Unit 4
Data Cube Technology

Prepared By

Arjun Singh Saud, Asst. Prof. CDCSIT

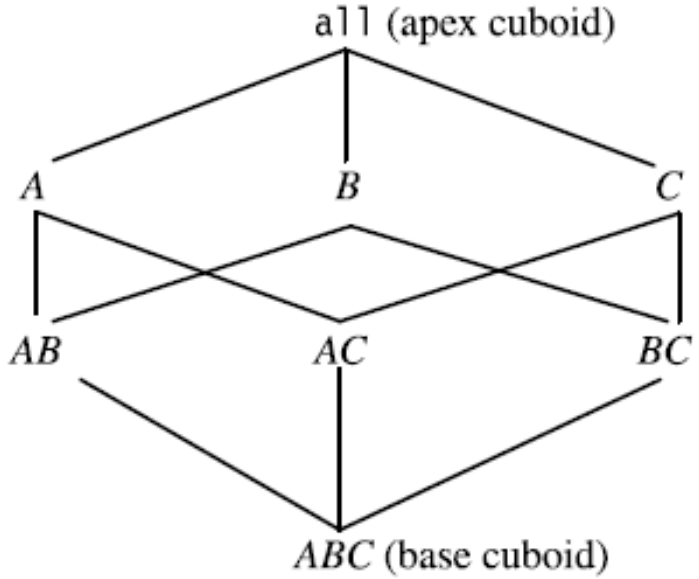
Efficient Method for Data Cube Computation

- 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.
- However, such computation is challenging because it may require substantial computational time and storage space.

Efficient Methods for Data Cube Computation

Cube Materialization

- The figure shows a 3-D data cube for the dimensions *A*, *B*, and *C* for some aggregate measure, *M*. A data cube is a lattice of cuboids.



Efficient Methods for Data Cube Computation

Cube Materialization

- Each cuboid represents a group-by. *ABC* is the base cuboid, containing all three of the dimensions. The base cuboid is the least generalized of all of the cuboids in the data cube.
- The most generalized cuboid is the apex cuboid, commonly represented as **all**. It contains one value.
- To drill down in the data cube, we move from the apex cuboid, downward in the lattice. To roll up, we move from the base cuboid, upward.

Efficient Methods for Data Cube Computation

Cube Materialization

- In order to ensure fast on-line analytical processing, it is sometimes desirable to precompute cubes. The different kinds of cubes that can be precomputed are given below.
- *Full Cube*
- *Iceberg Cube*
- *Close Cube*
- *Shell Cube*

Efficient Methods for Data Cube Computation

Full Cube

- Full cube is the data cube in which all cells or cuboids for the data cube are precomputed.
- This, however, is exponential to the number of dimensions. That is, a data cube of n dimensions contains 2^n cuboids. There are even more cuboids if we consider concept hierarchies for each dimension.
- Thus, precomputation of the full cube can require huge and often excessive amounts of memory.

Efficient Methods for Data Cube Computation

Iceberg Cube

- Instead of computing the full cube, we can compute only a subset of the data cube's cuboids.
- In many situations, it is useful to materialize only those cells in a cuboid (group-by) whose measure value is above some minimum threshold. For example, in a data cube for sales, we may wish to materialize only those cells for which count>10, or only those cells representing sales>$100. Such partially materialized cubes are known as **iceberg cubes**.

Efficient Methods for Data Cube Computation

Iceberg Cube

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

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

Efficient Methods for Data Cube Computation

Base and Aggregate Cells

- A cell in the base cuboid is a **base cell**. A cell from a non-base 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.
- Suppose we have an *n*-dimensional data cube. Let *a*= (*a*1, *a*2, …. , *an*, *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*<*n*) values among {*a*1, *a*2,……, *an)*} are *not* *. If *m*=*n*, then *a* is a base cell; otherwise, it is an aggregate cell.

Efficient Methods for Data Cube Computation

Base and Aggregate Cells

- Consider a data cube with the dimensions month, city, and customer group, and the measure sales.
- (Jan, * , * , 2800) and (*, Chicago, * , 1200) are 1-D cells; (Jan, * , Business, 150) is a 2-D cell; and (Jan, Chicago, Business, 45) is a 3-D cell.
- Here, all base cells are 3-D, whereas 1-D and 2-D cells are aggregate cells.

