2

OLAP and Data Cubes

This chapter reviews On-line Analytical Processing (OLAP) in Section 2.1 and data cubes in Section 2.2.

2.1 OLAP

Coined by Codd et. al [18] in 1993, OLAP stands for On-Line Analytical Processing. The concept has its root in earlier products such as the IRI Express, the Comshare system, and the Essbase system [67]. Unlike statistical databases which usually store census data and economic data, OLAP is mainly used for analyzing business data collected from daily transactions, such as sales data and health care data [65]. The main purpose of an OLAP system is to enable analysts to construct a mental image about the underlying data by exploring it from different perspectives, at different level of generalizations, and in an interactive manner.

As a component of decision support systems, OLAP interacts with other components, such as data warehouse and data mining, to assist analysts in making business decisions. A data warehouse usually stores data collected from multiple data sources, such as transactional databases throughout an organization. The data are cleaned and transformed to a common consistent format before they are stored in the data warehouse. Subsets of the data in a data warehouse can be extracted as data marts to meet the specific requirements of an organizational division. Unlike in transactional databases where data are constantly updated, typically the data stored in a data warehouse are refreshed from data sources only periodically.

OLAP and data mining both allow analysts to discover novel knowledge about the data stored in a data warehouse. Data mining algorithms automatically produce knowledge in a pre-defined form, such as association rule or classification. OLAP does not directly generate such knowledge, but instead relies on human analysts to observe it by interpreting the query results. On the other hand, OLAP is more flexible than data mining in the sense that

analysts may obtain all kinds of patterns and trends rather than only knowledge of fixed forms. OLAP and data mining can also be combined to enable analysts in obtaining data mining results from different portion of the data and at different level of generalization [39].

In a typical OLAP session, the analyst poses aggregation queries about underlying data. The OLAP system can usually return the result in a matter of seconds, even though the query may involve a large number of records. Based on the results, the analysts may decide to *roll up* to coarser-grained data so they can observe global patterns and trends. Upon observing an exception to any established pattern, the analysts may *drill down* to finer-grained data with more details to catch the outliers. Such a process is repeated in different portions of the data by *slicing* or *dicing* the data, until a satisfactory mental image of the data has been constructed.

The requirements on OLAP systems have been defined differently, such as the FASMI (Fast Analysis of Shared Multidimensional Information) test [58] and the Codd rules [18]. Some of the requirements are unique to OLAP. First, to make OLAP analysis an interactive process, the OLAP system must be highly efficient in answering queries. OLAP systems usually rely on extensive pre-computations, indexing, and specialized storage to improve the performance. Second, to allow analysts to explore the data from different perspectives and at different level of generalization, OLAP organizes and generalizes data along multiple dimensions and dimension hierarchies. The data cube model we shall address shortly is one of the most popular abstract models for this purpose.

The data to be analyzed by OLAP are usually stored based on the relational model in the backend data warehouse. The data are organized based on a *star schema*. Figure 2.1 shows an example of star schema. It has a *fact table* $(timeID, orgID, commission)$, where the first two attributes $timeID$ and $orgID$ are called *dimenions*, and *commission* is called a *measure*. Each dimenion has a *dimension table* associated with it, indicating a dimension hierarchy. The dimension tables may contain redundancy, which can be removed by splitting each dimension table into multiple tables, one per attribute in the dimension table. The result is called a *snowflake schema*, as illustrated in Figure 2.2.

Fig. 2.1. An Example of Star Schema

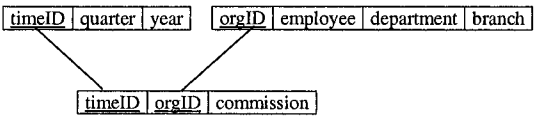
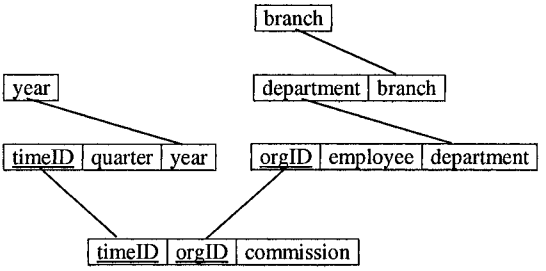


