COURSE NAME:
DATA WAREHOUSING & DATA MINING

LECTURE 9
TOPICS TO BE COVERED:

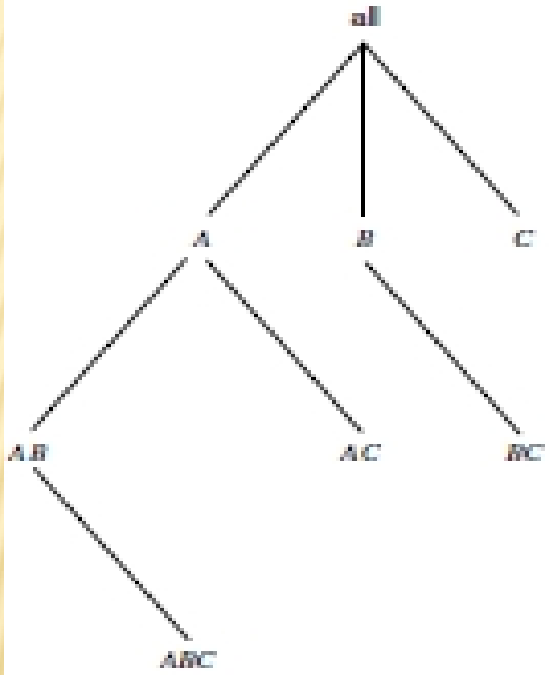
- Complex aggregation at multiple granularities (contd)

COMPUTE AGGREGATES BY VISITING CUBE CELLS

- **Compute aggregates by visiting (i.e., accessing the values at) cube cells:** The order in which cells are visited can be optimized so as to *minimize the number of times that each cell must be revisited, thereby reducing memory access and storage costs. The trick is* to exploit this ordering so that partial aggregates can be computed simultaneously, and any unnecessary revisiting of cells is avoided.

BUC :COMPUTING ICEBERG CUBES FROM APEX CUBOID DOWNWARD.

- **BUC** is an algorithm for the computation of sparse and iceberg cubes.
- BUC constructs the cube from the apex cuboid toward the base cuboid. This allows BUC to share data partitioning costs. This order of processing also allows BUC to prune during construction, using the Apriori property.

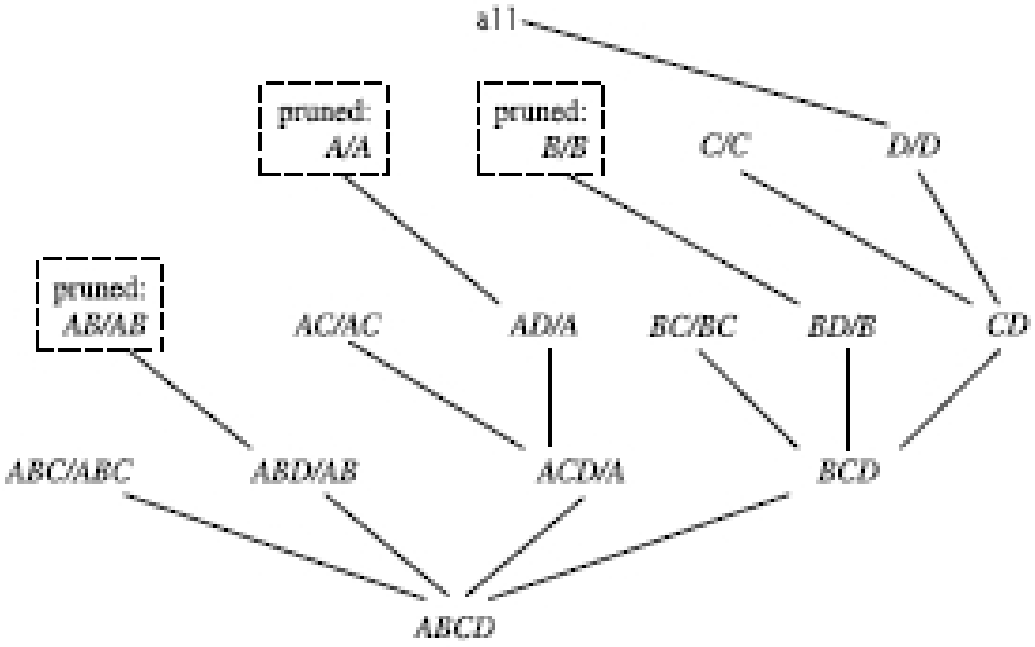


BUC's exploration for the computation of a 3-D data cube. Note that the computation starts from the apex cuboid.