Efficient Methods for Data Cube Computation

Ancestor and Descendent Cells

- In an n-dimensional data cube, an i-D cell a =($a1$, $a2$, ….. , an, $measures_a$) is an **ancestor** of a j-D cell b= ($b1$, $b2$, : : : , bn, $measures_b$), and b is a **descendant** of a, if and only if
  - $i < j$, and
  - For $1<k< n$, $ak= bk$ whenever $ak\neq *$
- In particular, cell a is called a **parent** of cell b, and b is a **child** of a, if and only if $j= i +1$.

Efficient Methods for Data Cube Computation

Ancestor and Descendent Cells

- 1-D cell *a*=(*Jan*, * ,* ,2800) and 2-D cell *b* = (*Jan*, * , *Business*, 150) are *ancestors* of 3-D cell *c* = (*Jan*, *Chicago*, *Business*, 45); *c* is a *descendant* of both *a* and *b*; *b* is a *parent* of *c*; and *c* is a *child* of *b*.

Efficient Methods for Data Cube Computation

Closed Cube

- A cell, *c*, is a *closed cell* if there exists no cell, *d*, such that *d* is a specialization (descendant) of cell c, and *d* has the same measure value as *c*. A closed cube is a data cube consisting of only closed cells.

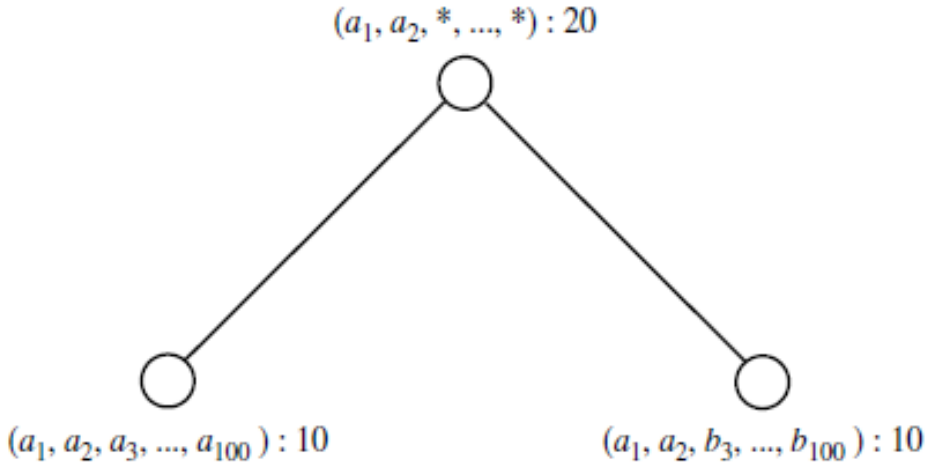


Fig. Three closed cells forming the lattice of a closed cube

Efficient Methods for Data Cube Computation

Shell Cube

- Another strategy for partial materialization is to precompute only the cuboids involving a small number of dimensions, such as 3 to 5. These cuboids form a shell cube.
- Queries on additional combinations of the dimensions will have to be computed on the fly.
- For example, we could compute all cuboids with 3 dimensions or less in an n-dimensional data cube, resulting in a Shell cube of size 3.

General Strategies for Efficient Cube Computation

- The following are general optimization techniques for the efficient computation of data cubes.

Optimization Technique 1: Sorting, hashing, and grouping

- Sorting, hashing, and grouping operations should be applied to the dimension attributes in order to reorder and cluster related tuples.
- In cube computation, aggregation is performed on the tuples (or cells) that share the same set of dimension values.
- For example, to compute total sales by *branch*, *day*, and *item*, it is more efficient to sort tuples or cells by *branch*, and then by *day*, and then group them according to the *item* name.

General Strategies for Efficient Cube Computation

- **Optimization Technique 2: Simultaneous aggregation and caching intermediate results**
- In cube computation, it is efficient to compute higher-level aggregates from previously computed lower-level aggregates, rather than from the base fact table.
- Moreover, simultaneous aggregation from cached intermediate computation results may lead to the reduction of expensive disk I/O operations.
- For example, to compute sales by *branch*, we can use the intermediate results derived from the computation of a lower-level cuboid, such as sales by *branch* and *month*.

General Strategies for Efficient Cube Computation

- **Optimization Technique 3: Aggregation from the smallest child, when there exist multiple child cuboids.**
- When there exist multiple child cuboids, it is usually more efficient to compute the desired parent (i.e., more generalized) cuboid from the smallest, previously computed child cuboid.
- For example, to compute a sales cuboid, C_{branch}, when there exist two previously computed cuboids, $C_{\{branch,year\}}$ and $C_{\{branch,item\}}$, it is obviously more efficient to compute C_{branch} from the former than from the latter if there are many more distinct items than distinct years.

