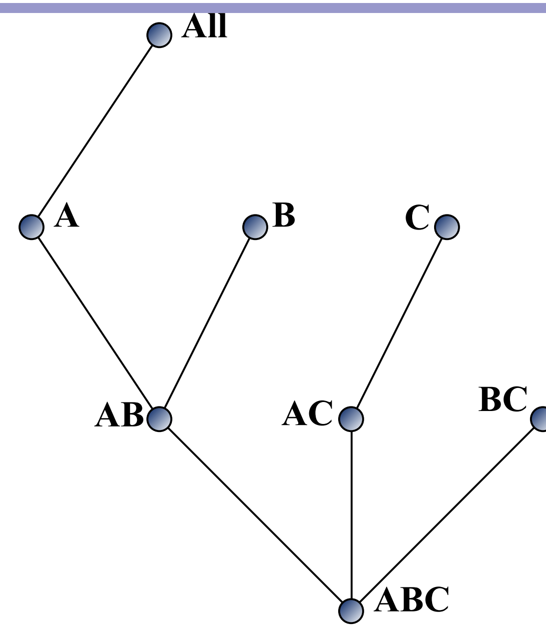
- Partition arrays into chunks (a small subcube which fits in memory).
- Compressed sparse array addressing: (chunk_id, offset)
- Compute aggregates in "multiway" by visiting cube cells in the order which minimizes the # of times to visit each cell, and reduces memory access and storage cost.



What is the best traversing order to do multi-way aggregation?

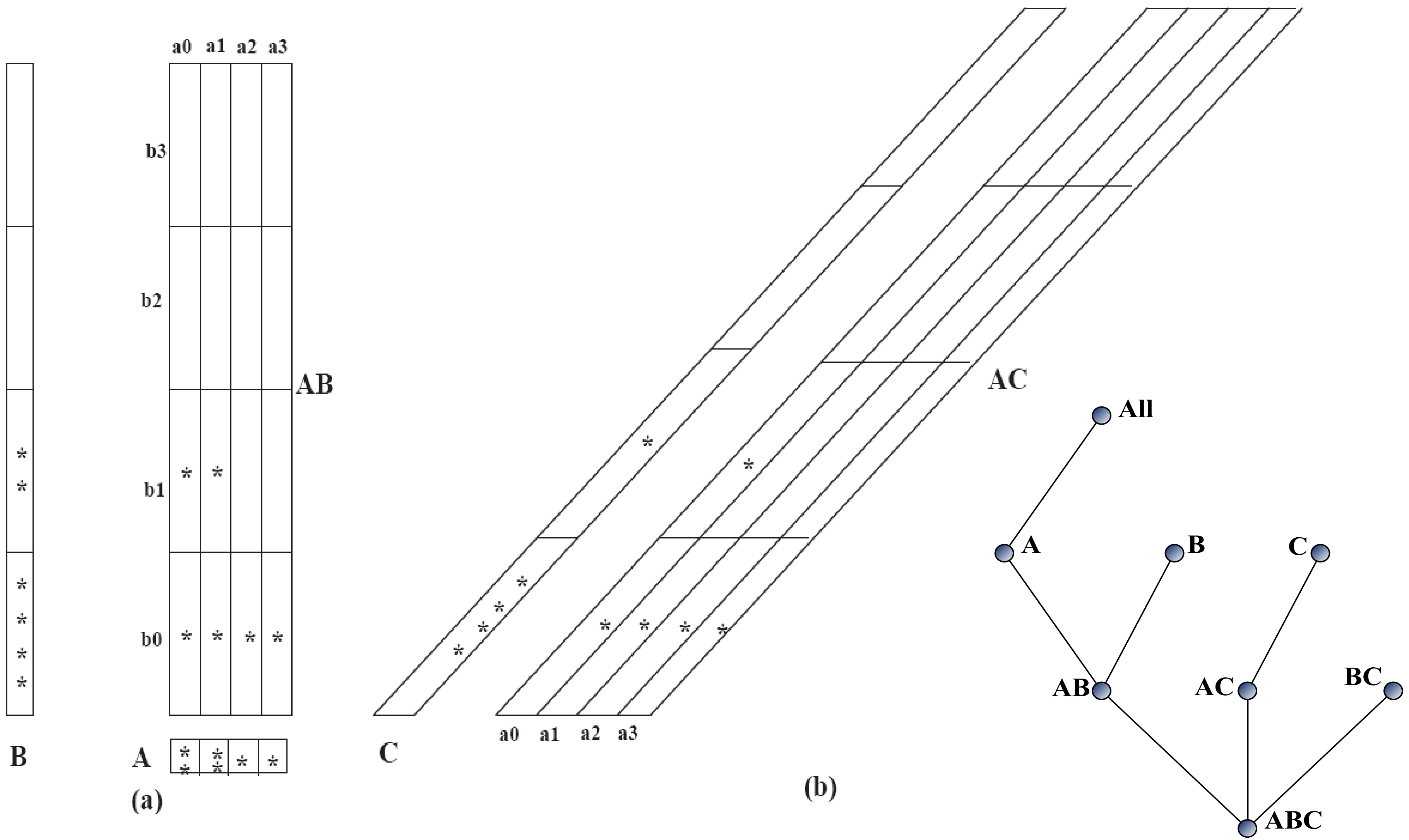
Multi-way Array Aggregation for Cube Computation (3-D to 2-D)



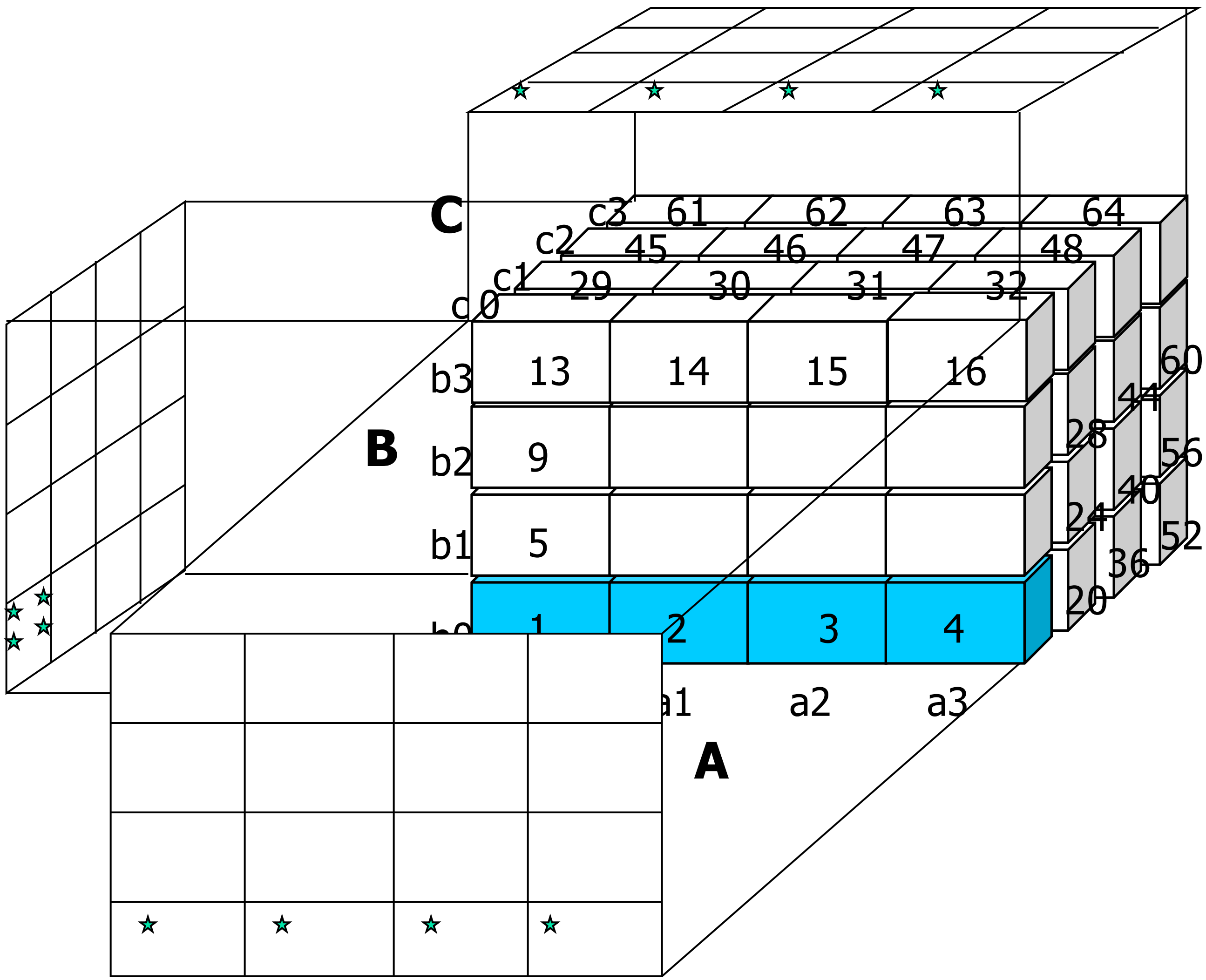


- The best order is the one that minimizes the memory requirement and reduced I/Os

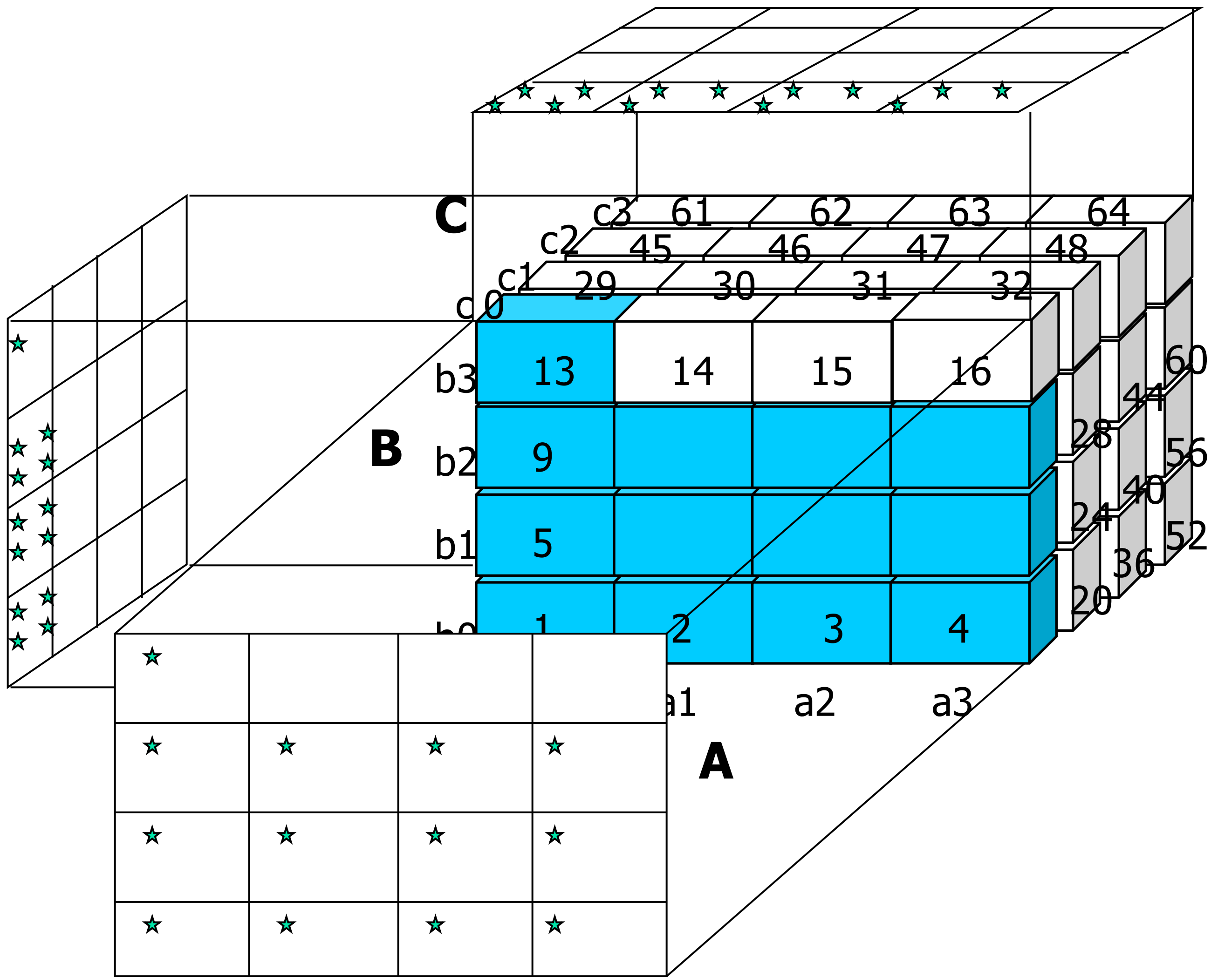
Multi-way Array Aggregation for Cube Computation (2-D to 1-D)



Multi-way Array Aggregation for Cube Computation



Multi-way Array Aggregation for Cube Computation



Multi-Way Array Aggregation for Cube Computation (Method Summary)

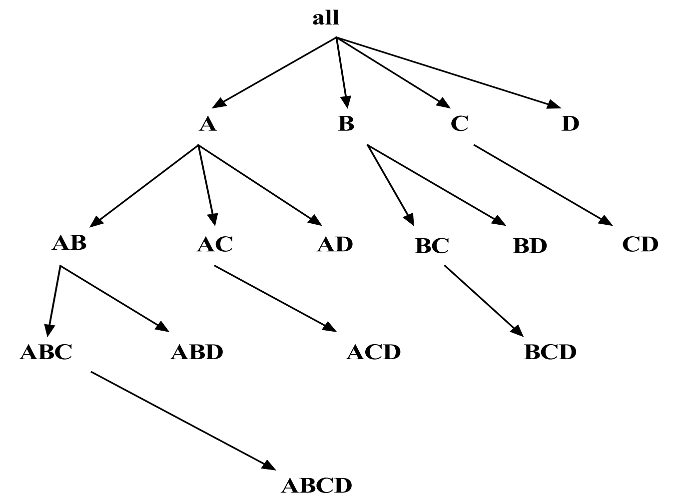
- Method: the planes should be sorted and computed according to their size in ascending order
  - Idea: keep the smallest plane in the main memory, fetch and compute only one chunk at a time for the largest plane
- Limitation of the method: computing well only for a small number of dimensions
  - If there are a large number of dimensions, "top-down" computation and iceberg cube computation methods can be explored