STAR-CUBING: COMPUTING ICEBERG CUBES USING A DYNAMIC STAR-TREE STRUCTURE

- It integrates top-down and bottom-up cube computation and explores both multidimensional aggregation.
- It operates from a data structure called a star-tree, which performs lossless data compression, thereby reducing the computation time and memory requirements.
- The **Star-Cubing algorithm** explores both the bottom-up and top-down computation models as follows: On the global computation order, it uses the bottom-up model. However, it has a sublayer underneath based on the top-down model, which explores the notion of *shared dimensions*

STAR-CUBING



Star-Cubing: Bottom-up computation with top-down expansion of shared dimensions.

STAR-TREE CONSTRUCTION

Star-tree construction. A base cuboid table is shown below. There are 5 tuples and 4 dimensions. The cardinalities for dimensions *A, B, C, D are 2, 4, 4, 4,*
respectively.

Base (Cuboid) Table: Before star reduction.

A	B	C	D	count
a_1	b_1	c_1	d_1	1
a_1	b_1	c_4	d_3	1
a_1	b_2	c_2	d_2	1
a_2	b_3	c_3	d_4	1
a_2	b_4	c_3	d_4	1

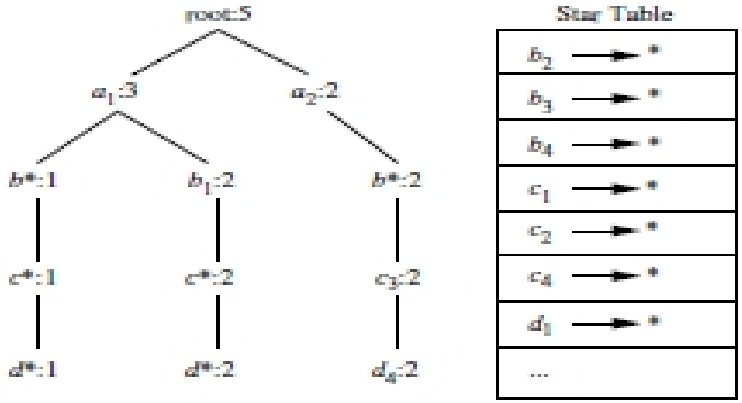
One-Dimensional Aggregates.

Dimension	count = 1	count $\geq$ 2
A	—	$a_1(3), a_2(2)$
B	b_2, b_3, b_4	$b_1(2)$
C	c_1, c_2, c_4	$c_3(2)$
D	d_1, d_2, d_3	$d_4(2)$

STAR-TREE CONSTRUCTION

Compressed Base Table: After star reduction.

