DATA MINING

MODULE 2

DATA WAREHOUSE AND OLAP TECHNOLOGY FOR DATA MINING: DATA WAREHOUSE

- Data warehousing provides architectures and tools for business executives to systematically organize, understand, and use their data to make strategic decisions.
- Loosely speaking, a data warehouse refers to a data repository that is maintained separately from an organization's operational databases. Data warehouse systems allow for integration of a variety of application systems. They support information processing by providing a solid platform of consolidated historic data for analysis.
- According to William H. Inmon, a leading architect in the construction of data warehouse systems, "A data warehouse is a subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management's decision making process".
- This short but comprehensive definition presents the major features of a data warehouse. The four keywords—subject-oriented, integrated, time-variant, and nonvolatile—distinguish data warehouses from other data repository systems, such as relational database systems, transaction processing systems, and file systems.

- Subject-oriented: A data warehouse is organized around major subjects such as customer, supplier, product, and sales. Rather than concentrating on the day-to-day operations and transaction processing of an organization, a data warehouse focuses on the modeling and analysis of data for decision makers. Hence, data warehouses typically provide a simple and concise view of particular subject issues by excluding data that are not useful in the decision support process.
- Integrated: A data warehouse is usually constructed by integrating multiple heterogeneous sources, such as relational databases, flat files, and online transaction records. Data cleaning and data integration techniques are applied to ensure consistency in naming conventions, encoding structures, attribute measures, and so on.
- Time-variant: Data are stored to provide information from an historic perspective (e.g., the past 5–10 years). Every key structure in the data warehouse contains, either implicitly or explicitly, a time element.
- Nonvolatile: A data warehouse is always a physically separate store of data transformed from the application data found in the operational environment. Due to this separation, a data warehouse does not require transaction processing, recovery, and concurrency control mechanisms. It usually requires only two operations in data accessing: initial loading of data and access of data.

Table 4.1 Comparison of OLTP and OLAP Systems

Feature	*OLTP*	*OLAP*
Characteristic	operational processing	informational processing
Orientation	transaction	analysis
User	clerk, DBA, database professional	knowledge worker (e.g., manager, executive, analyst)
Function	day-to-day operations	long-term informational requirements decision support
DB design	ER-based, application-oriented	star/snowflake, subject-oriented
Data	current, guaranteed up-to-date	historic, accuracy maintained over time
Summarization	primitive, highly detailed	summarized, consolidated
View	detailed, flat relational	summarized, multidimensional
Unit of work	short, simple transaction	complex query
Access	read/write	mostly read
Focus	data in	information out
Operations	index/hash on primary key	lots of scans
Number of records accessed	tens	millions
Number of users	thousands	hundreds
DB size	GB to high-order GB	$\geq$ TB
Priority	high performance, high availability	high flexibility, end-user autonomy
Metric	transaction throughput	query throughput, response time

MULTIDIMENSIONAL DATA MODEL

- A data cube for summarized sales data of AllElectronics is presented in Figure 1.7(a).
- The cube has three dimensions: address (with city values Chicago, New York, Toronto, Vancouver), time (with quarter values Q1, Q2, Q3, Q4), and item(with item type values home entertainment, computer, phone, security).
- The aggregate value stored in each cell of the cube is sales_amount (in thousands). For example, the total sales for the first quarter, Q1, for the items related to security systems in Vancouver is $400,000, as stored in cell (Vancouver, Q1, security).
- Additional cubes may be used to store aggregate sums over each dimension, corresponding to the aggregate values obtained using different SQL group-bys (e.g., the total sales amount per city and quarter, or per city and item, or per quarter and item, or per each individual dimension).

- By providing multidimensional data views and the precomputation of summarized data, data warehouse systems can provide inherent support for OLAP (Online analytical processing).
- Online analytical processing operations make use of background knowledge regarding the domain of the data being studied to allow the presentation of data at different levels of abstraction.
- Such operations accommodate different user viewpoints.
- Examples of OLAP operations include drill-down and roll-up, which allow the user to view the data at differing degrees of summarization, as illustrated in Figure 1.7(b).
- For instance, we can drill down on sales data summarized by quarter to see data summarized by month. Similarly, we can roll up on sales data summarized by city to view data summarized by country.
- Multidimensional data mining (also called exploratory multidimensional data mining) performs data mining in multidimensional space in an OLAP style. That is, it allows the exploration of multiple combinations of dimensions at varying levels of granularity in data mining, and thus has greater potential for discovering interesting patterns representing knowledge.