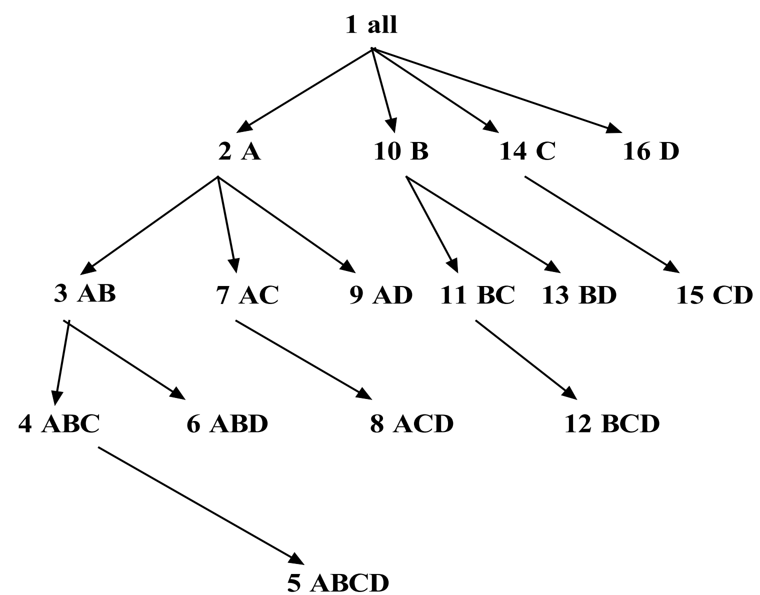
Data Cube Computation Methods

- Multi-Way Array Aggregation
- BUC
- Star-Cubing
- High-Dimensional OLAP

Bottom-Up Computation (BUC)

- BUC (Beyer & Ramakrishnan, SIGMOD'99)
- Bottom-up cube computation
  (Note: top-down in our view!)
- Divides dimensions into partitions and facilitates iceberg pruning
  - If a partition does not satisfy *min_sup*, its descendants can be pruned
  - If *minsup* = 1 ⇨ compute full CUBE!
- No simultaneous aggregation





BUC: Partitioning

- Usually, entire data set can't fit in main memory
- Sort *distinct* values
  - partition into blocks that can fit in main memory
- Continue processing
- Optimizations
  - Partitioning
    - External Sorting, Hashing, Counting Sort
  - Ordering dimensions to encourage pruning
    - Cardinality, Skew, Correlation
  - Collapsing duplicates
    - Can't do holistic aggregates anymore!

BUC: Partitioning

Algorithm: BUC. Algorithm for the computation of sparse and iceberg cubes.

Input:

- *input*: the relation to aggregate;
- *dim*: the starting dimension for this iteration.

Globals:

- constant *numDims*: the total number of dimensions;
- constant *cardinality[numDims]*: the cardinality of each dimension;
- constant *min_sup*: the minimum number of tuples in a partition for it to be output;
- *outputRec*: the current output record;
- *dataCount[numDims]*: stores the size of each partition. *dataCount[i]* is a list of integers of size *cardinality[i]*.

Output: Recursively output the iceberg cube cells satisfying the minimum support.

BUC: Partitioning

Method:

```
(1)  Aggregate(input); // Scan input to compute measure, e.g., count. Place result in outputRec.
(2)  if input.count() == 1 then // Optimization
         WriteDescendants(input[0], dim); return;
     endif
(3)  write outputRec;
(4)  for (d = dim; d < numDims; d++) do //Partition each dimension
(5)      C = cardinality[d];
(6)      Partition(input, d, C, dataCount[d]); //create C partitions of data for dimension d
(7)      k = 0;
(8)      for (i = 0; i < C; i++) do // for each partition (each value of dimension d)
(9)          c = dataCount[d][i];
(10)         if c >= min_sup then // test the iceberg condition
(11)             outputRec.dim[d] = input[k].dim[d];
(12)             BUC(input[k..k + c − 1], d + 1); // aggregate on next dimension
(13)         endif
(14)         k +=c;
(15)     endfor
(16)     outputRec.dim[d] = all;
(17) endfor
```

BUC: Example

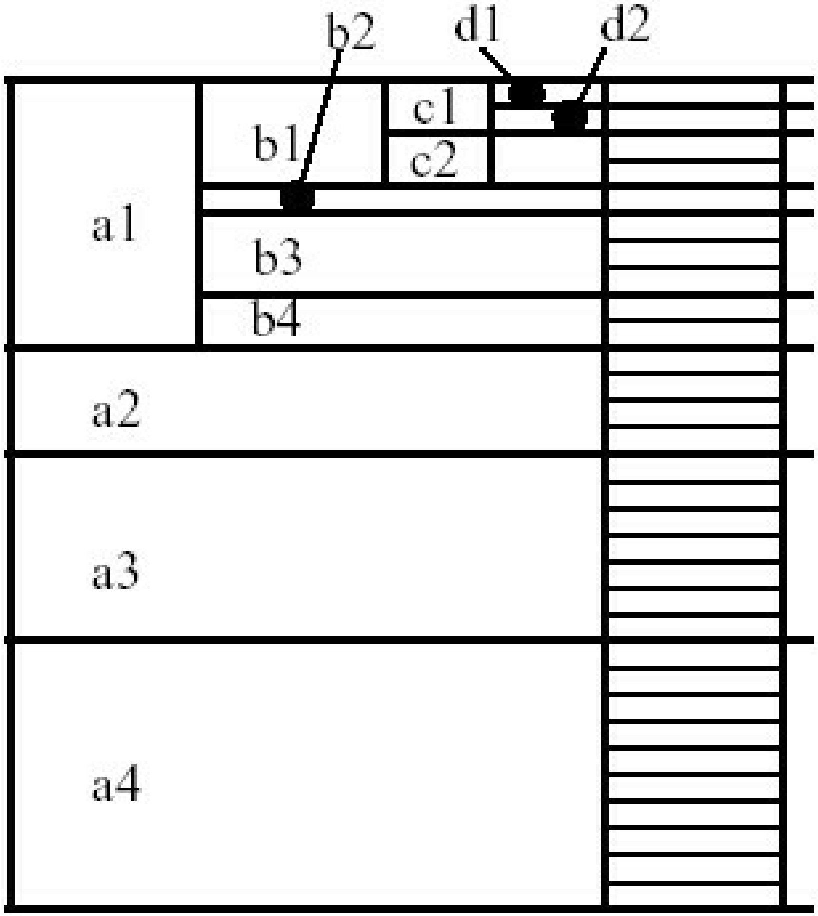
compute cube iceberg cube as

select A, B, C, D, count(*)

from R

cube by A, B, C, D

having count(*) >= 3

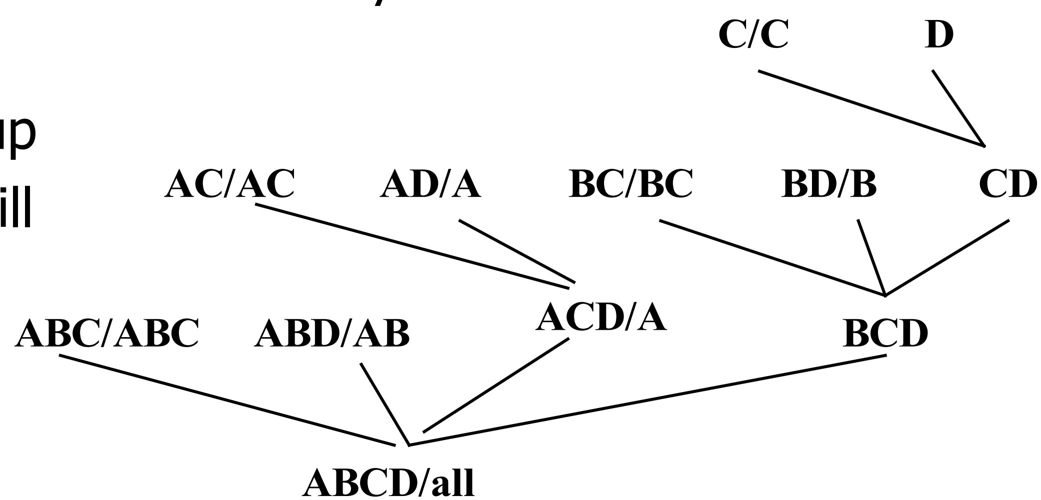


Data Cube Computation Methods

- Multi-Way Array Aggregation
- BUC
- Star-Cubing
- High-Dimensional OLAP

Star-Cubing: An Integrating Method

- D. Xin, J. Han, X. Li, B. W. Wah, Star-Cubing: Computing Iceberg Cubes by Top-Down and Bottom-Up Integration, VLDB'03
- *Explore shared dimensions*
  - E.g., dimension A is the shared dimension of ACD and AD
  - ABD/AB means cuboid ABD has shared dimensions AB
- Allows for shared computations
  - e.g., cuboid AB is computed simultaneously as ABD
- Aggregate in a top-down manner but with the bottom-up sub-layer underneath which will allow Apriori pruning
- Shared dimensions grow in bottom-up fashion

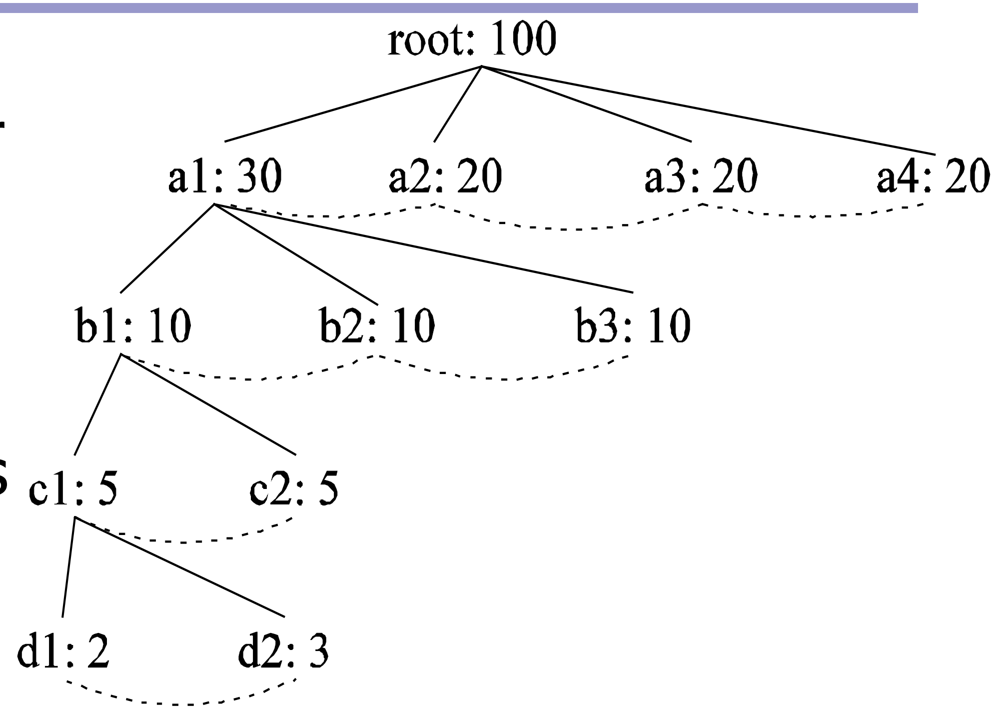


Iceberg Pruning in Shared Dimensions

- Anti-monotonic property of shared dimensions
  - If the measure is *anti-monotonic*, and if the aggregate value on a shared dimension does not satisfy the *iceberg condition*, then all the cells extended from this shared dimension cannot satisfy the condition either
- Intuition: if we can compute the shared dimensions before the actual cuboid, we can use them to do `Apriori` pruning
- Problem: how to prune while still aggregate simultaneously on multiple dimensions?

Cell Trees

- Use a tree structure similar to H-tree to represent cuboids
- Collapses common prefixes to save memory
- Keep count at node
- Traverse the tree to retrieve a particular tuple



Star Attributes and Star Nodes

- Intuition: If a single-dimensional aggregate on an attribute value *p* does not satisfy the iceberg condition, it is useless to distinguish them during the iceberg computation
  - E.g., b_2, b_3, b_4, c_1, c_2, c_4, d_1, d_2, d_3
- Solution: Replace such attributes by a *. Such attributes are *star attributes*, and the corresponding nodes in the cell tree are *star nodes*

A	B	C	D	Count
a1	b1	c1	d1	1
a1	b1	c4	d3	1
a1	b2	c2	d2	1
a2	b3	c3	d4	1
a2	b4	c3	d4	1

Example: Star Reduction

- Suppose minsup = 2
- Perform one-dimensional aggregation. Replace attribute values whose count < 2 with *. And collapse all *'s together
- Resulting table has all such attributes replaced with the star-attribute
- With regards to the iceberg computation, this new table is a *lossless compression* of the original table

A	B	C	D	Count
a1	b1	*	*	1
a1	b1	*	*	1
a1	*	*	*	1
a2	*	c3	d4	1
a2	*	c3	d4	1

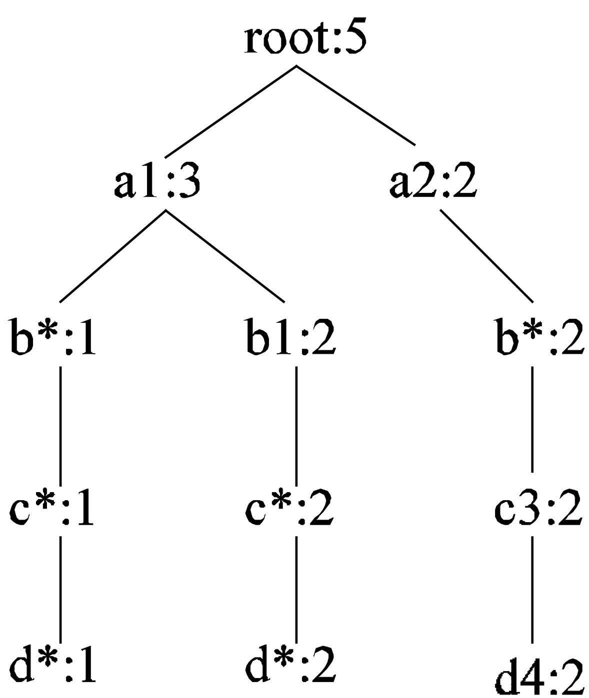
↓

A	B	C	D	Count
a1	b1	*	*	2
a1	*	*	*	1
a2	*	c3	d4	2

Star Tree

- Given the new compressed table, it is possible to construct the corresponding cell tree—called star tree
- Keep a star table at the side for easy lookup of star attributes
- The star tree is a *lossless compression* of the original cell tree

