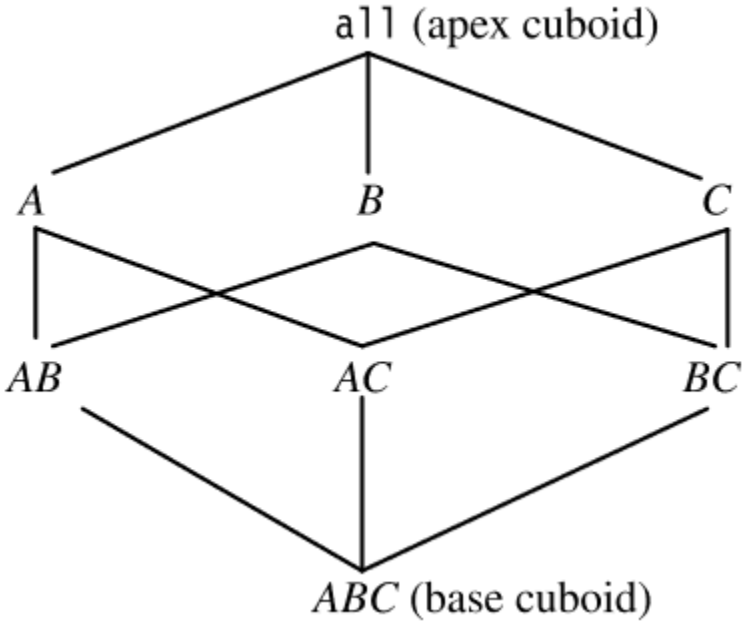
Efficient method for data cube computation

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.



This figure shows a 3-D data cube for the dimensions A, B, and C, and an aggregate measure, M. A data cube is a lattice of cuboids. Each cuboid represents a group-by. ABC is the base cuboid, containing all three of the dimensions. Here, the aggregate measure, M, is computed for each possible combination of the three 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—it aggregates measure M for all of the tuples stored in the base cuboid.
- 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.

Cube materialization

Full cube

In order to ensure **fast on-line analytical processing**, it is sometimes desirable to **pre-compute** the full cube (i.e., **all the cells of all of the cuboids for a given data cube**). Full cube computation is exponential to the number of dimensions, and also depends on the level of concept hierarchy and cardinality of each dimension which require huge and often excessive amounts of memory.

Nonetheless, full cube computation algorithms are important. Individual cuboids **may be stored on secondary storage** and **accessed when necessary**.

Alternatively, we can use such algorithms to **compute smaller cubes,** consisting of a subset of the given set of dimensions, or a smaller range of possible values for some of the dimensions. In such cases, the **smaller cube is a full cube** for the given subset of dimensions and/or dimension values.

Partial Cube

Partial materialization of data cubes offers an interesting **trade-off between storage space** and **response time** for OLAP. Instead of computing the full cube, we can compute **only a subset** of the data cube's **cuboids**, or **subcubes** consisting of subsets of cells from the various cuboids.

Many cells in a cuboid may actually be of little or no interest to the data analyst.

Iceberg cube

Iceberg Cube is a data cube that stores only those cube cells whose aggregate value is above some minimum support threshold

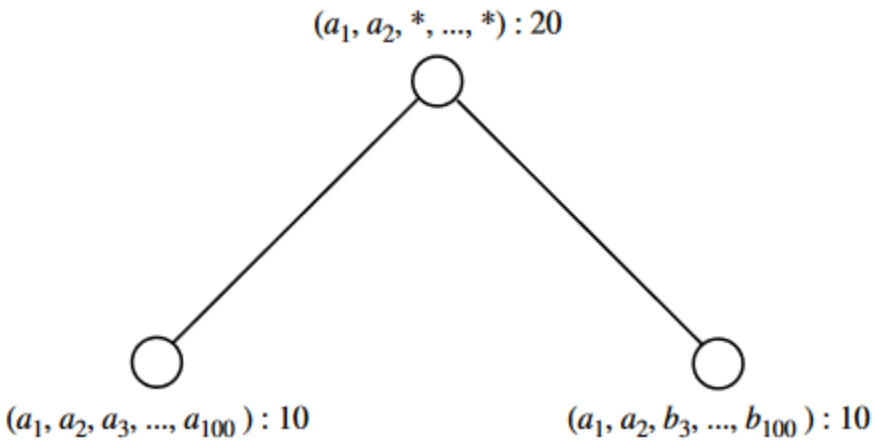
Closed cube

A closed cube is a data cube consisting of only closed cells.