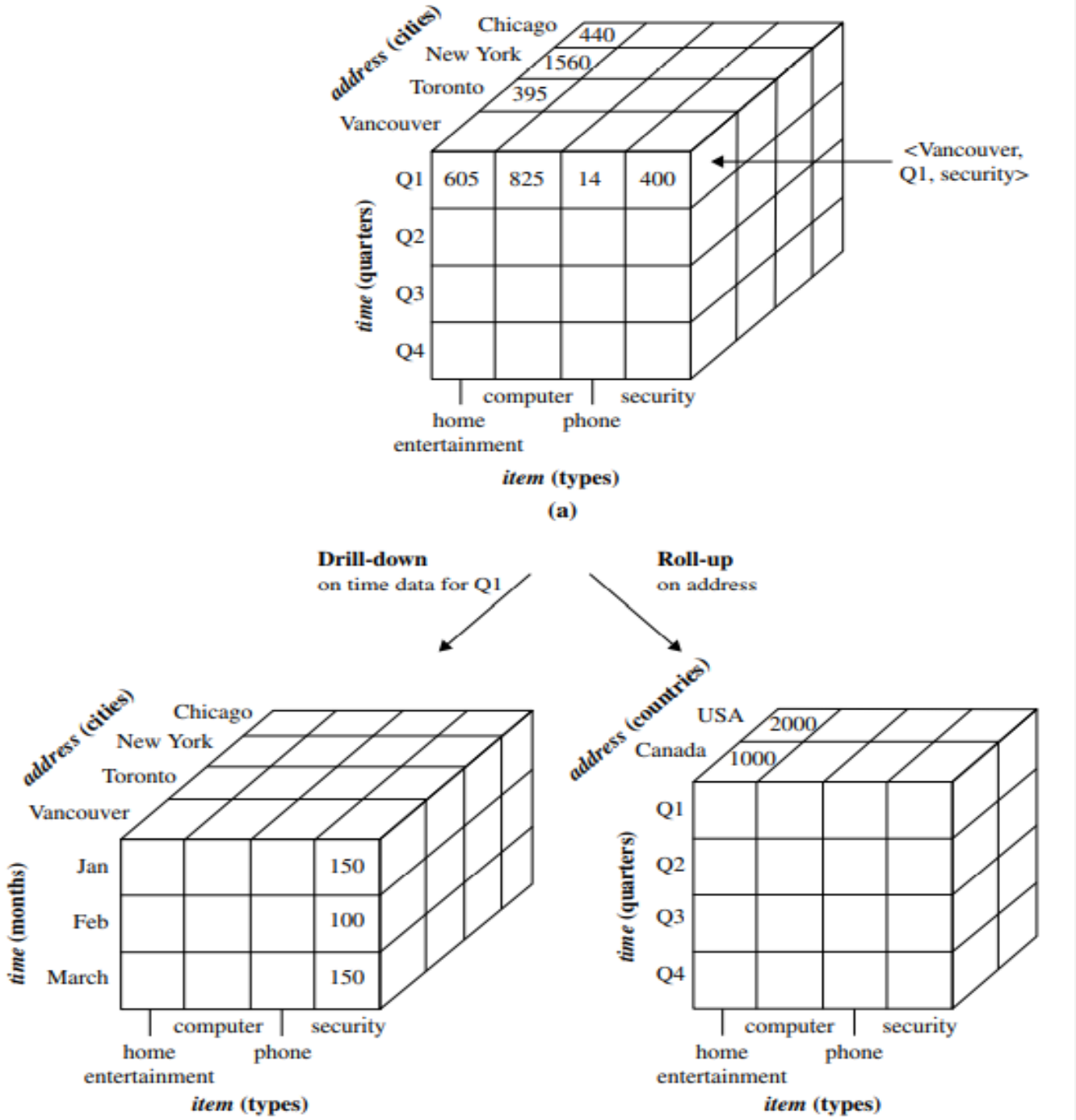


Figure 1.7 : A multidimensional data cube, commonly used for data warehousing,

(a) showing summarized data for AllElectronics and

(b) showing summarized data resulting from drill-down and roll-up operations on the cube in (a). For improved readability, only some of the cube cell values are shown

DATA WAREHOUSE ARCHITECTURE

Data warehouses often adopt a three-tier architecture, as presented in Figure 4.1.