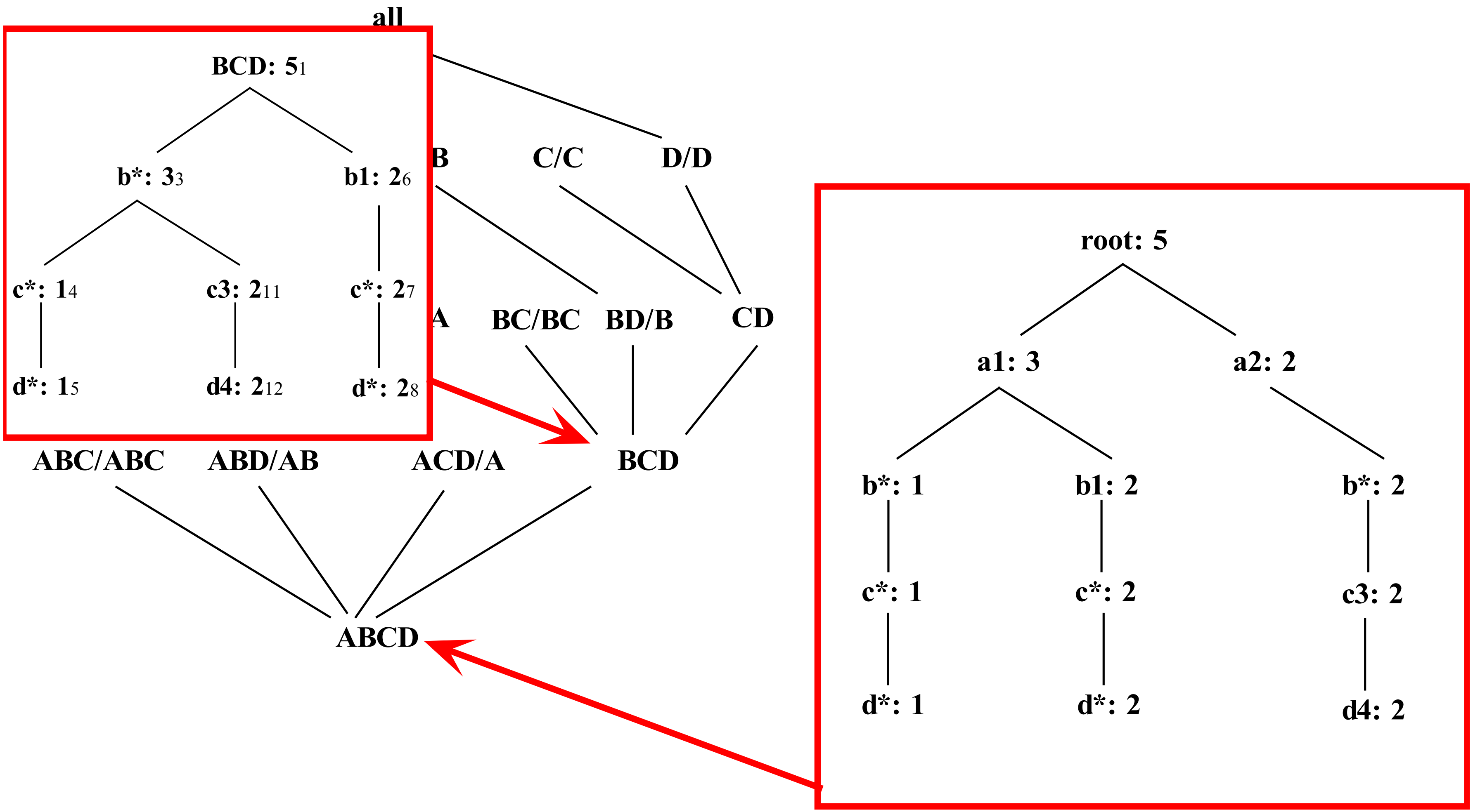
A	B	C	D	Count
a1	b1	*	*	2
a1	*	*	*	1
a2	*	c3	d4	2



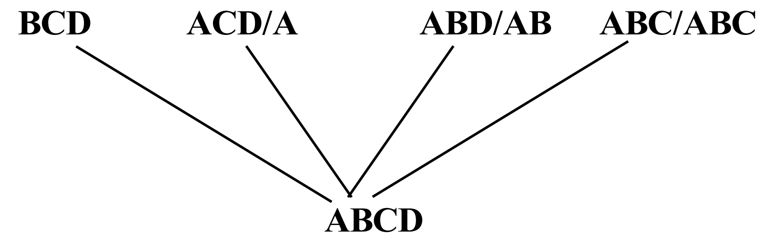
Star Table

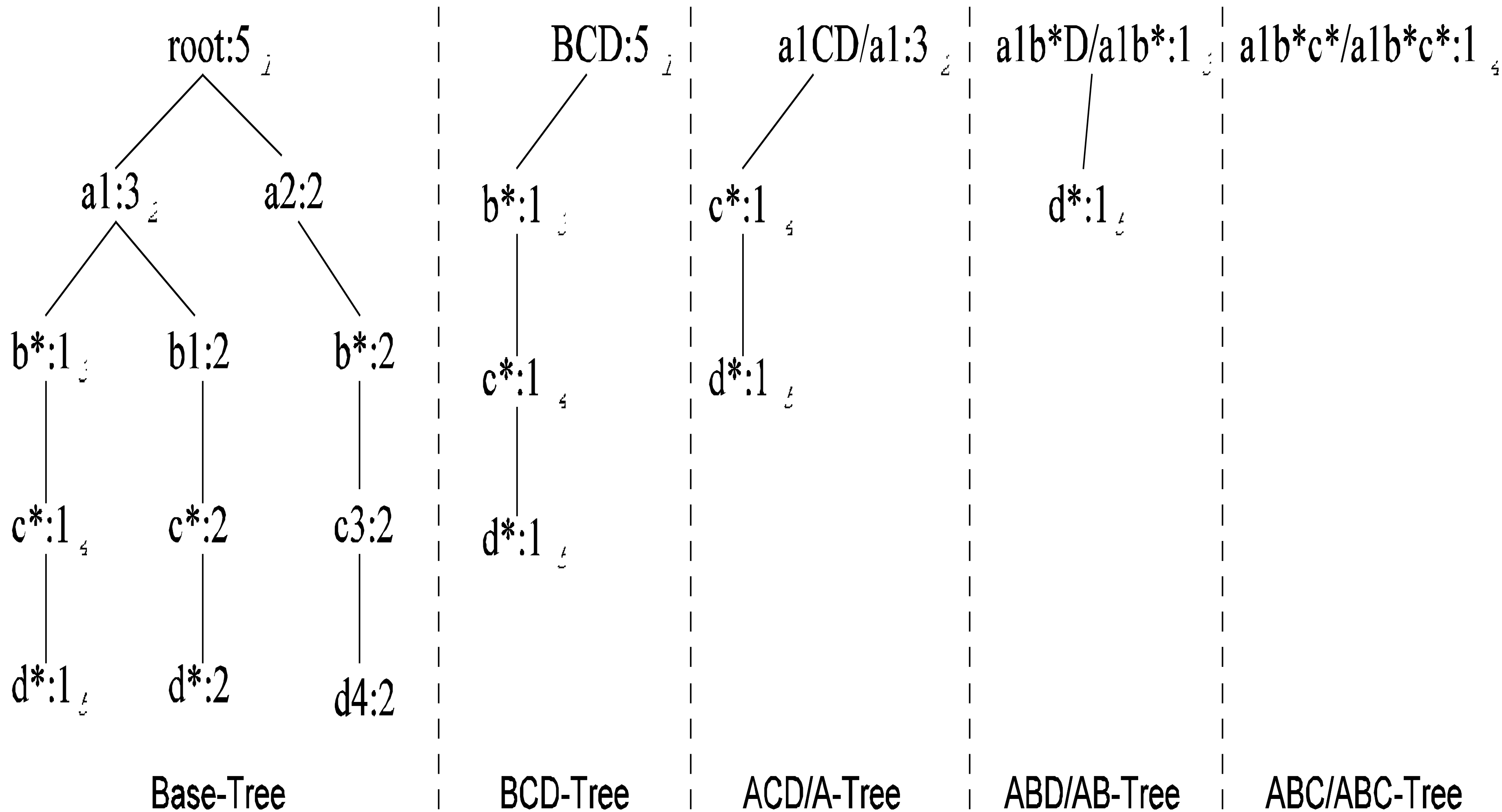
b2 → *
b3 → *
b4 → *
c1 → *
c2 → *
d1 → *
…

Star-Cubing Algorithm—DFS on Lattice Tree

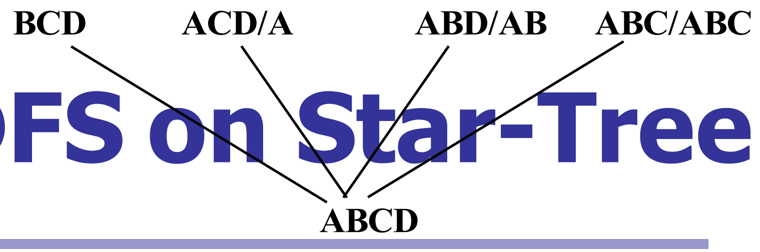


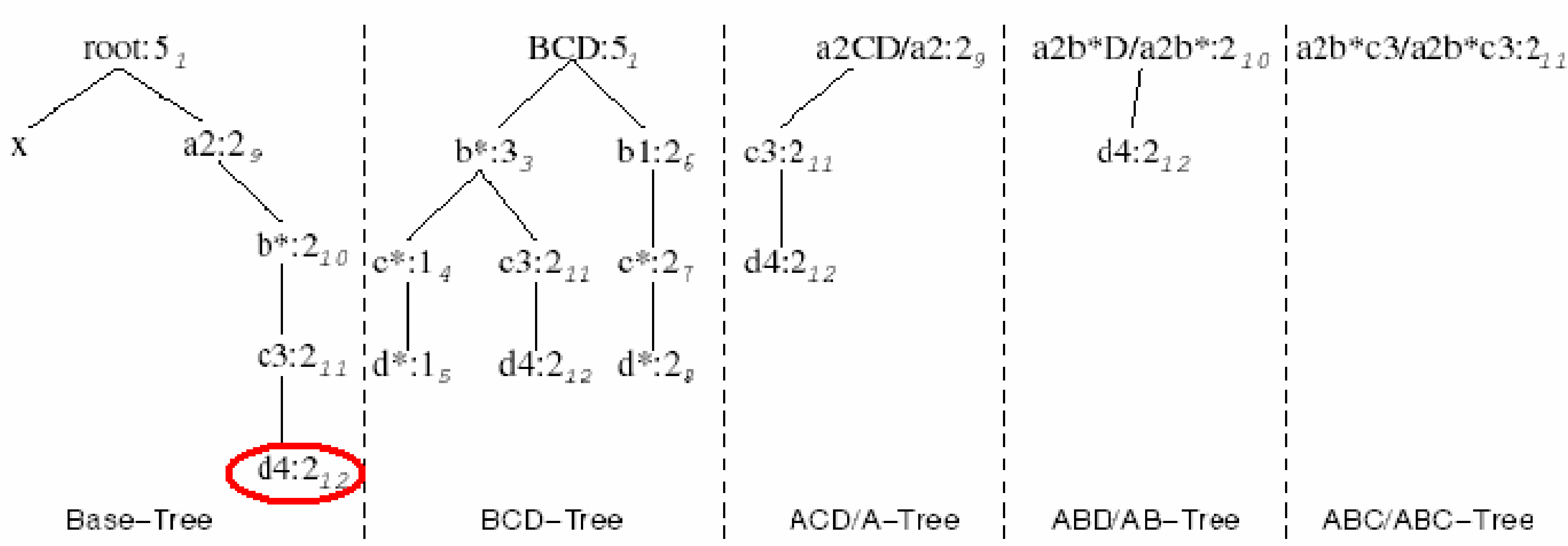
Multi-Way Aggregation



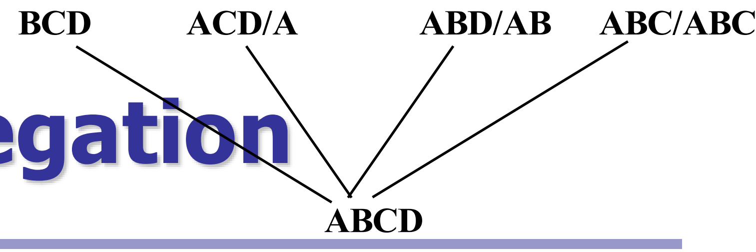


Star-Cubing Algorithm—DFS on Star-Tree





Multi-Way Star-Tree Aggregation



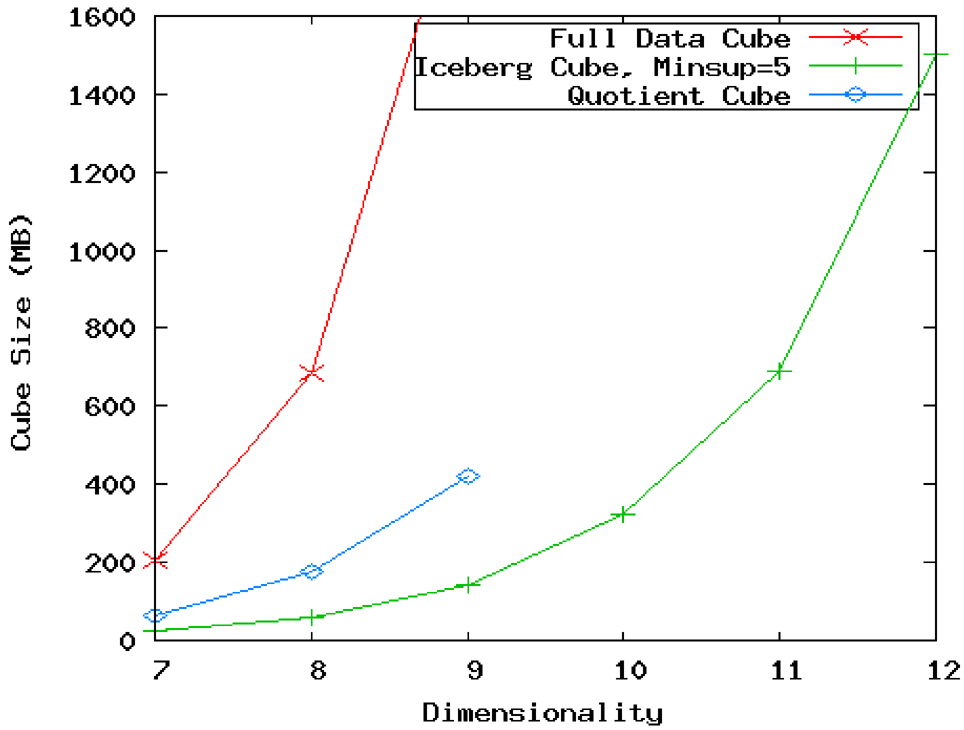
- Start depth-first search at the root of the base star tree
- At each new node in the DFS, create corresponding star tree that are descendants of the current tree according to the integrated traversal ordering
  - E.g., in the base tree, when DFS reaches `a1`, the ACD/A tree is created
  - When DFS reaches `b*`, the ABD/AD tree is created
- The counts in the base tree are carried over to the new trees
- When DFS reaches a leaf node (e.g., `d*`), start backtracking
- On every backtracking branch, the count in the corresponding trees are output, the tree is destroyed, and the node in the base tree is destroyed
- Example
  - When traversing from `d*` back to `c*`, the `a1b*c*/a1b*c*` tree is output and destroyed
  - When traversing from `c*` back to `b*`, the `a1b*D/a1b*` tree is output and destroyed
  - When at `b*`, jump to `b1` and repeat similar process

Data Cube Computation Methods

- Multi-Way Array Aggregation
- BUC
- Star-Cubing
- High-Dimensional OLAP

The Curse of Dimensionality

- None of the previous cubing method can handle high dimensionality!
- A database of 600k tuples. Each dimension has cardinality of 100 and *zipf* of 2.



