COURSE NAME:
DATA WAREHOUSING & DATA MINING

LECTURE 8
TOPICS TO BE COVERED:

- Complex aggregation at multiple granularities

DATA GENERALIZATION

- **Data generalization** is a process that abstracts a large set of task-relevant data in a database from a relatively low conceptual level to higher conceptual levels.
- Data generalization approaches include data cube–based data aggregation and attribute oriented induction.
- From a data analysis point of view, data generalization is a form of *descriptive data mining. Descriptive data mining describes data in a concise and summarative manner* and presents interesting general properties of the data.

EFFICIENT METHODS FOR DATA CUBE COMPUTATION

- A **data cube** consists of a lattice of cuboids. Each cuboid corresponds to a different degree of summarization of the given multidimensional data.
- Data cube computation is an essential task in data warehouse implementation. The precomputation of all or part of a data cube can greatly reduce the response time and enhance the performance of on-line analytical processing.

CUBE MATERIALIZATION

- Full Cube
- Iceberg Cube
- Closed Cube
- Shell Cube

FULL CUBE

- **Full Cube:-**all the cells of all of the cuboids for a given data cube. Thus, precomputation of the full cube can require huge and often excessive amounts of memory.
- **Full materialization** refers to the computation of all of the cuboids in a data cube lattice.

ICEBERG CUBE

- **Iceberg Cube:-**partially materialized cubes are known as iceberg cubes. The minimum threshold is called the minimum support threshold, or *minimum support(min sup)*
- **Partial materialization** refers to the selective computation of a subset of the cuboid cells in the lattice. Iceberg cubes and shell fragments are examples of partial materialization.
- **An iceberg cube** is a data cube that stores only those cube cells whose aggregate value (e.g., count) is above some minimum support threshold.

ICEBERG CUBE(EXAMPLE)

- An iceberg cube can be specified with an SQL query, as shown in the following example.

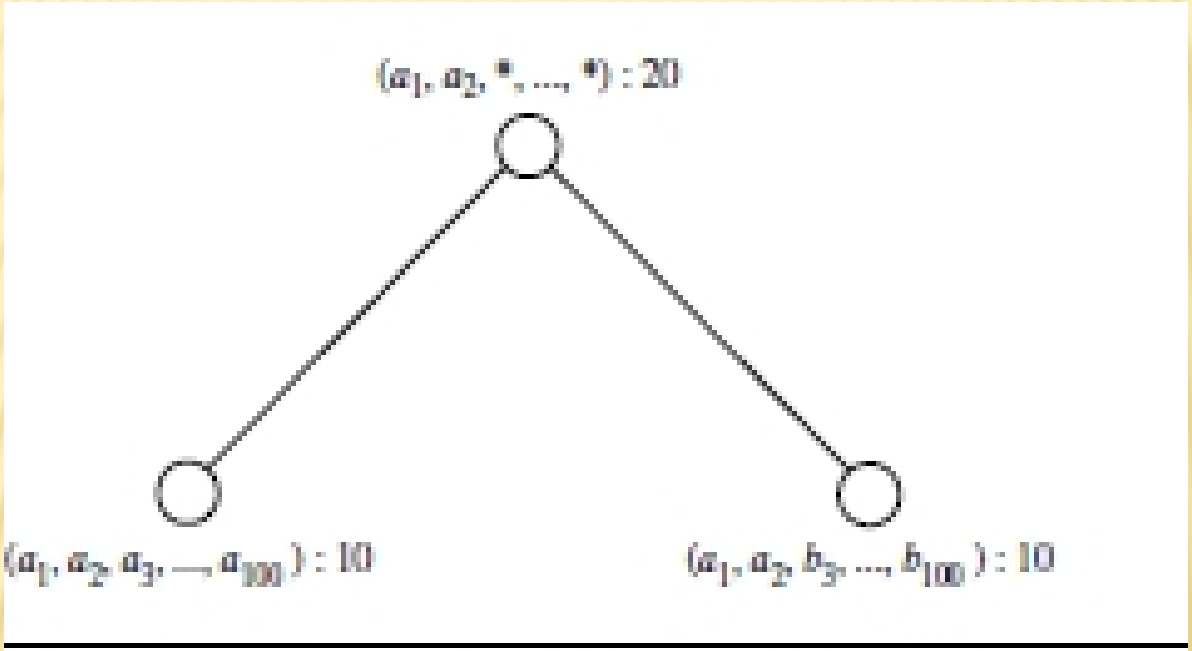
```
compute cube sales iceberg as
select month, city, customer group, count(*)
from salesInfo
cube by month, city, customer group
having count(*) >= min sup
```