Fig. 2.2. An Example of Snowflake Schema



Popular architectures of OLAP systems include *ROLAP* (relational OLAP) and *MOLAP* (multidimensional OLAP). ROLAP provides a front-end tool that translates multidimensional queries into corresponding SQL queries to be processed by the relational backend. ROLAP is thus light weight and scalable to large data sets, whereas its performance is constrained because optimization techniques in the relational backend are typically not designed for multidimensional queries. MOLAP does not rely on the relational model but instead materializes the multidimensional views. MOLAP can thus provide better performance with the materialized and optimized multidimensional views. However, MOLAP demands substantial storage for materializing the views and is usually not scalable to large datasets due to the multidimensional explosion problem [57]. Using MOLAP for dense parts of the data and ROLAP for the others leads to a hybrid architecture, namely, the *HOLAP* or hybrid OLAP.

2.2 Data Cube

Data cube was proposed as a SQL operator to support common OLAP tasks like histograms (that is, aggregation over computed categories) and subtotals [37]. Even though such tasks are usually possible with standard SQL queries, the queries may become very complex. For example, Table 2.1 shows a simple relation *comm*, where employee, quarter, and location are the three dimensions, and the commission is a measure. We call *comm* a *base relation* since it contains the ground facts to be analyzed.

Suppose analysts are interested in the sub-total commissions in the two-dimensional cross tabular in Table 2.2 (other possible ways of visually representing such subtotals are discussed in [37]). The inner tabular includes the quarterly commission for each employee; below the inner tabular are the total commissions for each employee; to the right are the total commissions in each

employee	quarter	location	commission
Alice	Q1	Domestic	800
Alice	Q1	International	200
Bob	Q1	Domestic	500
Mary	Q1	Domestic	1200
Mary	Q1	International	800
Bob	Q2	International	1500
Mary	Q2	Domestic	500
Jim	Q2	Domestic	1000

Table 2.1. The Base Relation *comm*

year; at the right bottom corner is the total commission of all employees in the two years.

	Alice	Bob	Mary	Jim	total(ALL)
Q1	1000	500	2000		3500
Q2		1500	500	1000	3000
total(ALL)	1000	2000	2500	1000	6500

Table 2.2. A Two-dimensional Cross Table

The sub-totals in Table 2.2 can be computed using SQL queries. However, we need to union four `GROUP BY` queries as follows:

```
SELECT employee, quarter, SUM(commission)
FROM   comm
GROUP BY employee,quarter
UNION
SELECT 'ALL', quarter, SUM(commission)
FROM   comm
GROUP BY quarter
UNION
SELECT employee, 'ALL', SUM(commission)
FROM   comm
GROUP BY employee
UNION
SELECT 'ALL','ALL', SUM(commission)
FROM   comm
```

The number of needed unions is exponential in the number of dimensions. A complex query may result in many scans of the base table, leading to poor performance. Because such sub-totals are very common in OLAP queries, it is

desired to define a new operator for the collection of such sub-totals, namely, *data cube.*

A data cube is essentially the generalization of the cross tabular illustrated in Table 2.2. The generalization happens in several perspectives. First, a data cube can be n dimensional. Table 2.3 shows a three-dimensional data cube built from this new base table. Based on the ALL values, the data cube is divided into eight parts, namely, *cuboids.* The first cuboid is a three-dimensional cube, usually called the *core* cuboid. The next three cuboids have one ALL value and are the two-dimensional planes. The next three are the one-dimensional lines. The last cuboid has a single value and is a zero-dimensional point.

The second perspective of the generalization is the aggregation function. The aggregation discussed so far is SUM. In general any aggregation function, including customized ones, can be used to construct a data cube. Those functions can be classified into three categories, the *distributive*, the *algebraic*, and the *holistic.* Let I be a set of values, and $P(I) = \{p_1, p_2, \ldots, p_n\}$ be any partition on I. Then an aggregation function $F()$ is distributive, if there exists a function $G()$ such that $F(I) = G(\{F(p_i) : 1 \leq i \leq n\})$. It is straightforward to verify that SUM, COUNT, MIN, and MAX are distributive.

By generalizing the function $G()$ into one that returns an m-vector, the algebraic aggregations have a similar property like that of the distributive ones, that is $F(I) = G(\{F(p_i) : 1 \leq i \leq n\})$. For example, Let $G() =< SUM(), COUNT() >$, then $AVERAGE$ is clearly an algebraic function. However, a holistic function like MEDIAN cannot be evaluated on $P(I)$ with any $G()$ that returns a vector of constant degree. The significance in distinguishing those three types of aggregation function lies in the difficulty of computing a data cube. Because the cuboids in a data cube form a hierarchy of aggregation, a cuboid can be more easily computed from other cuboids for distributive and algebraic functions. However, for a holistic function, any cuboid must be computed directly from the base table.