Motivation of High-D OLAP

- X. Li, J. Han, and H. Gonzalez, High-Dimensional OLAP: A Minimal Cubing Approach, VLDB'04
- Challenge to current cubing methods:
  - The "curse of dimensionality" problem
  - Iceberg cube and compressed cubes: only delay the inevitable explosion
  - Full materialization: still significant overhead in accessing results on disk
- High-D OLAP is needed in applications
  - Science and engineering analysis
  - Bio-data analysis: thousands of genes
  - Statistical surveys: hundreds of variables

Fast High-D OLAP with Minimal Cubing

- Observation: OLAP occurs only on a small subset of dimensions at a time
- Semi-Online Computational Model
  1. Partition the set of dimensions into **shell fragments**
  2. Compute data cubes for each shell fragment while retaining **inverted indices** or **value-list indices**
  3. Given the pre-computed **fragment cubes**, dynamically compute cube cells of the high-dimensional data cube *online*

Properties of Proposed Method

- Partitions the data vertically
- Reduces high-dimensional cube into a set of lower dimensional cubes
- Online re-construction of original high-dimensional space
- Lossless reduction
- Offers tradeoffs between the amount of pre-processing and the speed of online computation

Example Computation

- Let the cube aggregation function be `count`

tid	**A**	**B**	**C**	**D**	**E**
1	a1	b1	c1	d1	e1
2	a1	b2	c1	d2	e1
3	a1	b2	c1	d1	e2
4	a2	b1	c1	d1	e2
5	a2	b1	c1	d1	e3

- Divide the 5 dimensions into 2 shell fragments:
  - (A, B, C) and (D, E)

1-D Inverted Indices

- Build traditional invert index or RID list

Attribute Value	TID List	List Size
a1	1 2 3	3
a2	4 5	2
b1	1 4 5	3
b2	2 3	2
c1	1 2 3 4 5	5
d1	1 3 4 5	4
d2	2	1
e1	1 2	2
e2	3 4	2
e3	5	1

Shell Fragment Cubes: Ideas

- Generalize the 1-D inverted indices to multi-dimensional ones in the data cube sense
- Compute all cuboids for data cubes ABC and DE while retaining the inverted indices
- For example, shell fragment cube ABC contains 7 cuboids:
  - A, B, C
  - AB, AC, BC
  - ABC
- This completes the offline computation stage

Cell	Intersection	TID List	List Size
a1 b1	1 2 3 $\cap$ 1 4 5	1	1
a1 b2	1 2 3 $\cap$ 2 3	2 3	2
a2 b1	4 5 $\cap$ 1 4 5	4 5	2
a2 b2	4 5 $\cap$ 2 3	$\otimes$	0

Shell Fragment Cubes: Ideas

Cuboid A、B、C、D、E		
Cell	TID List	|*|
a1	{1,2,3}	3
a2	{4,5}	2
b1	{1,4,5}	3
b2	{2,3}	2
c1	{1,2,3,4,5}	5
d1	{1,3,4,5}	4
d2	{2}	1
e1	{1,2}	2
e2	{3,4}	2
e3	{5}	1

Shell Fragment Cubes: Ideas

Cuboid AB

Cell	Intersection	T List	|AB|
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

Cuboid ABC

Cell	intersection	T List	|ABC|
a_1 b_1 c_1	{1}∩{1,2,3,4,5}	{1}	1
a_1 b_2 c_1	{2,3}∩{1,2,3,4, 5}	{2,3}	2
a_2 b_1 c_1	{4,5}∩{1,2,3,4,5 }	{4,5}	2
a_2 b_2 c_1	{}∩{1,2,3,4,5}	{}	0

Cuboid AC

Cell	Intersection	T List	|AC|
a_1 c_1	{1,2,3}∩{1,2,3,4,5 }	{1,2,3 }	3
a_2 c_1	{4,5}∩{1,2,3,4,5}	{4,5}	2

Cuboid BC

Cell	Intersection	T List	|BC|
b_1 c_1	{1,4,5}∩{1,2,3,4,5 }	{1,4,5 }	3
b_2 c_1	{2,3}∩{1,2,3,4,5}	{2,3}	2

Cuboid DE

Cell	Intersection	T List	|DE|
d_1 e_1	{1,3,4,5}∩{1,2}	{1}	1
d_1 e_2	{1,3,4,5}∩{3,4}	{3,4}	2
d_1 e_3	{1,3,4,5}∩{5}	{5}	1
d_2 e_1	{2}∩{1,2}	{2}	1

Shell Fragment Cubes: Size and Design

- Given a database of T tuples, D dimensions, and shell fragment size F, the fragment cubes' space requirement is

$$O\left(T\left\lceil\frac{D}{F}\right\rceil(2^F-1)\right)$$

  - For F < 5, the growth is sub-linear
- Shell fragments do not have to be disjoint
- Fragment groupings can be arbitrary to allow for maximum online performance
  - Known common combinations (e.g.,<city, state>) should be grouped together.
- Shell fragment sizes can be adjusted for optimal balance between offline and online computation

ID_Measure Table

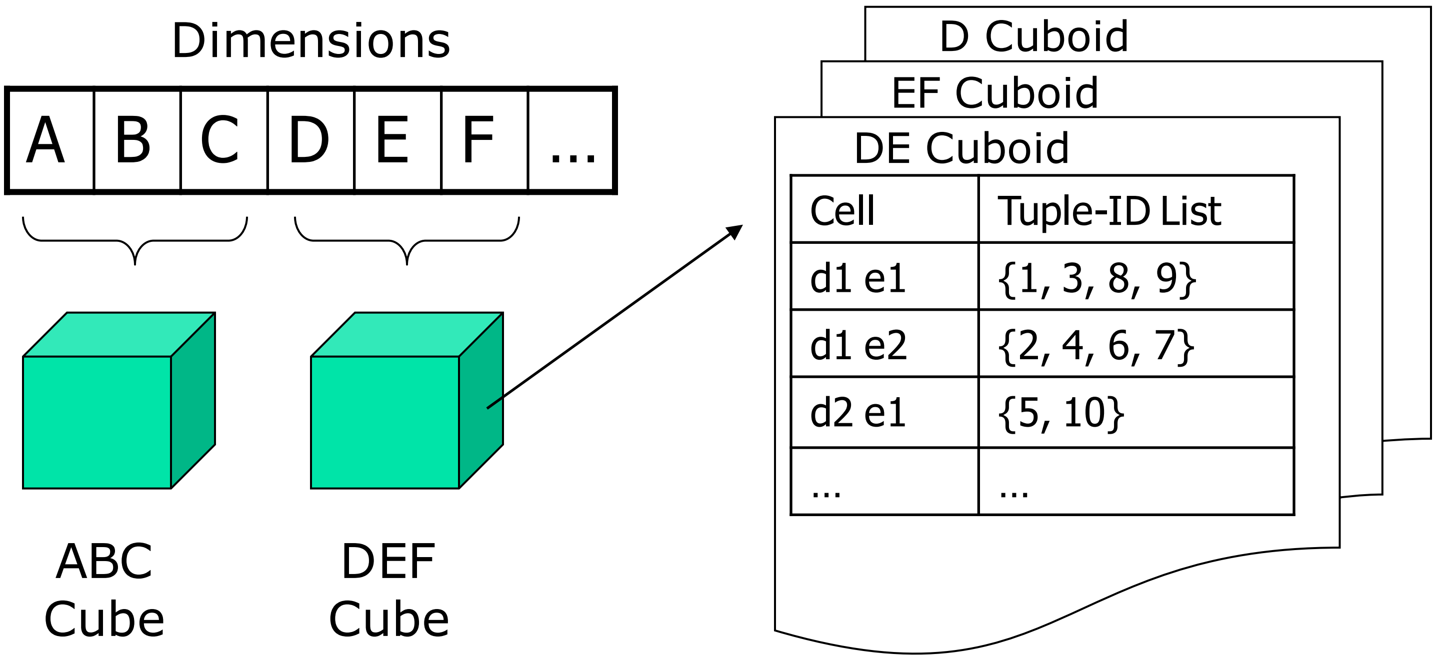
- If measures other than count are present, store in *ID_measure* table separate from the shell fragments

tid	count	sum
1	5	70
2	3	10
3	8	20
4	5	40
5	2	30

The Frag-Shells Algorithm

1. Partition set of dimension ($A_1,\ldots,A_n$) into a set of k fragments ($P_1,\ldots,P_k$).
2. Scan base table once and do the following
3. insert <tid, measure> into ID_measure table.
4. for each attribute value a_i of each dimension A_i
5. build inverted index entry <a_i, tidlist>
6. For each fragment partition P_i
7. build local fragment cube S_i by intersecting tid-lists in bottom-up fashion.

Frag-Shells (2)



Cell	Tuple-ID List
d1 e1	{1, 3, 8, 9}
d1 e2	{2, 4, 6, 7}
d2 e1	{5, 10}
...	...

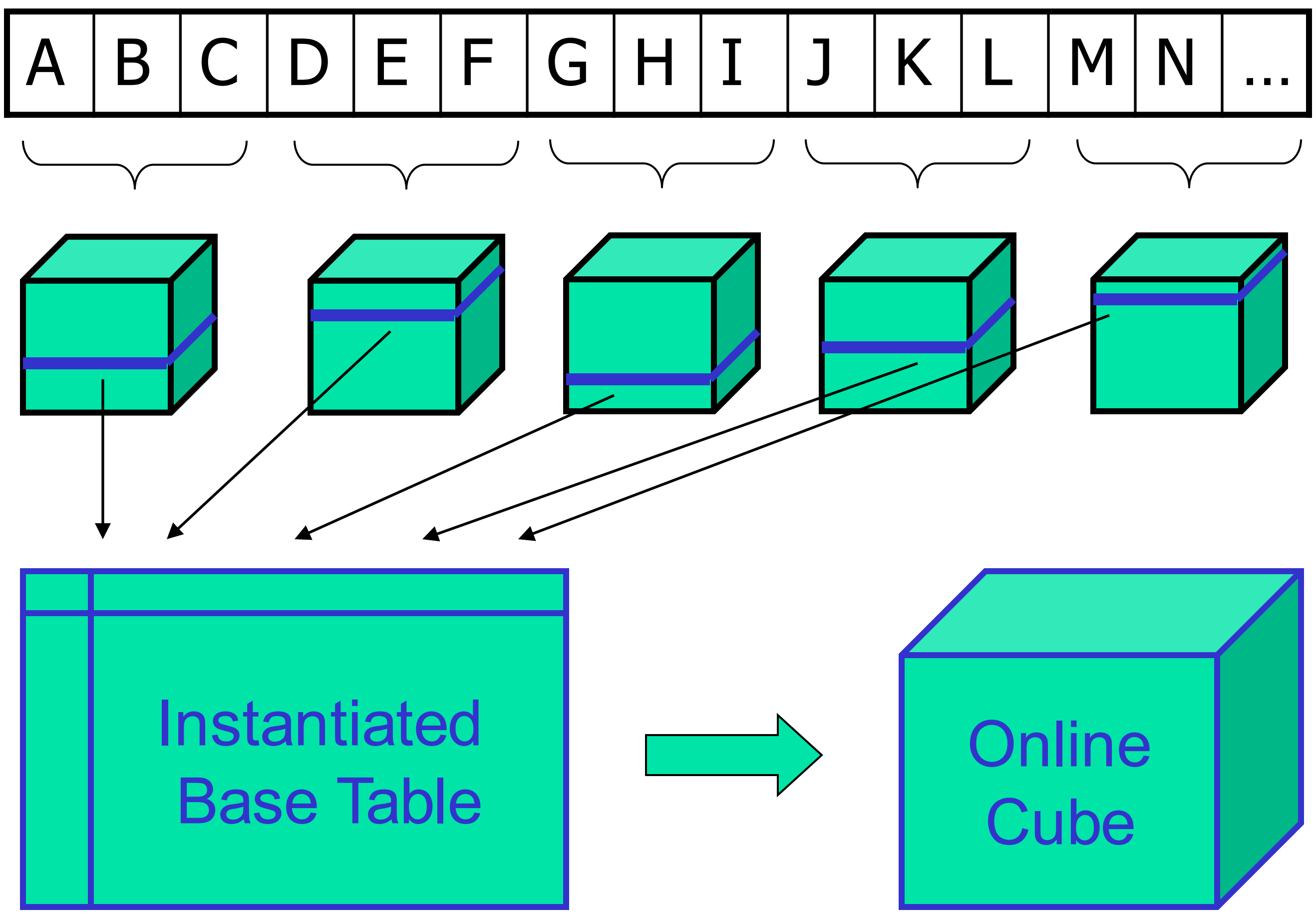
Online Query Computation: Query

- A query has the general form $<a_1,a_2,\ldots,a_n:M?>$
- Each a_i has 3 possible values
  1. Instantiated value
  2. Aggregate * function
  3. Inquire ? function
- For example, $<3,?,?,*,1:count?>$ returns a 2-D data cube.