CLOSED CUBE

- To systematically compress a data cube, we need *closed cell.*
- *A **closed cube** is a data cube consisting of only* closed cells.
- *A cell, c, is a closed cell if there exists no cell, d, such that d is a specialization* (descendant) of cell *c (i.e, where d is obtained by replacing a in c with a non- value),* and *d has the same measure value as c.*

- For example, the three cells derived above are the three closed cells of the data cube for the data set: {(*a1, a2, a3, - - - -, a100) : 10, (a1, a2, b3, - - - -, b100) : 10}.They form the* lattice of a closed cube as shown in Figure. Other nonclosed cells can be derived from their corresponding closed cells in this lattice. For example, "(*a1, * ,*, - - - - , *) : 20" can* be derived from "(*a1, a2, - - - -,*) : 20" because the former is a generalized nonclosed* cell of the latter. Similarly, we have "(*a1, a2, b3, *,- - - -,*) : 10".*

CLOSED CELL



Three closed cells forming the lattice of a closed cube.

SHELL CUBE

- Another strategy for **partial materialization** is to precompute only the cuboids involving a small number of dimensions, These cuboids form a cube shell for the corresponding data cube

- For **shell fragments** of a data cube, only some cuboids involving a small number of dimensions are computed. Queries on additional ombinations of the dimensions can be computed on the fly.

OPTIMIZATION TECHNIQUES FOR EFFICIENT COMPUTATION OF DATA CUBES

- Following are general optimization techniques for the efficient computation of data cubes.
    1. Sorting, hashing, and grouping.
    2. Simultaneous aggregation and caching intermediate results.
    3. Aggregation from the smallest child, when there exist multiple child cuboids.
    4. The Apriori pruning method can be explored to compute iceberg cubes efficiently.

DATA CUBE COMPUTATION METHODS

1. MultiWay array aggregation for full data computation.
2. BUC :Computing iceberg cubes by exploring ordering and sorting for efficient top-down computation.
3. Star-Cubing for integration of top-down and bottom-up computation using a star-tree structure.
4. High-dimensional OLAP by precomputing only the partitioned shell fragments.

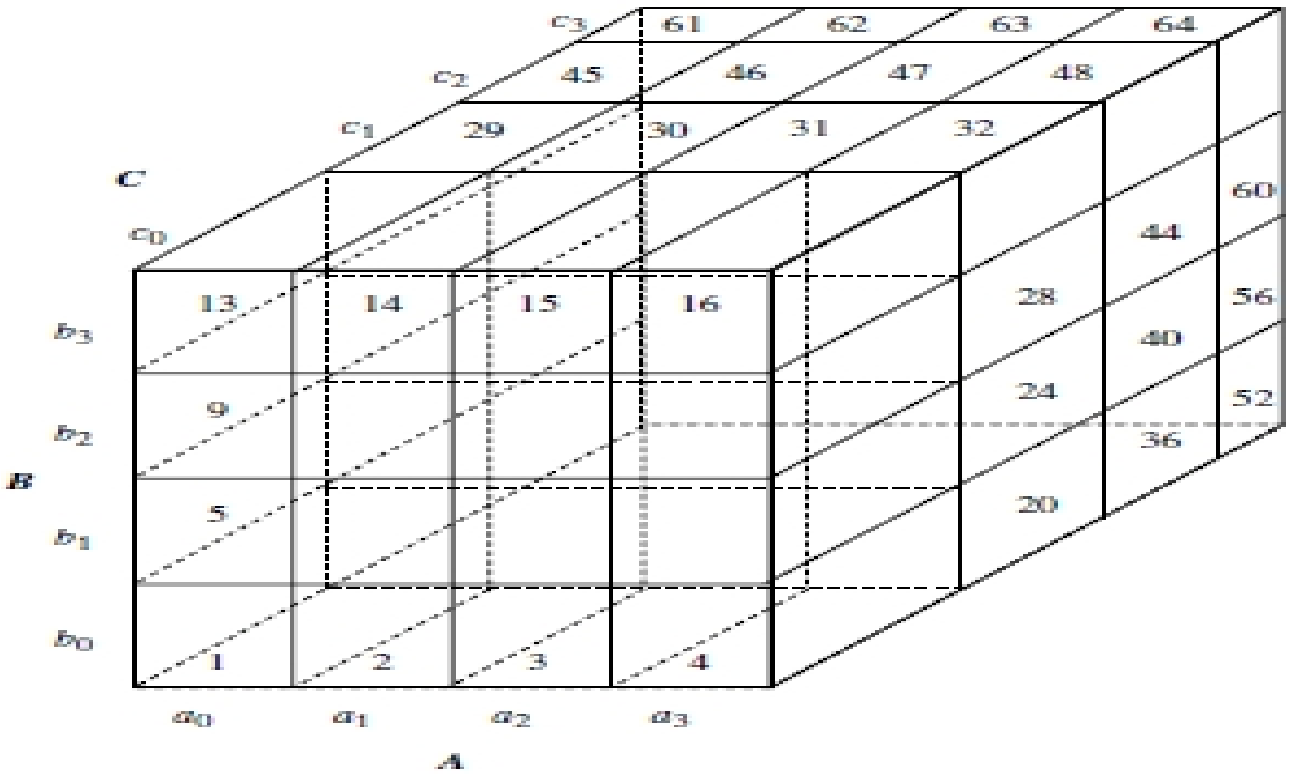
MULTIWAY ARRAY AGGREGATION FOR FULL DATA COMPUTATION

