Unit-4
Data Cube Technology

Efficient Method for Data Cube Computation

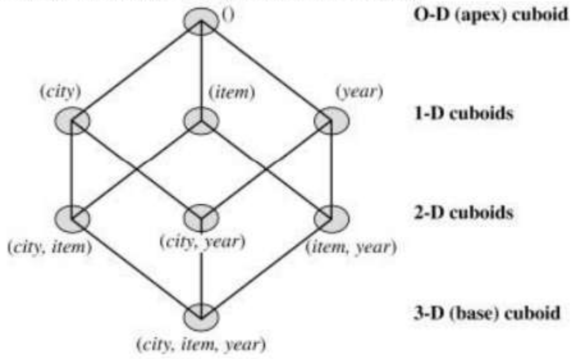
Data cube computation is an essential task in data warehouse implementation. The pre-computation of all or part of a data cube can greatly reduce the response time and enhance the performance of on-line analytical processing. At the core of multidimensional data analysis is the efficient computation of aggregations across many sets of dimensions. In SQL terms, these aggregations are referred to as group-by's. Each group-by can be represented by a cuboid, where the set of group-by's forms a lattice of cuboids defining a data cube.

A major challenge related to pre-computation would be time and storage space if all the cuboids in the data cube are computed, especially when the cube has many dimensions.

The Compute Cube Operator

The ***compute cube operator*** computes aggregates over all subsets of the dimensions specified in the operation.

Following figure shows a 3-D data cube for the dimensions *city, item, and year*, and an aggregate measure, M. A data cube is a lattice of cuboids.



Total no. of cuboids = 2^3

Each cuboid represents a group-by. *(city, item, year)* 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.

Consider the following 2 cases for n-dimensional cube

- ***Case 1: Dimensions have no hierarchies***
  - Then the total number of cuboids computed for a n-dimensional cube = 2^n
- ***Case 2: Dimensions have hierarchies*** (*E.g. Hierarchy "day<month<quarter<year" for time*)
  - Then the total number of cuboids computed for a n-dimensional cube = $\prod_{i=1}^{n}(L_i + 1)$
    Where L_i is the number of levels associated with dimension i. One is added to L_i to include the *virtual* top level, *all*.

Q. If the cube has 10 dimensions and each dimension has 4 levels, what will be the number of cuboids generated?

Solution

Here n=10 Li=4 for i=1, 2……,10

Total number of cuboids = $5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 = 5^{10} \approx 9.8 \times 10^6$

Choices for Data Cube Materialization (Pre-computation)

There are three choices for data cube materialization given a base cuboid:

1. ***No materialization:*** Do not pre-compute any of the "non-base" cuboids. This leads to computing expensive multidimensional aggregates on-the-fly, which can be extremely slow.
2. ***Full materialization:*** Pre-compute all of the cuboids. The resulting lattice of computed cuboids is referred to as the full cube. This choice typically requires huge amounts of memory space in order to store all of the precomputed cuboids.
3. ***Partial materialization:*** Selectively compute a proper subset of the whole set of possible cuboids. Alternatively, we may compute a subset of the cube, which contains only those cells that satisfy some user-specified criterion, such as where the tuple count of each cell is above some threshold.

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.

Cube Materialization

In order to ensure fast on-line analytical processing, it is sometimes desirable to precompute cubes. Different cube materialization include: ***Full cube, Iceberg cube, Closed cube*** *and* ***Shell cube.***

1. ***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.

2. ***Iceberg Cube***

   An Iceberg-Cube contains only those cells of the data cube that meet an aggregate condition. It is called an Iceberg-Cube because it contains only some of the cells of the full cube, like the tip of an iceberg. The aggregate condition could be, for example, minimum support or a lower bound on average, min or max. The purpose of the Iceberg-Cube is to identify and compute only those values that will most likely be required for decision support queries. The aggregate condition specifies which cube values are more meaningful and should therefore be stored.

3. ***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.

4. ***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 Cube Computation

The following are ***general optimization techniques*** for the efficient computation of data cubes:

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. These operation facilitate aggregation, i.e. computation of the cells that share the same set of dimension values. These technique can also perform:
   - *Shared-sorts:* Sharing sorting costs across multiple cuboids when sort-based methods are used.
   - *Share-partitions:* Sharing partitioning costs across multiple cuboids when hash based algorithms are used.

   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.*

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

   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.*

3. ***Aggregation from smallest child when there exist multiple child cuboid***

   If a parent 'cuboid' has more than one child, it is efficient to compute it from the smallest previously computed child 'cuboid'.

   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.