Online Query Computation: Method

- Given the fragment cubes, process a query as follows
  1. Divide the query into fragment, same as the shell
  2. Fetch the corresponding TID list for each fragment from the fragment cube
  3. Intersect the TID lists from each fragment to construct **instantiated base table**
  4. Compute the data cube using the base table with any cubing algorithm

Online Query Computation: Sketch



Point Query

Query **$<a_2, b_1, c_1, d_1, *: count()?>$**

- Query is broken down into two subqueries based on the precomputed fragments:

  $<a_2, b_1, c_1, *, *>$, $<*, *, *, d_1, *>$

- The best-fit precomputed fragment for $<a_2, b_1, c_1, *, *>$: **ABC**
- The best-fit precomputed fragment for $<*, *, *, d_1, *>$: **D**
- The TID lists for the two subqueries: **{4,5} {1,3,4,5}**
- The intersection is: **{4,5}**, so the answer is: **count()=2**

Subcube Query

Query **<a_2, b_1, ?, *, ?: count()?>**

- Query is broken down into 3 best-fit fragments:

  AB, C, E

- The TID lists for them:

  (a_2, b_2): {4,5}、 (c_1): {1,2,3,4,5}、

  {(e_1: {1,2}), (e_2: {3,4}), (e_3: {5})}

- The intersections of them get a cuboid with two tuples

  {(c_1, e_2): {4}, (c_1, e_3): {5}}

Computing Cubes with Non-Antimonotonic Iceberg Conditions

- J. Han, J. Pei, G. Dong, K. Wang. Efficient Computation of Iceberg Cubes With Complex Measures. SIGMOD'01
- Most cubing algorithms cannot compute cubes with non-antimonotonic iceberg conditions efficiently
- Example

  CREATE CUBE Sales_Iceberg AS

  SELECT month, city, cust_grp, AVG(price), COUNT(*)

  FROM Sales_Infor

  CUBEBY month, city, cust_grp

  HAVING AVG(price) >= 800 AND COUNT(*) >= 50

- How to push constraint into the iceberg cube computation?

Non-Anti-Monotonic Iceberg Condition

- Anti-monotonic: if a process fails a condition, continue processing will still fail
- The cubing query with avg is non-anti-monotonic!
  - (Mar, *, *, 600, 1800) fails the HAVING clause
  - (Mar, *, Bus, 1300, 360) passes the clause

Month	City	Cust_grp	Prod	Cost	Price
Jan	Tor	Edu	Printer	500	485
Jan	Tor	Hld	TV	800	1200
Jan	Tor	Edu	Camera	1160	1280
Feb	Mon	Bus	Laptop	1500	2500
Mar	Van	Edu	HD	540	520
…	…	…	…	…	…

```
CREATE CUBE Sales_Iceberg AS
SELECT month, city, cust_grp,
        AVG(price), COUNT(*)
FROM Sales_Infor
CUBEBY month, city, cust_grp
HAVING AVG(price) >= 800 AND
        COUNT(*) >= 50
```

From Average to Top-k Average

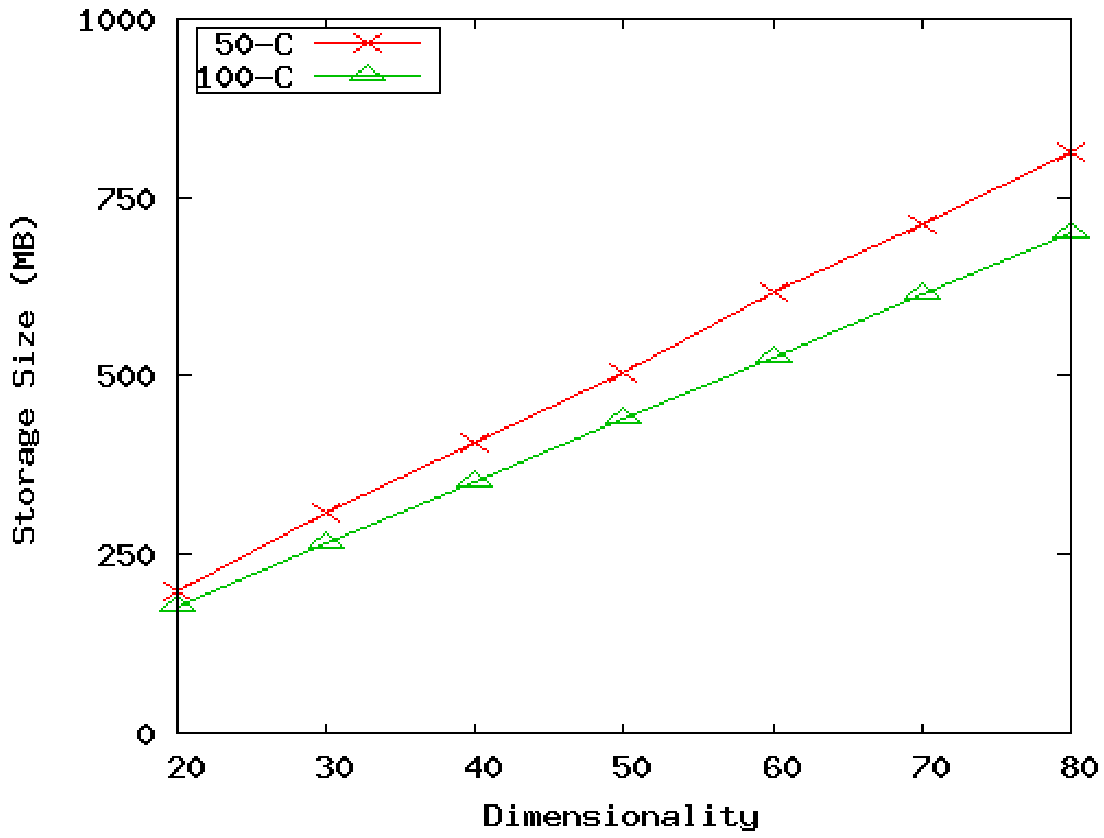
- Let (*, Van, *) cover 1,000 records
  - Avg(price) is the average price of those 1000 sales
  - Avg^{50}(price) is the average price of the top-50 sales (top-50 according to the sales price)
- Top-k average is anti-monotonic
  - The top 50 sales in Van. is with avg(price) <= 800 → the top 50 deals in Van. during Feb. must be with avg(price) <= 800

Month	City	Cust_grp	Prod	Cost	Price
…	…	…	…	…	…

Computing Iceberg Cubes with Other Complex Measures

- Computing other complex measures
  - Key point: find a function which is weaker but ensures certain anti-monotonicity
- Examples
  - Avg() ≤ v: avg^k(c) ≤ v (bottom-k avg)
  - Avg() ≥ v only (no count): max(price) ≥ v
  - Sum(profit) (profit can be negative):
    - p_sum(c) ≥ v if p_count(c) ≥ k; or otherwise, sum^k(c) ≥ v

Experiment: Size vs. Dimensionality (50 and 100 cardinality)



- (50-C): 10^6 tuples, 0 skew, 50 cardinality, fragment size 3.
- (100-C): 10^6 tuples, 2 skew, 100 cardinality, fragment size 2.

Experiments on Real World Data

- UCI Forest CoverType data set
  - 54 dimensions, 581K tuples
  - Shell fragments of size 2 took 33 seconds and 325MB to compute
  - 3-D subquery with 1 instantiated D: 85ms~1.4 sec.
- Longitudinal Study of Vocational Rehab. Data
  - 24 dimensions, 8818 tuples
  - Shell fragments of size 3 took 0.9 seconds and 60MB to compute
  - 5-D query with 0 instantiated D: 227ms~2.6 sec.

Chapter 5: Data Cube Technology

- Data Cube Computation: Preliminary Concepts
- Data Cube Computation Methods
- Processing Advanced Queries by Exploring Data Cube Technology
- Multidimensional Data Analysis in Cube Space
- Summary

Processing Advanced Queries by Exploring Data Cube Technology

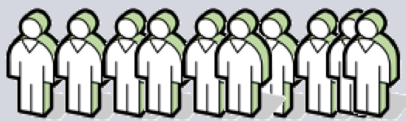
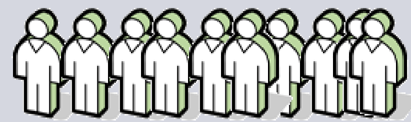
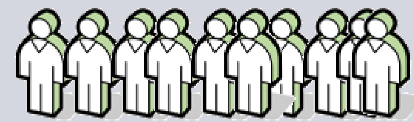
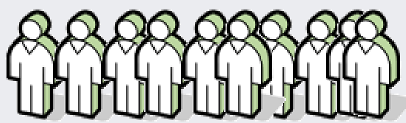
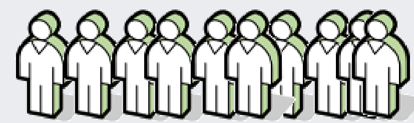
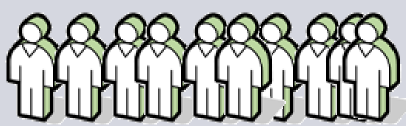
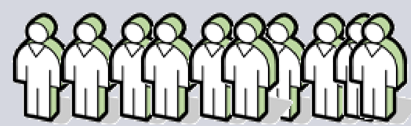
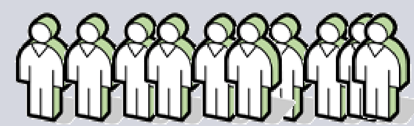
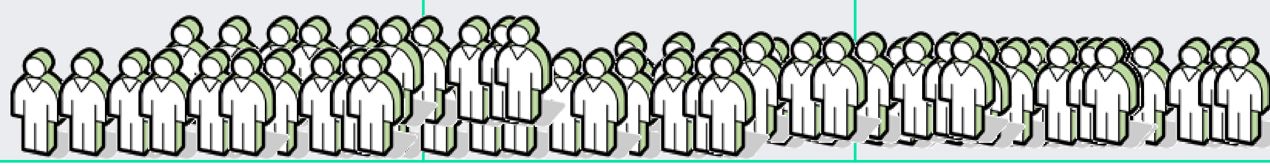
- Sampling Cube
  - X. Li, J. Han, Z. Yin, J.-G. Lee, Y. Sun, "Sampling Cube: A Framework for Statistical OLAP over Sampling Data", SIGMOD'08
- Ranking Cube
  - D. Xin, J. Han, H. Cheng, and X. Li. Answering top-k queries with multi-dimensional selections: The ranking cube approach. VLDB'06
- Other advanced cubes for processing data and queries
  - Stream cube, spatial cube, multimedia cube, text cube, RFID cube, etc. — to be studied in volume 2

Statistical Surveys and OLAP

- Statistical survey: A popular tool to collect information about a **population** based on a **sample**
  - Ex.: TV ratings, US Census, election polls
- A common tool in politics, health, market research, science, and many more
- An efficient way of collecting information (Data collection is expensive)
- Many **statistical tools** available, to determine validity
  - Confidence intervals
  - Hypothesis tests
- OLAP (multidimensional analysis) on survey data
  - highly desirable but can it be done well?

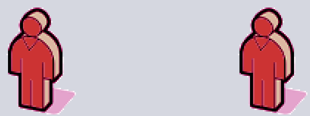
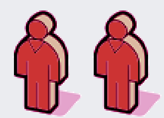
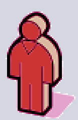
Surveys: Sample vs. Whole Population

Data is only a sample of **population**

Age\Education	High-school	College	Graduate
18			
19			
20			
…			

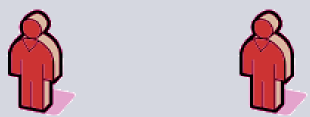
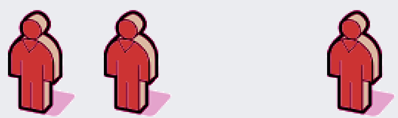
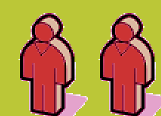
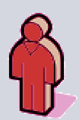
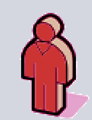
Problems for Drilling in Multidim. Space

Data is only a **sample** of population but samples could be small when drilling to certain multidimensional space

Age\Education	High-school	College	Graduate
18			
19			
20			
…			

OLAP on Survey (i.e., Sampling) Data

- Semantics of query is unchanged
- Input data has changed

Age/Education	High-school	College	Graduate
18			
19			
20			
…			

Challenges for OLAP on Sampling Data

- Computing confidence intervals in OLAP context
- No data?
  - Not exactly. No data in subspaces in cube
  - Sparse data
  - Causes include sampling bias and query selection bias
- Curse of dimensionality
  - Survey data can be high dimensional
  - Over 600 dimensions in real world example
  - Impossible to fully materialize

Example 1: Confidence Interval

What is the average income of 19-year-old high-school students?

Return not only query result but also confidence interval

Age/Education	High-school	College	Graduate
18			
19			
20			
...			

Confidence Interval

- Confidence interval at $\bar{x}$: $\bar{x} \pm t_c \hat{\sigma}_{\bar{x}}$
  - x is a sample of data set; $\bar{x}$ is the mean of sample
  - t_c is the critical t-value, calculated by a look-up
  - $\hat{\sigma}_{\bar{x}} = \frac{s}{\sqrt{l}}$ is the estimated standard error of the mean
- Example: \$50,000 $\pm$ \$3,000 with 95% confidence
  - Treat points in cube cell as samples
  - Compute confidence interval as traditional sample set
- Return answer in the form of confidence interval
  - Indicates **quality** of query answer
  - User selects desired confidence interval

Efficient Computing Confidence Interval Measures

- Efficient computation in all cells in data cube
  - Both mean and confidence interval are **algebraic**
  - Why confidence interval measure is algebraic?

$$\bar{x} \pm t_c \hat{\sigma}_{\bar{x}}$$

$\bar{x}$ is algebraic

$\hat{\sigma}_{\bar{x}} = \frac{s}{\sqrt{l}}$ where both *s* and l (count) are algebraic

- Thus one can calculate cells efficiently at more general cuboids without having to start at the base cuboid each time

Example 2: Query Expansion

What is the average income of 19-year-old college students?

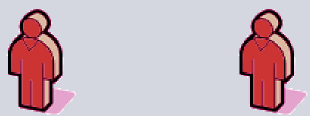
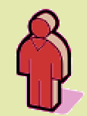
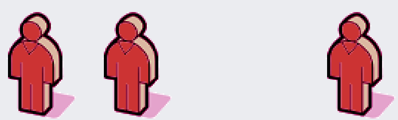
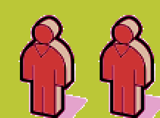
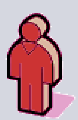
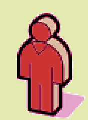
Age/Education	High-school	College	Graduate
18			
19			
20			
…			

Boosting Confidence by Query Expansion

- From the example: The queried cell "19-year-old college students" contains only 2 samples
- Confidence interval is large (i.e., low confidence). why?
  - Small sample size
  - High standard deviation with samples
- Small sample sizes can occur at relatively low dimensional selections
  - Collect more data?— expensive!
  - Use data in other cells? Maybe, but have to be careful

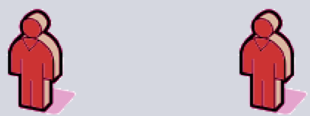
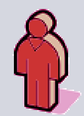
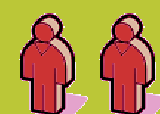
Intra-Cuboid Expansion: Choice 1

Expand query to include **18** and **20** year olds?