- **The Multiway Array Aggregation** (or simply MultiWay) method computes a full data cube by using a multidimensional array as its basic data structure. It is a typical MOLAP approach that uses direct array addressing, where dimension values are accessed via the position or index of their corresponding array locations. Hence, MultiWay cannot perform any value-based reordering as an optimization technique.
- A different approach is developed for the array-based cube construction, as follows:
  - **Partition the array into chunks**
  - **Compute aggregates by visiting cube cells**

PARTITION THE ARRAY INTO CHUNKS

- **Partition the array into chunks**: A chunk is a subcube that is small enough to fit into the memory available for cube computation. Chunking is a method for dividing an *n-dimensional array into small n-dimensional chunks, where each chunk is stored as* an object on disk. The chunks are compressed so as to remove wasted space resulting from empty array cells (i.e., cells that do not contain any valid data, whose cell count is zero). For instance, "*chunkID + offset" can be used as a cell addressing mechanism* to compress a sparse array structure and when searching for cells within a chunk. Such a compression technique is powerful enough to handle sparse cubes, both on disk and in memory.

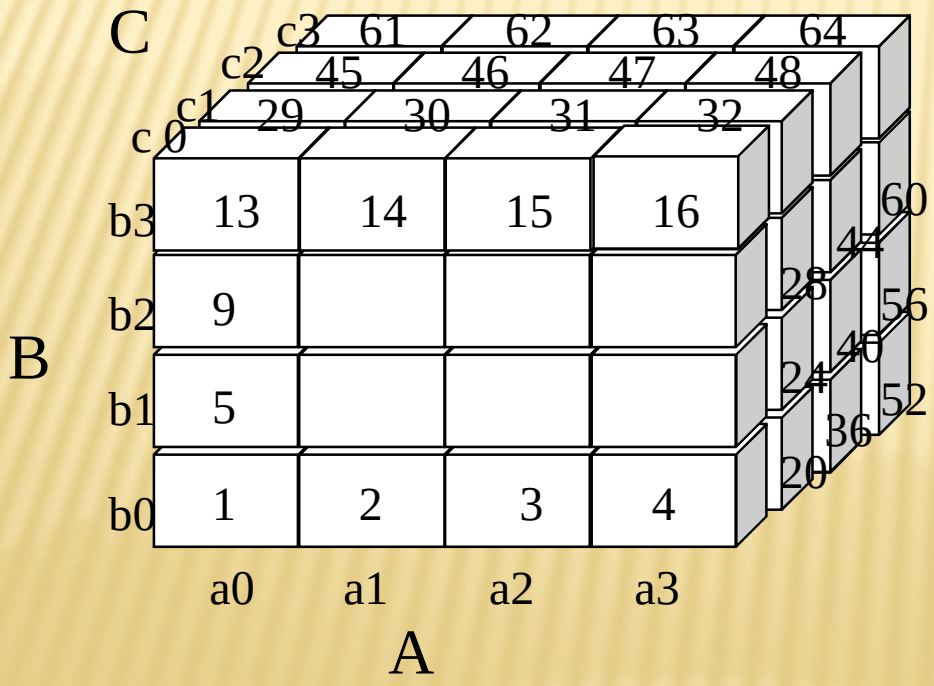
Multi-way Array Aggregation for Cube Computation