Closed Cell: A cell, c, is a closed cell if there exists no cell, d, such that d is a specialization(descendant) of cell c (that is, 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 the figure below. Other non-closed 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 non-closed cell of the latter. Similarly, we have “(a1, a2, b3, ∗ , . . . , ∗) : 10”.



Shell cube

Shell cube is a cube generated by precomputing the cuboids for only a small number of dimensions of a data cube.

For example, for a 30-dimensional data cube, we may only compute the 5-dimensional cuboids for every possible 5-dimensional combination. Queries on additional combinations of the dimensions are computed on-the-fly.

General strategies for efficient cube computation

In general, there are two basic data structures used for storing cuboids. **Relational tables** are used as the basic data structure for the implementation of relational OLAP (ROLAP), while **multidimensional arrays** are used as the basic data structure in multidimensional OLAP (MOLAP). The following are general optimization techniques for the efficient computation of data cubes for both architectures:

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. Thus it is important to explore sorting, hashing, and grouping operations to access and group such data together to facilitate computation of such aggregates.

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. Efficient implementations of such operations in large data sets have been extensively studied in the database research community. Such implementations can be extended to data 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 day. This technique can be further extended to perform amortized scans (i.e., computing as many cuboids as possible at the same time to amortize disk reads).

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.

Optimization Technique 4: The Apriori pruning method can be explored to compute iceberg cubes efficiently

If a given cell does not satisfy minimum support, then no descendant (i.e., more specialized or detailed version) of the cell will satisfy minimum support either. This property can be used to substantially reduce the computation of iceberg cubes.

Attribute oriented induction for data characterization

Attribute-oriented induction approach is basically a query-oriented, generalization-based, online data analysis technique that implements off-line aggregation prior to OLAP query submission. The general procedure for AOI can be summarized as:

1. **Data focusing:**

   Collect task-relevant data, including dimensions, and the result is the initial working relation

2. **Generalization:**

   Generalization is implemented by attribute removal or attribute generalization

   - **Attribute-removal:**

     Remove attribute A if there is a large set of distinct values for A but

     - there is no generalization operator on A (i.e. no concept hierarchy), or
     - A's higher level concepts are expressed in terms of other attributes

   - **Attribute-generalization:**

     If there is a large set of distinct values for A, and there exists a set of generalization operators on A, then select an operator and generalize A.

3. **Aggregation:**

   Aggregation is implemented by combining identical generalized tuples and accumulating their specific counts. This decreases the size of the generalized data set.

4. **Presentation:**

   The resulting generalized association can be mapped into several forms for presentation to the user, including tables, charts, crosstabs, graphs or rules.

Attribute generalization control

The control of how high an attribute should be generalized is called **attribute generalization control**. If the attribute is generalized "too high," it may lead to over-generalization, and the

resulting rules may not be very informative. On the other hand, if the attribute is not generalized to a "sufficiently high level," then under-generalization may result, where the rules obtained may not be informative either. Thus, a balance should be attained in attribute-oriented generalization. There are many possible ways to control a generalization process but the two most common approaches are:

attribute generalization threshold control: either set one generalization threshold for all of the attributes, or set one threshold for each attribute. If the number of distinct values in an attribute is greater than the **attribute threshold**, further attribute removal or attribute generalization should be performed. Data mining systems typically have a default attribute threshold value generally ranging from 2 to 8 and should allow experts and users to modify the threshold values as well.

generalized relation threshold control: sets a threshold for the generalized relation. If the number of (distinct) tuples in the generalized relation is greater than the threshold, further generalization should be performed. Otherwise, no further generalization should be performed. The threshold value for generalized relation control typically ranges from 10 to 30.

Mining class comparison or discriminating between different classes

Rather than characterizing a single class, we sometimes might be interested in mining 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** that share **similar dimensions and attributes**.