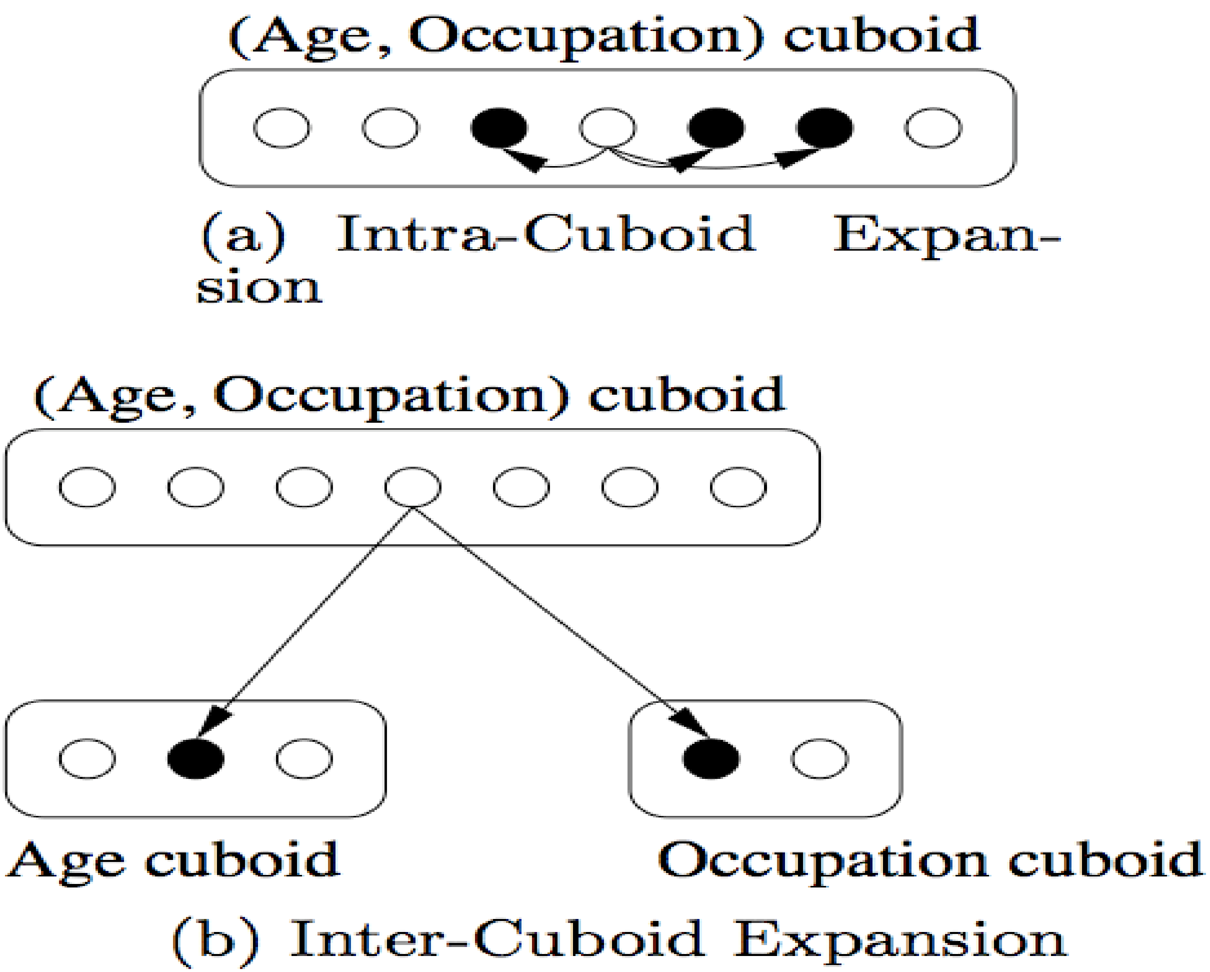
Age/Education	**High-school**	**College**	**Graduate**
18			
19			
20			
…			

Intra-Cuboid Expansion: Choice 2

Expand query to include **high-school** and **graduate** students?

Age/Education	**High-school**	**College**	**Graduate**
18			
19			
20			
…			

Query Expansion



Intra-Cuboid Expansion

- Combine other cells’ data into own to "boost" confidence
  - If share semantic and cube similarity
  - Use only if necessary
  - Bigger sample size will decrease confidence interval
- Cell segment similarity
  - Some dimensions are clear: **Age**
  - Some are fuzzy: **Occupation**
  - May need domain knowledge
- Cell value similarity
  - How to determine if two cells’ samples come from the same population?
  - Two-sample t-test (confidence-based)

Inter-Cuboid Expansion

- If a query dimension is
  - Not correlated with cube value
  - But is causing small sample size by drilling down too much
- Remove dimension (i.e., generalize to *) and move to a more general cuboid
- Can use two-sample t-test to determine similarity between two cells across cuboids
- Can also use a different method to be shown later

Query Expansion Experiments

- Real world sample data: 600 dimensions and 750,000 tuples
- 0.05% to simulate "sample" (allows error checking)

(a) Intra-Cuboid Expansion with **Age** dimension and **Average Number of Children** cube measure

Query		Average Query Answer Error			Sampling Sizes		
Gender	Marital	No Expand	Expand	% Improve	Population	Sample	Expanded
FEMALE	MARRIED	0.48	0.32	33%	2473.0	2.2	28.3
FEMALE	SINGLE	0.31	0.21	30%	612.6	0.6	6.4
FEMALE	DIVORCED	0.49	0.43	11%	321.1	0.3	3.4
MALE	MARRIED	0.42	0.21	49%	4296.8	4.4	37.6
MALE	SINGLE	0.26	0.21	16%	571.8	0.5	3.6
MALE	DIVORCED	0.33	0.27	19%	224.7	0.2	1.2
	Average	0.38	0.27	26%	1416.7	1.4	13.4

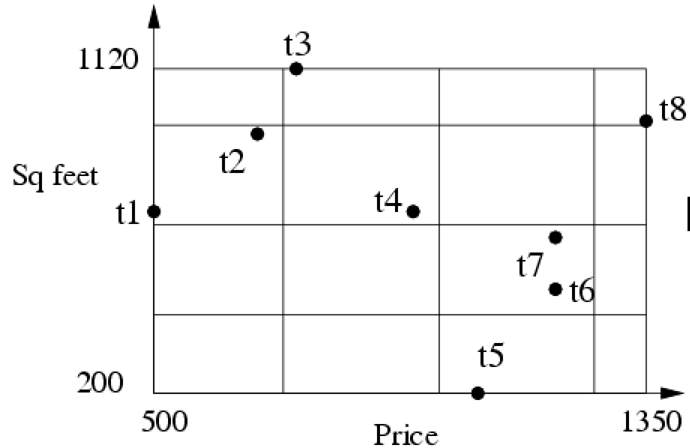
Ranking Cubes – Efficient Computation of Ranking queries

- Data cube helps not only OLAP but also ranked search
- **(top-k) ranking query**: only returns the best k results according to a user-specified preference, consisting of (1) a *selection condition* and (2) a *ranking function*
- Ex.: Search for apartments with expected price 1000 and expected square feet 800
  - Select top 1 from Apartment
  - where City = "LA" and Num_Bedroom = 2
  - order by [price – 1000]^2 + [sq feet - 800]^2 asc
- Efficiency question: Can we only search what we need?
  - Build a ranking cube on both selection dimensions and ranking dimensions

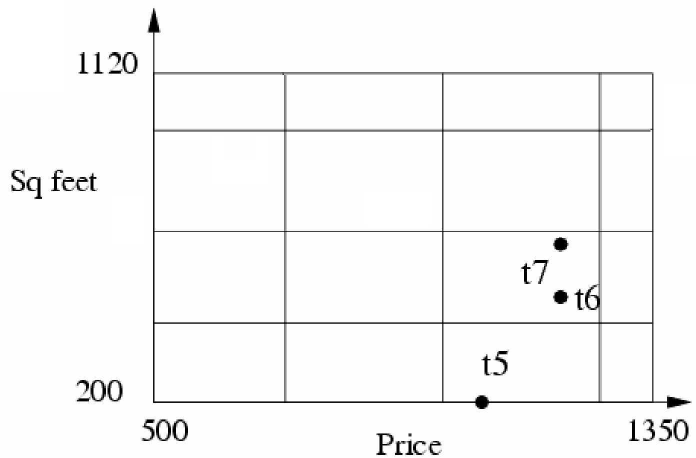
Ranking Cube: Partition Data on Both Selection and Ranking Dimensions

One single data partition as the template

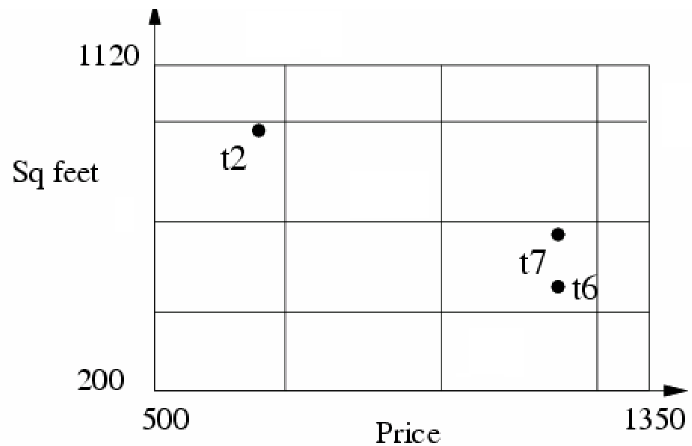
Slice the data partition by selection conditions



Partition for all data



Sliced Partition for city="LA"

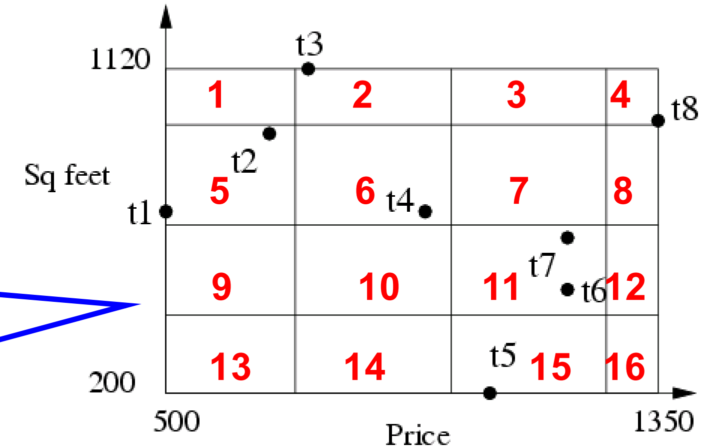


Sliced Partition for BR=2

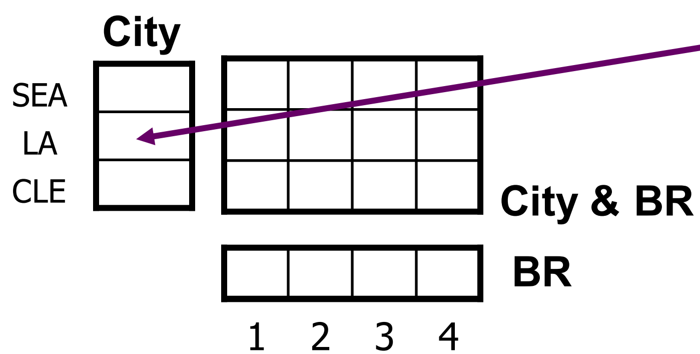
Materialize Ranking-Cube

tid	City	BR	Price	Sq feet
t1	SEA	1	500	600
t2	CLE	2	700	800
t3	SEA	1	800	900
t4	CLE	3	1000	1000
t5	LA	1	1100	200
t6	LA	2	1200	500
t7	LA	2	1200	560
t8	CLE	3	1350	1120

Step 1: Partition Data on Ranking Dimensions



Step 2: Group data by Selection Dimensions

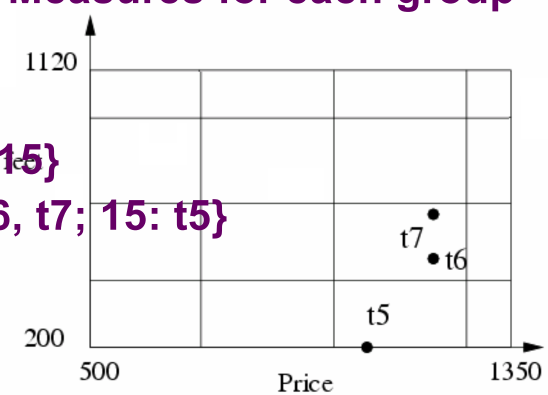


Step 3: Compute Measures for each group

For the cell (LA)

Block-level: {11, 15}

Data-level: {11: t6, t7; 15: t5}

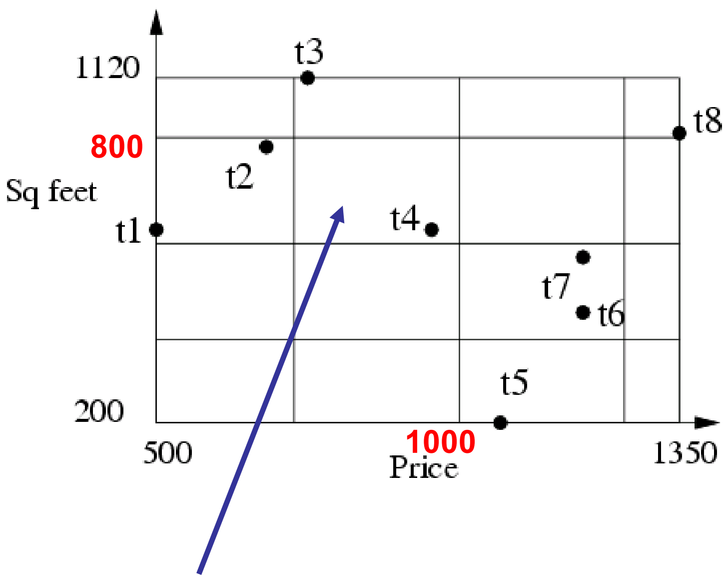


Search with Ranking-Cube: Simultaneously Push Selection and Ranking

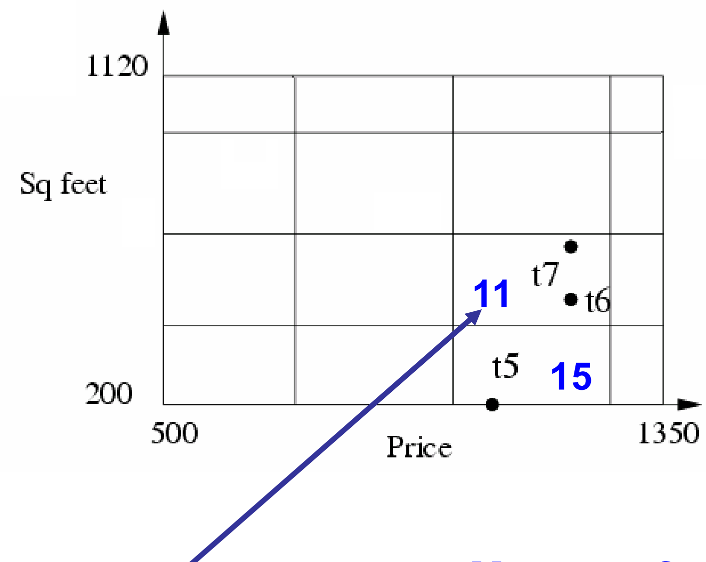
Select top 1 from Apartment
where city = "LA"
order by [price – 1000]^2 + [sq feet - 800]^2 asc