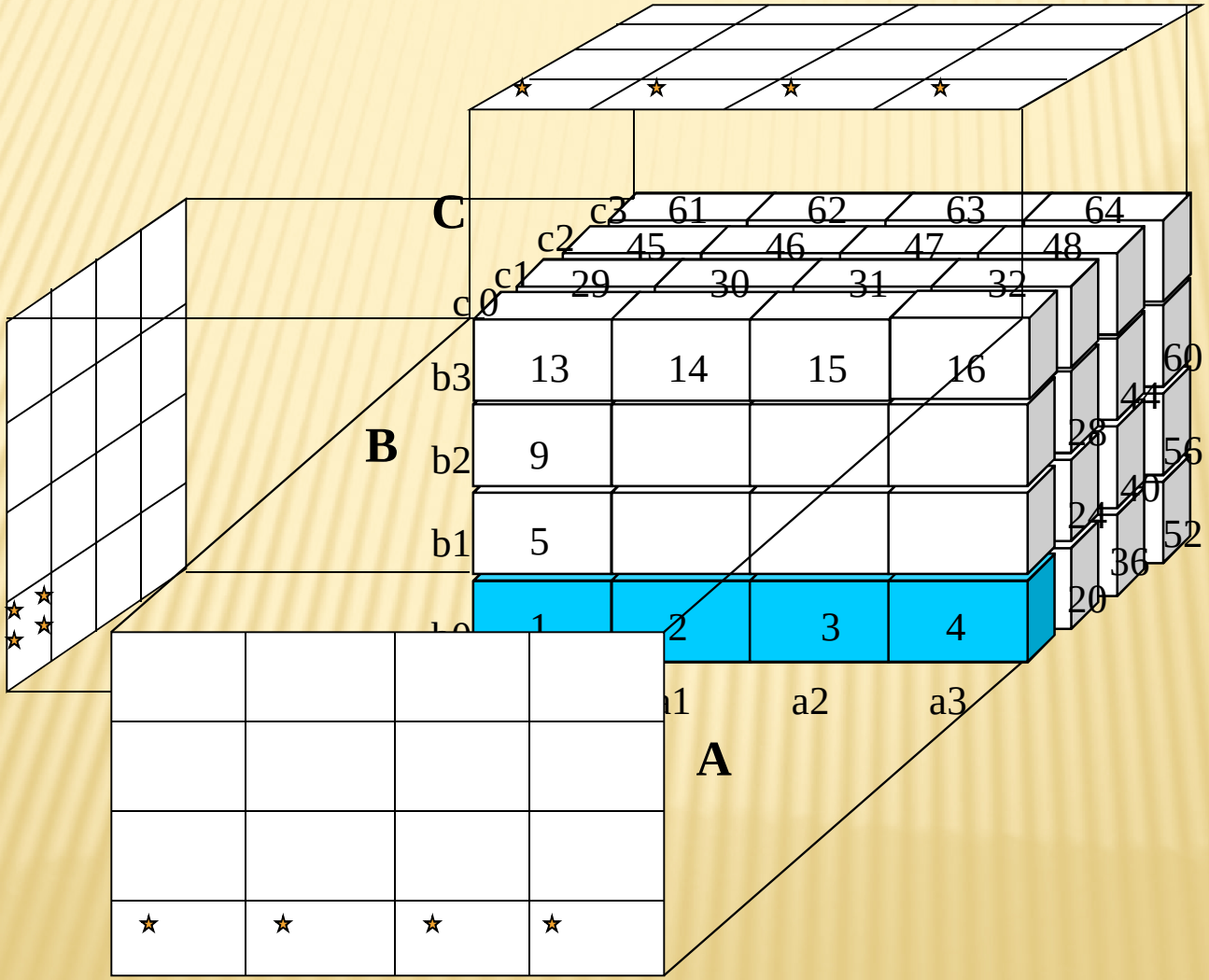
A 3-D array for the dimensions *A*, *B*, and *C*, organized into 64 *chunks*. Each chunk is small enough to fit into the memory available for cube computation.

MULTI-WAY ARRAY AGGREGATION FOR CUBE COMPUTATION

- 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.



MULTI-WAY ARRAY AGGREGATION FOR CUBE COMPUTATION



MULTI-WAY ARRAY AGGREGATION FOR CUBE COMPUTATION