For example:

- three classes, person, address, and item, are not comparable.
- the sales in the last three years are comparable.
- computer science students and physics students are comparable.

In general class comparison is performed as:

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. Correlation or entropy-based measures can be used for this step.

*entropy: randomness of the data set. More the entropy more will be the scatterness in the data set.

3. **Synchronous generalization:**

Generalization is performed on the target class to the level controlled by a user- or expert-specified **dimension threshold**, which results in a **prime target class relation**. The concepts in the contrasting class(es) are generalized to the same level as those in the prime target class relation, forming the **prime contrasting class(es) relation**. This allows simultaneous comparison between classes at the **same levels of abstraction**.

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% **that reflects the comparison between the target and contrasting classes**. 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.

For example:

Suppose that you would like to compare (count%) the general properties between the graduate students and the undergraduate students at Big University, given the attributes name, gender, major, birth place, birth date, residence, phone#, and gpa.

This mining process can be expressed in terms of DMQL as:

> ***use*** *Big University DB*
>
> ***mine comparison as*** *"grad vs undergrad students"*
>
> ***in relevance to*** *name, gender, major, birth place, birth date, residence, phone#, gpa*
>
> ***for*** *"graduate students"*
>
> ***where*** *status in "graduate"*
>
> ***versus*** *"undergraduate students"*
>
> ***where*** *status in "undergraduate"*
>
> ***analyze*** *count%*
>
> ***from*** *student*

Step 1:

The query is transformed into two relational queries that collect two sets of task-relevant data: one for the **initial target class working relation**, and the other for the **initial contrasting class working relation**.

Initial Target Class Working Relation

name	*gender*	*major*	*birth_place*	*birth_date*	*residence*	*phone#*	*gpa*
Jim Woodman	M	CS	Vancouver, BC, Canada	8-12-76	3511 Main St., Richmond	687-4598	3.67
Scott Lachance	M	CS	Montreal, Que, Canada	28-7-75	345 1st Ave., Vancouver	253-9106	3.70
Laura Lee	F	Physics	Seattle, WA, USA	25-8-70	125 Austin Ave., Burnaby	420-5232	3.83
...	...	...	...	...	...	...	...

Initial Contrasting Class Working Relation

name	*gender*	*major*	*birth_place*	*birth_date*	*residence*	*phone#*	*gpa*
Bob Schumann	M	Chemistry	Calgary, Alt, Canada	10-1-78	2642 Halifax St., Burnaby	294-4291	2.96
Amy Eau	F	Biology	Golden, BC, Canada	30-3-76	463 Sunset Cres., Vancouver	681-5417	3.52
...	...	...	...	...	...	...	...

Step 2:

Dimension relevance analysis can be performed, when necessary, on the two classes of data. After this analysis, irrelevant or weakly relevant dimensions, such as name, gender, birth place, residence, and phone#, are removed from the resulting classes.

major	*birth_date*	*gpa*
CS	8-12-76	3.67
CS	28-7-75	3.70
Physics	25-8-70	3.83
...	...	...

major	*birth_date*	*gpa*
Chemistry	10-1-78	2.96
Biology	30-3-76	3.52
...	...	...

Step 3:

Synchronous Generalization is performed on the target class to the levels controlled by user- or expert-specified dimension thresholds, forming the prime target class relation. The contrasting class is generalized to the same levels as those in the prime target class relation, forming the prime contrasting class(es)

Prime generalized relation for the *target class* (graduate students)

major	*age_range*	*gpa*	count%
Science	21...25	good	5.53%
Science	26...30	good	5.02%
Science	over_30	very_good	5.86%
...	...	...	...
Business	over_30	excellent	4.68%

Prime generalized relation for the *contrasting class* (undergraduate students)

major	*age_range*	*gpa*	count%
Science	16...20	fair	5.53%
Science	16...20	good	4.53%
...	...	...	...
Science	26...30	good	2.32%
...	...	...	...
Business	over_30	excellent	0.68%

Step 4:

The results can be **visualized** in the form of tables, graphs, and rules. With exception of logic rules, bar charts, crosstabs, pie charts, cubes, tables, curves or other visualization technique are similar to data characterization.