Bin boundary for price	**[500, 600, 800, 1100,1350]**
Bin boundary for sq feet	**[200, 400, 600, 800, 1120]**

Given the bin boundaries, locate the block with top score



Without ranking-cube: start search from here



With ranking-cube: start search from here

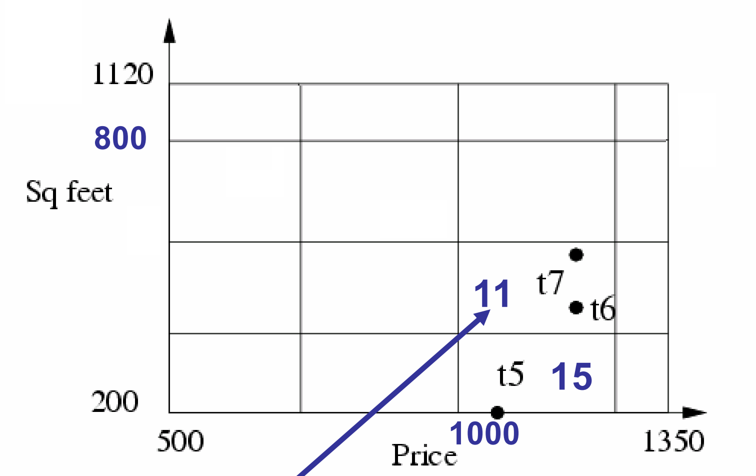
Measure for LA:
{11, 15}
{11: t6,t7; 15:t5}

Processing Ranking Query: Execution Trace

Select top 1 from Apartment
where city = "LA"
order by [price – 1000]^2 + [sq feet - 800]^2 asc

Bin boundary for price	**[500, 600, 800, 1100,1350]**
Bin boundary for sq feet	**[200, 400, 600, 800, 1120]**

f=[price-1000]^2 + [sq feet – 800]^2



With ranking-cube: start search from here

Measure for LA:
{11, 15}
{11: t6,t7; 15:t5}

Execution Trace:

1. **Retrieve High-level measure for LA {11, 15}**
2. **Estimate *lower bound score* for block 11, 15**

   f(block 11) = 40,000, f(block 15) = 160,000
3. **Retrieve block 11**
4. **Retrieve low-level measure for block 11**
5. **f(t6) = 130,000, f(t7) = 97,600**

   Output t7, done!

Ranking Cube: Methodology and Extension

- Ranking cube methodology
  - Push selection and ranking simultaneously
  - It works for many sophisticated ranking functions
- How to support high-dimensional data?
  - Materialize only those *atomic* cuboids that contain single selection dimensions
    - Uses the idea similar to high-dimensional OLAP
    - Achieves low space overhead and high performance in answering ranking queries with a high number of selection dimensions

Chapter 5: Data Cube Technology

- Data Cube Computation: Preliminary Concepts
- Data Cube Computation Methods
- Processing Advanced Queries by Exploring Data Cube Technology
- Multidimensional Data Analysis in Cube Space
- Summary

Multidimensional Data Analysis in Cube Space

- Prediction Cubes: Data Mining in Multi-Dimensional Cube Space
- Multi-Feature Cubes: Complex Aggregation at Multiple Granularities
- Discovery-Driven Exploration of Data Cubes

Data Mining in Cube Space

- Data cube greatly increases the analysis bandwidth
- Four ways to interact OLAP-styled analysis and data mining
  - Using cube space to define data space for mining
  - Using OLAP queries to generate features and targets for mining, e.g., multi-feature cube
  - Using data-mining models as building blocks in a multi-step mining process, e.g., prediction cube
  - Using data-cube computation techniques to speed up repeated model construction
    - Cube-space data mining may require building a model for each candidate data space
    - Sharing computation across model-construction for different candidates may lead to efficient mining

Prediction Cubes

- **Prediction cube**: A cube structure that stores prediction models in multidimensional data space and supports prediction in OLAP manner
- Prediction models are used as building blocks to define the interestingness of subsets of data, i.e., to answer which subsets of data indicate better prediction

How to Determine the Prediction Power of an Attribute?

- Ex. A customer table **D:**
  - Two dimensions **Z**: *Time* (*Month*, *Year*) and *Location* (*State, Country*)
  - Two features **X**: *Gender* and *Salary*
  - One class-label attribute Y: *Valued Customer*
- Q: “Are there times and locations in which the value of a customer depended greatly on the customers gender (i.e., Gender: predictiveness attribute **V**)?”
- Idea:
  - Compute the difference between the model built on that using **X** to predict Y and that built on using **X – V** to predict Y
  - If the difference is large, **V** must play an important role at predicting Y

Efficient Computation of Prediction Cubes

- Naïve method: Fully materialize the prediction cube, i.e., exhaustively build models and evaluate them for each cell and for each granularity
- Better approach: Explore score function decomposition that reduces prediction cube computation to data cube computation

Multidimensional Data Analysis in Cube Space

- Prediction Cubes: Data Mining in Multi-Dimensional Cube Space
- Multi-Feature Cubes: Complex Aggregation at Multiple Granularities
- Discovery-Driven Exploration of Data Cubes

Complex Aggregation at Multiple Granularities: Multi-Feature Cubes

- Multi-feature cubes (Ross, et al. 1998): Compute complex queries involving multiple dependent aggregates at multiple granularities
- Ex. Grouping by all subsets of {item, region, month}, find the maximum price in 2010 for each group, and the total sales among all maximum price tuples

```
select item, region, month, max(price), sum(R.sales)
from purchases
where year = 2010
cube by item, region, month: R
such that R.price = max(price)
```

- Continuing the last example, among the max price tuples, find the min and max shelf live, and find the fraction of the total sales due to tuple that have min shelf life within the set of all max price tuples

Multidimensional Data Analysis in Cube Space

- Prediction Cubes: Data Mining in Multi-Dimensional Cube Space
- Multi-Feature Cubes: Complex Aggregation at Multiple Granularities
- Discovery-Driven Exploration of Data Cubes

Discovery-Driven Exploration of Data Cubes

- Hypothesis-driven
  - exploration by user, huge search space
- Discovery-driven (Sarawagi, et al.'98)
  - Effective navigation of large OLAP data cubes
  - pre-compute measures indicating exceptions, guide user in the data analysis, at all levels of aggregation
  - Exception: significantly different from the value anticipated, based on a statistical model
  - Visual cues such as background color are used to reflect the degree of exception of each cell

Kinds of Exceptions and their Computation

- Parameters
  - SelfExp: surprise of cell relative to other cells at same level of aggregation
  - InExp: surprise beneath the cell
  - PathExp: surprise beneath cell for each drill-down path
- Computation of exception indicator (modeling fitting and computing SelfExp, InExp, and PathExp values) can be overlapped with cube construction
- Exception themselves can be stored, indexed and retrieved like precomputed aggregates

Examples: Discovery-Driven Data Cubes

item	all
region	all

Sum of sales	month											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Total		1%	-1%	0%	1%	3%	-1	-9%	-1%	2%	-4%	3%

Avg sales	month											
item	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Sony b/w printer		9%	-8%	2%	-5%	14%	-4%	0%	41%	-13%	-15%	-11%
Sony color printer		0%	0%	3%	2%	4%	-10%	-13%	0%	4%	-6%	4%
HP b/w printer		-2%	1%	2%	3%	8%	0%	-12%	-9%	3%	-3%	6%
HP color printer		0%	0%	-2%	1%	0%	-1%	-7%	-2%	1%	-5%	1%
IBM home computer		1%	-2%	-1%	-1%	3%	3%	-10%	4%	1%	-4%	-1%
IBM laptop computer		0%	0%	-1%	3%	4%	2%	-10%	-2%	0%	-9%	3%
Toshiba home computer		-2%	-5%	1%	1%	-1%	1%	5%	-3%	-5%	-1%	-1%
Toshiba laptop computer		1%	0%	3%	0%	-2%	-2%	-5%	3%	2%	-1%	0%
Logitech mouse		3%	-2%	-1%	0%	4%	6%	-11%	2%	1%	-4%	0%
Ergo-way mouse		0%	0%	2%	3%	1%	-2%	-2%	-5%	0%	-5%	8%

item	IBM home computer

Avg sales	month											
region	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
North		-1%	-3%	-1%	0%	3%	4%	-7%	1%	0%	-3%	-3%
South		-1%	1%	-9%	6%	-1%	-39%	9%	-34%	4%	1%	7%
East		-1%	-2%	2%	-3%	1%	18%	-2%	11%	-3%	-2%	-1%
West		4%	0%	-1%	-3%	5%	1%	-18%	8%	5%	-8%	1%

Chapter 5: Data Cube Technology

- Data Cube Computation: Preliminary Concepts
- Data Cube Computation Methods
- Processing Advanced Queries by Exploring Data Cube Technology
- Multidimensional Data Analysis in Cube Space
- Summary

Data Cube Technology: Summary

- Data Cube Computation: Preliminary Concepts
- Data Cube Computation Methods
  - MultiWay Array Aggregation
  - BUC
  - Star-Cubing
  - High-Dimensional OLAP with Shell-Fragments
- Processing Advanced Queries by Exploring Data Cube Technology
  - Sampling Cubes
  - Ranking Cubes
- Multidimensional Data Analysis in Cube Space
  - Discovery-Driven Exploration of Data Cubes
  - Multi-feature Cubes
  - Prediction Cubes

Ref.(I) Data Cube Computation Methods

- S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. VLDB’96
- D. Agrawal, A. E. Abbadi, A. Singh, and T. Yurek. Efficient view maintenance in data warehouses. SIGMOD’97
- K. Beyer and R. Ramakrishnan. Bottom-Up Computation of Sparse and Iceberg CUBEs.. SIGMOD’99
- M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. D. Ullman. Computing iceberg queries efficiently. VLDB’98
- J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals. Data Mining and Knowledge Discovery, 1:29–54, 1997.
- J. Han, J. Pei, G. Dong, K. Wang. Efficient Computation of Iceberg Cubes With Complex Measures. SIGMOD’01
- L. V. S. Lakshmanan, J. Pei, and J. Han, Quotient Cube: How to Summarize the Semantics of a Data Cube, VLDB'02
- X. Li, J. Han, and H. Gonzalez, High-Dimensional OLAP: A Minimal Cubing Approach, VLDB'04
- Y. Zhao, P. M. Deshpande, and J. F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. SIGMOD’97
- K. Ross and D. Srivastava. Fast computation of sparse datacubes. VLDB’97
- D. Xin, J. Han, X. Li, B. W. Wah, Star-Cubing: Computing Iceberg Cubes by Top-Down and Bottom-Up Integration, VLDB'03
- D. Xin, J. Han, Z. Shao, H. Liu, C-Cubing: Efficient Computation of Closed Cubes by Aggregation-Based Checking, ICDE'06

Ref. (II) Advanced Applications with Data Cubes

- D. Burdick, P. Deshpande, T. S. Jayram, R. Ramakrishnan, and S. Vaithyanathan. OLAP over uncertain and imprecise data. VLDB'05
- X. Li, J. Han, Z. Yin, J.-G. Lee, Y. Sun, "Sampling Cube: A Framework for Statistical OLAP over Sampling Data", SIGMOD'08
- C. X. Lin, B. Ding, J. Han, F. Zhu, and B. Zhao. Text Cube: Computing IR measures for multidimensional text database analysis. ICDM'08
- D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient OLAP operations in spatial data warehouses. SSTD'01
- N. Stefanovic, J. Han, and K. Koperski. Object-based selective materialization for efficient implementation of spatial data cubes. IEEE Trans. Knowledge and Data Engineering, 12:938–958, 2000.
- T. Wu, D. Xin, Q. Mei, and J. Han. Promotion analysis in multidimensional space. VLDB'09
- T. Wu, D. Xin, and J. Han. ARCube: Supporting ranking aggregate queries in partially materialized data cubes. SIGMOD'08
- D. Xin, J. Han, H. Cheng, and X. Li. Answering top-k queries with multi-dimensional selections: The ranking cube approach. VLDB'06
- J. S. Vitter, M. Wang, and B. R. Iyer. Data cube approximation and histograms via wavelets. CIKM'98
- D. Zhang, C. Zhai, and J. Han. Topic cube: Topic modeling for OLAP on multi-dimensional text databases. SDM'09

Ref. (III) Knowledge Discovery with Data Cubes

- R. Agrawal, A. Gupta, and S. Sarawagi. Modeling multidimensional databases. ICDE'97
- B.-C. Chen, L. Chen, Y. Lin, and R. Ramakrishnan. Prediction cubes. VLDB'05
- B.-C. Chen, R. Ramakrishnan, J.W. Shavlik, and P. Tamma. Bellwether analysis: Predicting global aggregates from local regions. VLDB'06
- Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang, Multi-Dimensional Regression Analysis of Time-Series Data Streams, VLDB'02
- G. Dong, J. Han, J. Lam, J. Pei, K. Wang. Mining Multi-dimensional Constrained Gradients in Data Cubes. VLDB' 01
- R. Fagin, R. V. Guha, R. Kumar, J. Novak, D. Sivakumar, and A. Tomkins. Multi-structural databases. PODS'05
- J. Han. Towards on-line analytical mining in large databases. SIGMOD Record, 27:97–107, 1998
- T. Imielinski, L. Khachiyan, and A. Abdulghani. Cubegrades: Generalizing association rules. Data Mining & Knowledge Discovery, 6:219–258, 2002.
- R. Ramakrishnan and B.-C. Chen. Exploratory mining in cube space. Data Mining and Knowledge Discovery, 15:29–54, 2007.
- K. A. Ross, D. Srivastava, and D. Chatziantoniou. Complex aggregation at multiple granularities. EDBT'98
- S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of OLAP data cubes. EDBT'98
- G. Sathe and S. Sarawagi. Intelligent Rollups in Multidimensional OLAP Data. *VLDB'01*