General Strategies for Efficient Cube Computation

- **Optimization Technique 4: The Apriori pruning method can be explored to compute iceberg cubes efficiently**
- The Apriori property, in the context of data cubes, states as follows: *If a given cell does not satisfy minimum threshold, then no descendant of the cell will satisfy minimum threshold*. This property can be used to substantially reduce the computation of iceberg cubes.

Attribute Oriented Induction for Data Characterization

- The attribute-oriented induction (AOI) approach to concept description is alternative of the data cube approach.
- Generally, the data cube approach performs off-line aggregation before an OLAP or data mining query is submitted for processing.
- On the other hand, the attribute-oriented induction approach is basically a *query-oriented* and performs on-line data analysis.

Attribute Oriented Induction for Data Characterization

- The general idea of attribute-oriented induction is to first collect the task-relevant data using a database query and then perform generalization based on the examination of the number of distinct values of each attribute in the relevant set of data.
- The generalization is performed by either *attribute removal* or *attribute generalization*. The following examples illustrate the process of attribute-oriented induction.

Attribute Oriented Induction for Data Characterization

Example: Suppose that a user would like to describe the general characteristics of graduate students in the *Big University* database, given the attributes *Name, Gender, Major, BirthPlace, BirthDate, Residence, Phone#*, and *GPA*. A data mining query for this characterization can be expressed in the data mining query language, DMQL, as follows:

Use BigUniversityDB
mine characteristics as "GraduateStudents"
in relevance to Name, Gender, Major, BirthPlace, BirthDate, Residence,
Phone#, GPA
From Student
where Status in "Graduate"

Attribute Oriented Induction for Data Characterization

- The data mining query presented above is transformed into the following relational query for the collection of the task-relevant set of data:

  Use BigUniversityDB

  Select Name, Gender, Major, BirthPlace, BirthDate, Residence, Phone#, GPA

  From student

  Where status in {"M.Sc.", "M.A.", "M.B.A.", "Ph.D."}

- The result of above query is called the (task-relevant) initial working relation. Now that the data are ready for attribute-oriented induction.

Attribute Oriented Induction for Data Characterization

- Attribute removal is based on the following rule
  1. *If there is a large set of distinct values for an attribute of the initial working relation, but there is no generalization operator on the attribute (e.g., there is no concept hierarchy defined for the attribute) then remove the attribute*
  2. *If there is a large set of distinct values for an attribute of the initial working relation, but its higher-level concepts are expressed in terms of other attributes, then the attribute should be removed from the working relation.*

Attribute Oriented Induction for Data Characterization

- Attribute generalization is based on the following rule: *If there is a large set of distinct values for an attribute in the initial working relation, and there exists a set of generalization operators on the attribute, then a generalization operator should be selected and applied to the attribute.*

Mining Class Comparisons: Discriminating between Different Classes

- In many applications, users may not be interested in having a single class (or concept) described or characterized, but rather would prefer to mine a description that compares or distinguishes one class (or concept) from other comparable classes (or concepts).
- Class discrimination or comparison mines descriptions that distinguish a target class from its contrasting classes.
- The target and contrasting classes must be *comparable* in the sense that they share similar dimensions and attributes. For example, the three classes, *person, address*, and *item*, are not comparable. However, the sales in the last three years are comparable classes, and so are computer science students versus physics students.

Mining Class Comparisons: Discriminating between Different Classes

- *How is class comparison performed?"* In general, the procedure is as follows:

1. **Data Collection**: The set of relevant data in the database is collected by query processing and is partitioned respectively into a *target class* and one or a set of *contrasting class(es)*.
2. **Dimension Relevance Analysis**: If there are many dimensions, then dimension relevance analysis should be performed on these classes to select only the highly relevant dimensions for further analysis.

Mining Class Comparisons: Discriminating between Different Classes

3. **Synchronous Generalization**: Generalization is performed on the target class and contrasting class(es) to the level controlled by a user- or expert-specified dimension threshold.
4. **Presentation of the Derived Comparison**: The resulting class comparison description can be visualized in the form of tables, graphs, and rules. This presentation usually includes a "contrasting" measure such as count% (percentage count). The user can adjust the comparison description by applying drill-down, roll-up, and other OLAP operations to the target and contrasting classes, as desired.