4. ***The Apriori pruning method can be explored to compute iceberg cube efficiently***

   The Apriori property, in the context of data cubes, states as follows: *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

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.

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.

Basic Principles of Attribute Oriented Induction

- ***Data focusing:*** Analyzing task-relevant data, including dimensions, and the result is the initial relation.
- ***Attribute-removal:*** To remove attribute A if there is a large set of distinct values for A but
  1. there is no generalization operator on A, or
  2. 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.
- ***Attribute-threshold control:*** Typical 2-8, specified/default.
- ***Generalized relation threshold control (10-30):*** To control the final relation/rule size.

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, birth place, birth date, residence, phone#, and gpa. A data mining query for this characterization can be expressed in the data mining query language, DMQL, as follows:

use *Big University DB*
mine *characteristics as "Science Students"*
in relevance to *name, gender, major, birth place, birth date, residence, phone#, gpa*
from *Student*
where *status in "graduate"*

Attribute Removal and Generalization:

Initial 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., Richmond	253-9106	3.70
Laura Lee	F	Physics	Seattle, WA, USA	25-8-70	125 Austin Ave., Burnaby	420-5232	3.83
...	...	...	...	...	...	...	...
Removed	Retained	Sci,Eng, Bus	Country	Age range	City	Removed	Excl, VG,..

Prime Generalized Relation

Gender	Major	Birth_region	Age_range	Residence	GPA	Count
M	Science	Canada	20-25	Richmond	Very-good	16
F	Science	Foreign	25-30	Burnaby	Excellent	22
...	...	...	...	...	...	...

- ***Name:*** Since there are a large number of distinct values for name and there is no generalization operation defined on it, this attribute is removed
- ***Gender:*** Since there are only two distinct values for gender, this attribute is retained and no generalization is performed on it
- ***Major:*** Suppose that a concept hierarchy has been defined that allows the attribute major to be generalized to the values {arts & science, engineering, business}
- ***Birth place:*** This attribute has a large number of distinct values; therefore, we would like to generalize it. Suppose that a concept hierarchy exists for birth place, defined as "city < province or state < country"

- ***Birth date:*** Suppose that a hierarchy exists that can generalize birth date to age, and age to age range, and that the number of age ranges
- ***Residence :*** Suppose that residence is defined by the attributes number, street, residence city, residence province or state, and residence country
- ***Phone#:*** As with the attribute name above, this attribute contains too many distinct values and should therefore be removed in generalization
- ***Gpa:*** Suppose that a concept hierarchy exists for gpa that groups values for grade point average into numerical intervals like 3.75–4.0, 3.5–3.75,. . ., which in turn are grouped into descriptive values, such as excellent, very good,. . .. The attribute can therefore be generalized

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 (hereafter referred to as class comparison) mines descriptions that distinguish a target class from its contrasting classes. Notice that 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.

The ***procedures for class comparison*** performed as follows:

1. ***Data Collection:*** 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:*** select only the highly relevant dimensions for further analysis.
3. ***Synchronous Generalization:*** Generalization is performed on the target class to the level controlled by a user- or expert-specified dimension threshold.
4. ***Presentation of the Derived Comparison:*** Results can be visualized in the form of tables, graphs.

Example

Suppose that we would like to compare 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. For this, the DMQL query would be:

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*

1. ***First Phase (Task Relevant Data):***

Initial working relations: the *target class* (graduate students)

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 working relations: the *contrasting class* (undergraduate students)

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
...	...	...	...	...	...	...	...

2. ***Second Phase (Dimension Relevance Analysis):***

Only the highly relevant attributes are included in the subsequent analysis. Irrelevant or weakly relevant dimensions, such as ***name, gender, birth place, residence, and phone#,*** are removed from the resulting classes.

3. ***Third Phase (Synchronous Generalization):***

Prime generalized relation for the target class: Graduate students

Major	Age_range	Gpa	Count%
Science	20-25	Good	5.53%
Science	26-30	Good	2.32%
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	15-20	Fair	5.53%
Science	15-20	Good	4.53%
...	...	...	...
Science	26-30	Good	5.02%
...	...	...	...
Business	Over_30	Excellent	0.68%

4. ***Fourth Phase (Result Presentation):***

Resulting class comparison is presented in the form of tables, graphs, or contrasting measure (such as count%) that compares between the target class and the contrasting class. For example, 5.02% of the graduate students majoring in Science are between 26 and 30 years of age and have a "good" GPA, while only 2.32% of undergraduates have these same characteristics

Please let me know if I missed anything or anything is incorrect.

poudeljayanta99@gmail.com