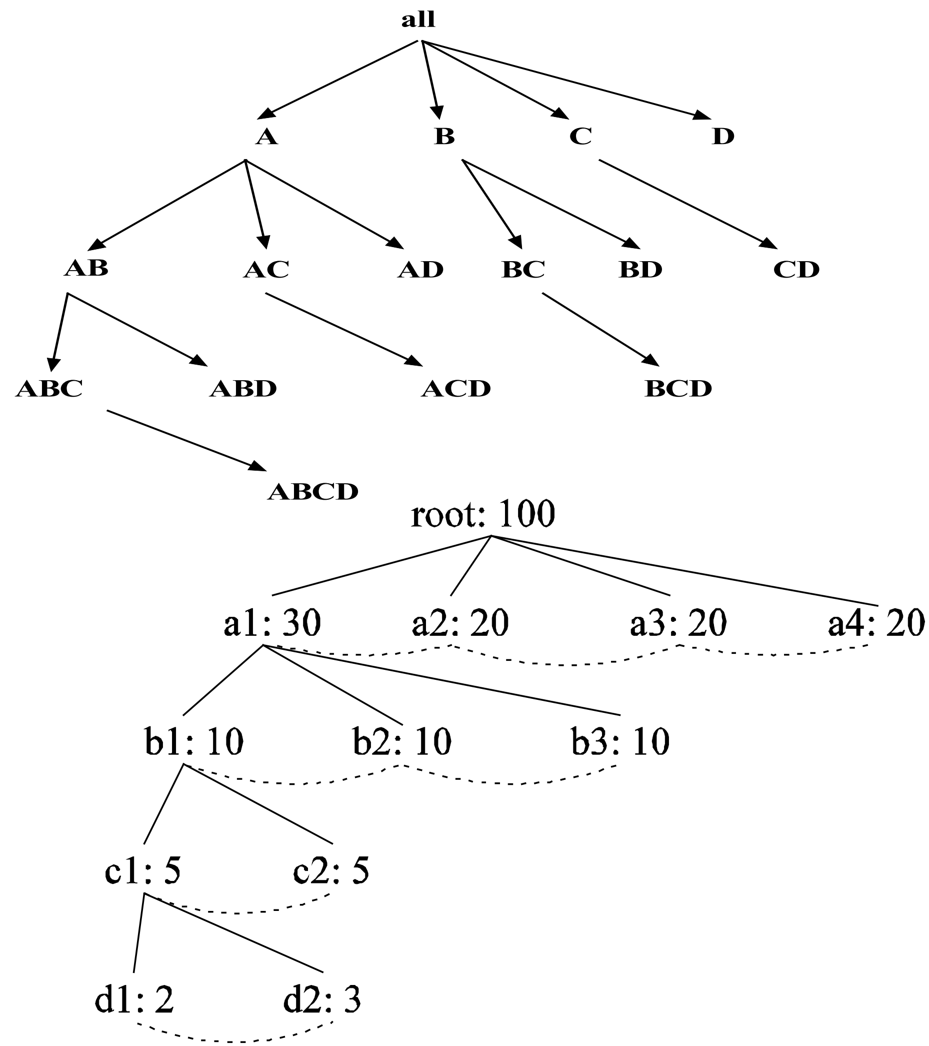
Surplus Slides

Chapter 5: Data Cube Technology

- Efficient Methods for Data Cube Computation
  - Preliminary Concepts and General Strategies for Cube Computation
  - Multiway Array Aggregation for Full Cube Computation
  - BUC: Computing Iceberg Cubes from the Apex Cuboid Downward
  - H-Cubing: Exploring an H-Tree Structure
  - Star-cubing: Computing Iceberg Cubes Using a Dynamic Star-tree Structure
  - Precomputing Shell Fragments for Fast High-Dimensional OLAP
- Data Cubes for Advanced Applications
  - Sampling Cubes: OLAP on Sampling Data
  - Ranking Cubes: Efficient Computation of Ranking Queries
- Knowledge Discovery with Data Cubes
  - Discovery-Driven Exploration of Data Cubes
  - Complex Aggregation at Multiple Granularity: Multi-feature Cubes
  - Prediction Cubes: Data Mining in Multi-Dimensional Cube Space
- Summary

H-Cubing: Using H-Tree Structure

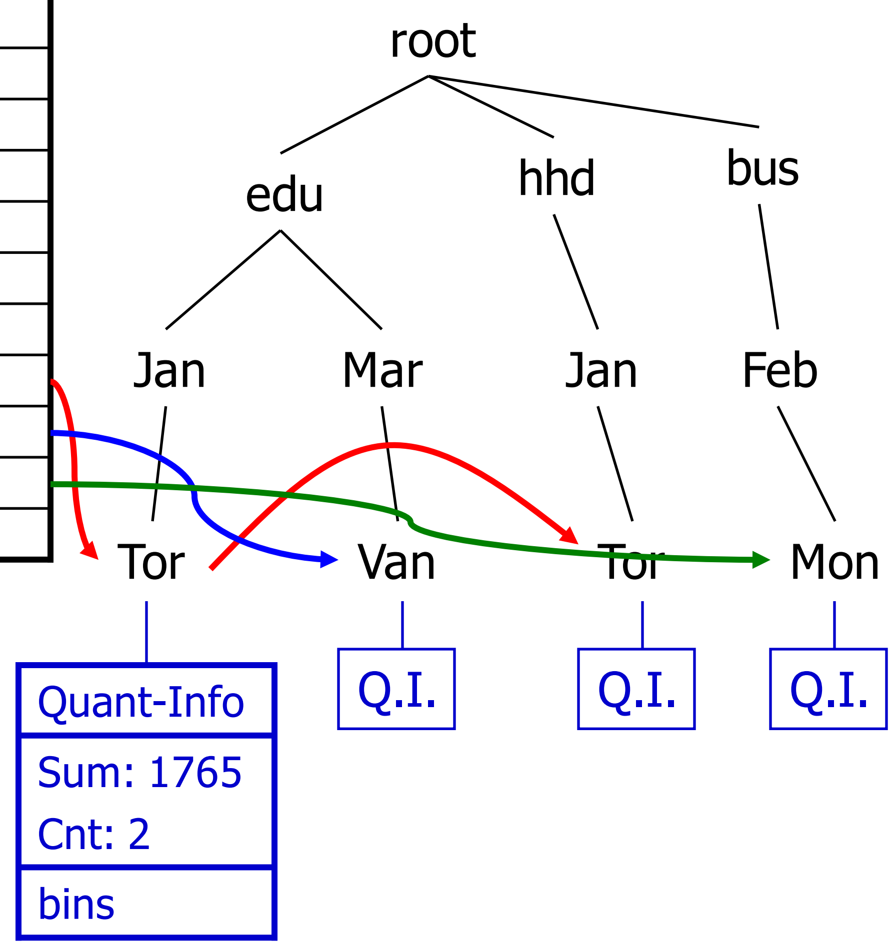
- Bottom-up computation
- Exploring an H-tree structure
- If the current computation of an H-tree cannot pass min_sup, do not proceed further (pruning)
- No simultaneous aggregation



H-tree: A Prefix Hyper-tree

Header table

Attr. Val.	Quant-Info	Side-link
Edu	Sum:2285 …	
Hhd	…	
Bus	…	
…	…	
Jan	…	
Feb	…	
…	…	
Tor	**…**	
Van	**…**	
Mon	**…**	
…	…	



Month	City	Cust_grp	Prod	Cost	Price
Jan	Tor	Edu	Printer	500	485
Jan	Tor	Hhd	TV	800	1200
Jan	Tor	Edu	Camera	1160	1280
Feb	Mon	Bus	Laptop	1500	2500
Mar	Van	Edu	HD	540	520
…	…	…	…	…	…

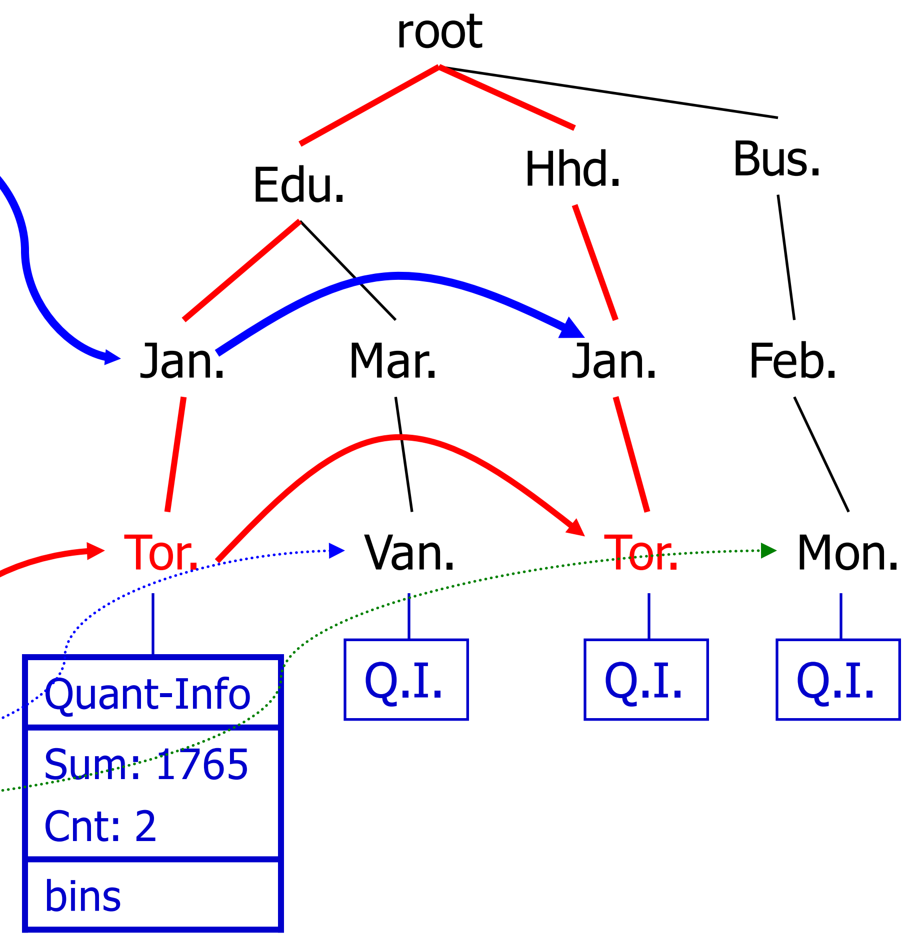
Computing Cells Involving "City"

Header Table H_{Tor}

Attr. Val.	Q.I.	Side-link
Edu	…	
Hhd	…	
Bus	…	
…	…	
Jan	**…**	
Feb	…	
…	…	

Attr. Val.	Quant-Info	Side-link
Edu	Sum:2285 …	
Hhd	…	
Bus	…	
…	…	
Jan	…	
Feb	…	
…	…	
Tor	**…**	
Van	…	
Mon	…	
…	…	

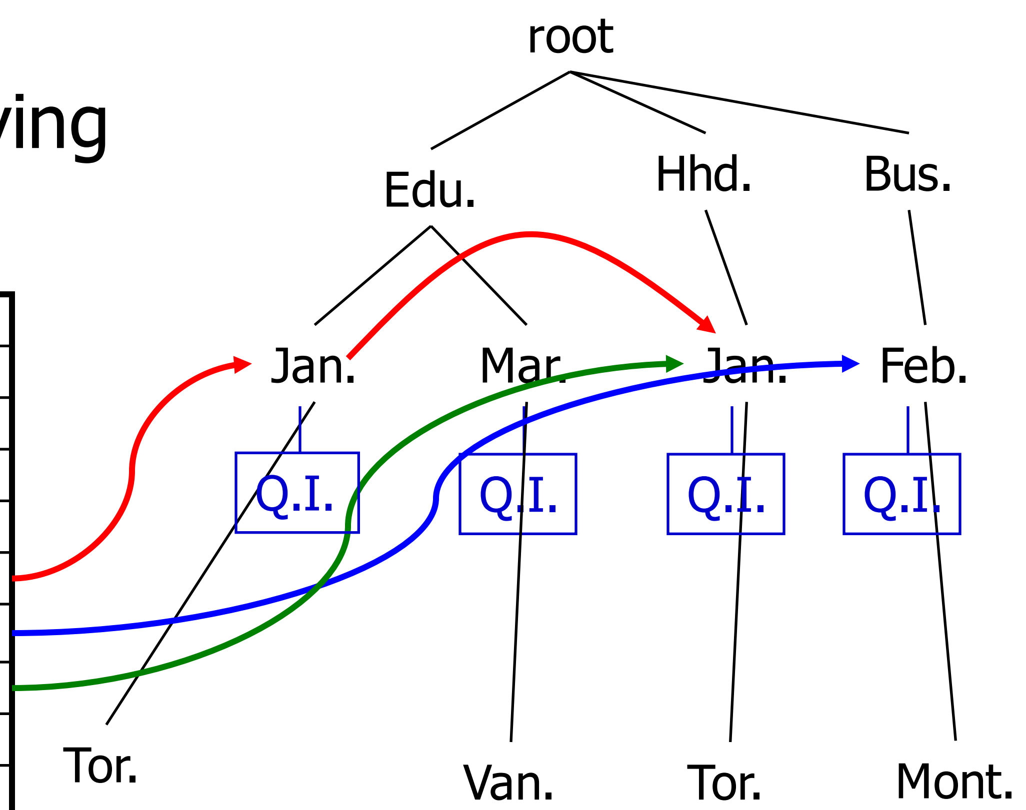
From (*, *, Tor) to (*, Jan, Tor)



Computing Cells Involving Month But No City

1. Roll up quant-info
2. Compute cells involving month but no city

Attr. Val.	Quant-Info	Side-link
Edu.	Sum:2285 …	
Hhd.	…	
Bus.	…	
…	…	
Jan.	…	
Feb.	…	
Mar.	…	
…	…	
Tor.	…	
Van.	…	
Mont.	…	
…	…	



Top-k OK mark: if Q.I. in a child passes top-k avg threshold, so does its parents. No binning is needed!

Computing Cells Involving Only Cust_grp

Check header table directly

Attr. Val.	Quant-Info	Side-link
Edu	**Sum:2285 …**	
Hhd	**…**	
Bus	**…**	
…	…	
Jan	…	
Feb	…	
Mar	…	
…	…	
Tor	…	
Van	…	
Mon	…	
…	…