The third perspective of the generalization is the dimension hierarchy. For the data cubes shown in Table 2.2 and Table 2.3, each dimension is a two-level pure hierarchy. For example, the employee dimension has basically two attributes, employee and ALL (ALL should be regarded as both an attribute and its only value). In general, each dimension can have many attributes, such as the example in Figure 2.1. The attributes of a dimension may form a lattice instead of a pure hierarchy, such as day, week, month, and year (week and month are incomparable). The attribute in the base table is the lower bound of the lattice, and ALL (regarded as an attribute) is the upper bound. The product of the dimension lattices is still a lattice, as illustrated in Figure 2.3. This lattice is essentially the schema of a data cube.

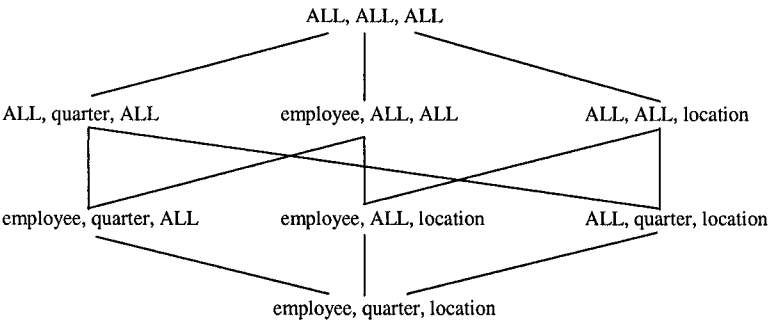
The lattice structure has played an important role in many perspectives of data cubes. For example, materializing the whole data cube usually incurs prohibitive costs in computation and storage. Moreover, materializing a cuboid does not always bring much benefits to answering queries. In Figure 2.3, the

employee	quarter	location	commission
Alice	Q1	Domestic	800
Alice	Q1	International	200
Bob	Q1	Domestic	500
Mary	Q1	Domestic	1200
Mary	Q1	International	800
Bob	Q2	International	1500
Mary	Q2	Domestic	500
Jim	Q2	Domestic	1000
Alice	Q1	ALL	1000
Bob	Q1	ALL	500
Mary	Q1	ALL	2000
Bob	Q2	ALL	1500
Mary	Q2	ALL	500
Jim	Q2	ALL	1000
Alice	ALL	Domestic	800
Alice	ALL	International	200
Bob	ALL	Domestic	500
Mary	ALL	Domestic	1700
Mary	ALL	International	800
Bob	ALL	International	1500
Jim	ALL	Domestic	1000
ALL	Q1	Domestic	2500
ALL	Q1	International	1000
ALL	Q2	International	1500
ALL	Q2	Domestic	1500
Alice	ALL	ALL	1000
Bob	ALL	ALL	2000
Mary	ALL	ALL	2500
Jim	ALL	ALL	1000
ALL	Q1	ALL	3300
ALL	Q2	ALL	3000
ALL	ALL	Domestic	4000
ALL	ALL	International	2500
ALL	ALL	ALL	6500

Table 2.3. A Three-Dimensional Data Cube

core cuboid must be materialized because it cannot be computed from others. However, if any of the cuboids with one ALL value has a comparable size to the core cuboid, then it may not need to be materialized, because answering a query using that cuboid incurs similar costs as using the core cuboid instead (the cost of answering a query is roughly proportional to the size of the cuboid being used). Greedy algorithms have been proposed to find optimal materialization of a data cube under resource constraints [40].

Fig. 2.3. The Lattice Structure of Data Cube



Even if the whole data cube needs to be computed, the lattice structure can help to improve the performance of such computations. For example, if the computation is based on sorting the records, then the core cuboid can be sorted in three different ways (by any two of the three attributes). Each choice will simplify the computation of one of the three cuboids with one ALL value because that cuboid can be computed without additional sorting. However, computing the other two cuboids will require the core cuboid be re-sorted. Based on estimated costs, algorithms exist to make the optimal choice in sorting each cuboid, and those choices can be linked to form pipelines of computation so the needs for re-sorting cuboids can be reduced [2].