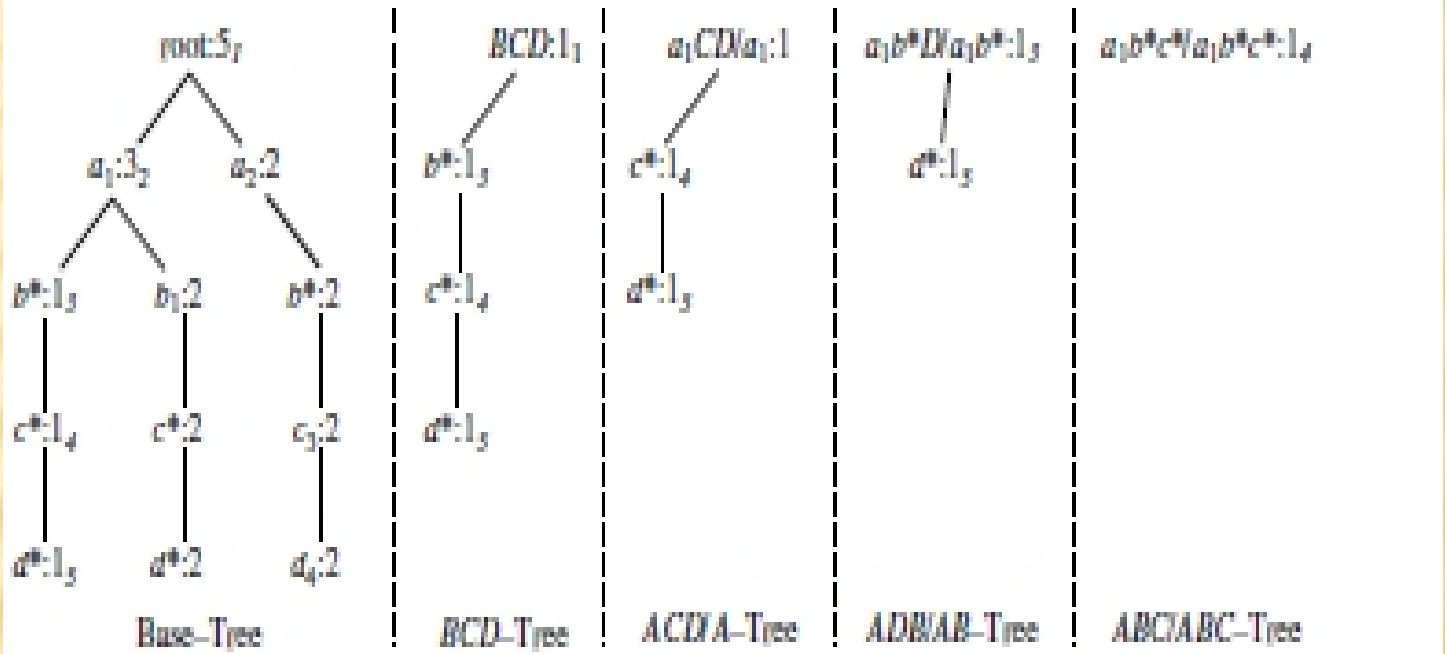
A	B	C	D	count
a_1	b_1	*	*	2
a_1	*	*	*	1
a_2	*	c_3	d_4	2



Star-tree and star-table.

STAR-CUBING

- Star-Cubing is generated Using the star-tree we start process of aggregation by traversing in a bottom-up fashion. Traversal is depth-first. The first stage (i.e., the processing of the first branch of the tree) is shown in next slide. The leftmost tree in the figure is the base star-tree. Each attribute value is shown with its corresponding aggregate value. In addition, subscripts by the nodes in the tree show the order of traversal. The remaining four trees are *BCD, ACD/A,ABD/AB, ABC/ABC*



Aggregation Stage One: Processing of the left-most branch of BaseTree.

PRECOMPUTING SHELL FRAGMENTS FOR FAST HIGH-DIMENSIONAL OLAP

- To illustrate the shell fragment approach, We first look at how to construct the inverted index for the given database.
- **Construct the inverted index.** For each attribute value in each dimension, list the tuple identifiers (*TIDs) of all the tuples that have that value. For example, attribute value a2* appears in tuples 4 and 5. The TIDlist for *a2 then contains exactly two items, namely* 4 and 5. The resulting inverted index table is shown in next slide . It retains all of the information of the original database. It uses exactly the same amount of memory as the original database

The original database.

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

The inverted index.

Attribute Value	Tuple ID 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

- **Compute shell fragments**. Suppose we are to compute the shell fragments of size 3. We first divide the five dimensions into two fragments, namely (*A, B, C) and (D, E). For each* fragment, we compute the full local data cube by intersecting the TIDlists in next slide in a top-down depth-first order in the cuboid lattice.
- To compute the cell(*a1, b2, *), we intersect the tuple ID lists of a1 and b2 to obtain a new list of {2, 3}*

Cuboid *AB*.

Cell	Intersection	Tuple ID List	List Size
(a_1, b_1)	$\{1, 2, 3\} \cap \{1, 4, 5\}$	$\{1\}$	1
(a_1, b_2)	$\{1, 2, 3\} \cap \{2, 3\}$	$\{2, 3\}$	2
(a_2, b_1)	$\{4, 5\} \cap \{1, 4, 5\}$	$\{4, 5\}$	2
(a_2, b_2)	$\{4, 5\} \cap \{2, 3\}$	$\{\}$	0

Cuboid *DE*.

Cell	Intersection	Tuple ID List	List Size
(d_1, e_1)	$\{1, 3, 4, 5\} \cap \{1, 2\}$	$\{1\}$	1
(d_1, e_2)	$\{1, 3, 4, 5\} \cap \{3, 4\}$	$\{3, 4\}$	2
(d_1, e_3)	$\{1, 3, 4, 5\} \cap \{5\}$	$\{5\}$	1
(d_2, e_1)	$\{2\} \cap \{1, 2\}$	$\{2\}$	1

COMPLEX AGGREGATION AT MULTIPLE GRANULARITY: MULTIFEATURE CUBES

- Data cubes facilitate the answering of data mining queries as they allow the computation of aggregate data at multiple levels of granularity.
- *multifeature cubes, which compute complex queries involving multiple dependent* aggregates at multiple granularity. These cubes are very useful in practice. Many complex data mining queries can be answered by multifeature cubes without any significant increase in computational cost, in comparison to cube computation for simple queries with standard data cubes.

SIMPLE DATA CUBE QUERY

- Query 1: A **simple data cube query**. Find the total sales in 2004, broken down by *item, region, and month, with subtotals for each dimension.*
- To answer Query 1, a data cube is constructed that aggregates the total sales at the following eight different levels of granularity: {(*item, region, month*), (*item*, *region*), (*item, month*), (*month, region*), (*item*), (*month*), (*region*), ()}, *where () represents all.*
- Query 1 uses a typical data cube like that introduced in the previous chapter. We call such a data cube a simple data cube because it does not involve any dependent aggregates.

COMPLEX QUERY

- Query 2: A **complex query.** Grouping by all subsets of f*item, region, monthg, find the* maximum *price in 2004 for each group and the total sales among all maximum price* tuples.
- The specification of such a query using standard SQL can be long, repetitive, and difficult to optimize and maintain. Alternatively, Query 2 can be specified concisely using an extended SQL syntax as follows:

```
select item, region, month, max(price), sum(R.sales)
from Purchases
where year = 2004
cube by item, region, month: R
such that R.price = max(price)
```

DATA GRANULARITY

- A data warehouse typically stores data in different levels of granularity or summarization, depending on the data requirements of the business. If an enterprise needs data to assist strategic planning, then only highly summarized data is required. The lower the level of granularity of data required by the enterprise, the higher the number of resources (specifically data storage) required to build the data warehouse. The different levels of summarization in order of increasing granularity are:
  - Current operational data
  - Historical operational data
  - Aggregated data
  - Metadata

DATA GRANULARITY

- Current and historical operational data are taken, unmodified, directly from operational systems. Historical data is operational level data no longer queried on a regular basis, and is often archived onto secondary storage.
- Aggregated, or summary, data is a filtered version of the current operational data. The design of the data warehouse affects how the current data is aggregated. Considerations for generating summary data include the period of time used to aggregate the data (for example, weekly, monthly, and so on), and the parts of the operational data to be summarized. For example, an organization can choose to aggregate at the part level the quantity of parts sold per sales representative per week.
- There may be several levels of summary data. It may be necessary to create summary level data based on an aggregated version of existing summary data. This can give an organization an even higher level view of the business. For example, an organization can choose to aggregate summary level data further by generating the quantity of parts sold per month.
- Metadata does not contain any operational data, but is used to document the way the data warehouse is constructed. Metadata can describe the structure of the data warehouse, source of the data, rules used to summarize the data at each level, and any transformations of the data from the operational systems.

Granularity

Granularity describes the level of detail stored in the physical warehouse