1. The bottom tier is a warehouse database server that is almost always a relational database system. Back-end tools and utilities are used to feed data into the bottom tier from operational databases or other external sources (e.g., customer profile information provided by external consultants). These tools and utilities perform data extraction, cleaning, and transformation (e.g., to merge similar data from different sources into a unified format), as well as load and refresh functions to update the data warehouse. This tier also contains a metadata repository, which stores information about the data warehouse and its contents.
2. The middle tier is an OLAP server that is typically implemented using either (1) a relational OLAP (ROLAP) model (i.e., an extended relational DBMS that maps operations on multidimensional data to standard relational operations); or (2) a multidimensional OLAP (MOLAP) model (i.e., a special-purpose server that directly implements multidimensional data and operations).
3. The top tier is a front-end client layer, which contains query and reporting tools, analysis tools, and/or data mining tools (e.g., trend analysis, prediction, and so on).

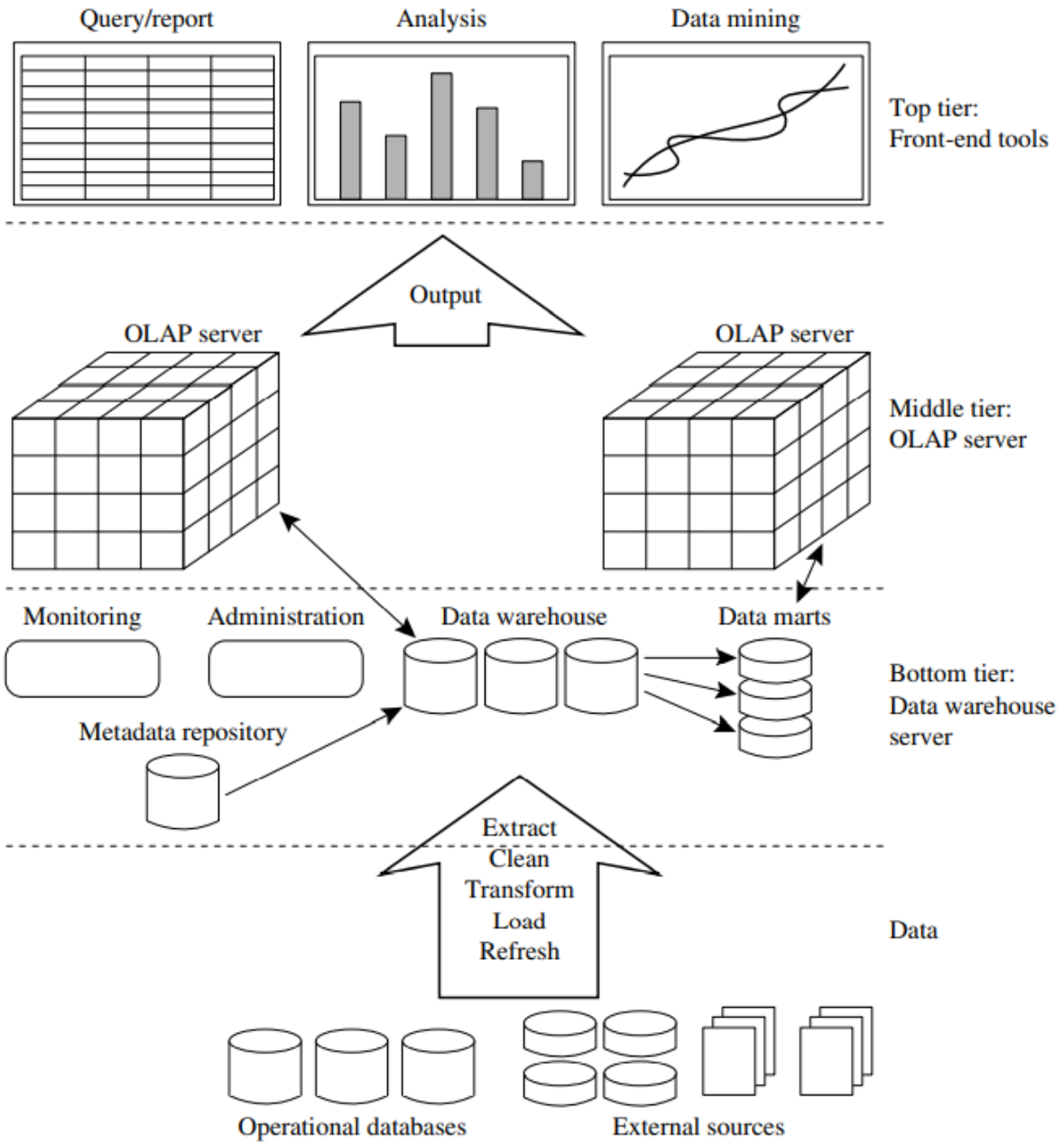


Figure 4.1 A three-tier data warehousing architecture.

DATA WAREHOUSE IMPLEMENTATION

An overview of methods for the efficient implementation of data warehouse systems:

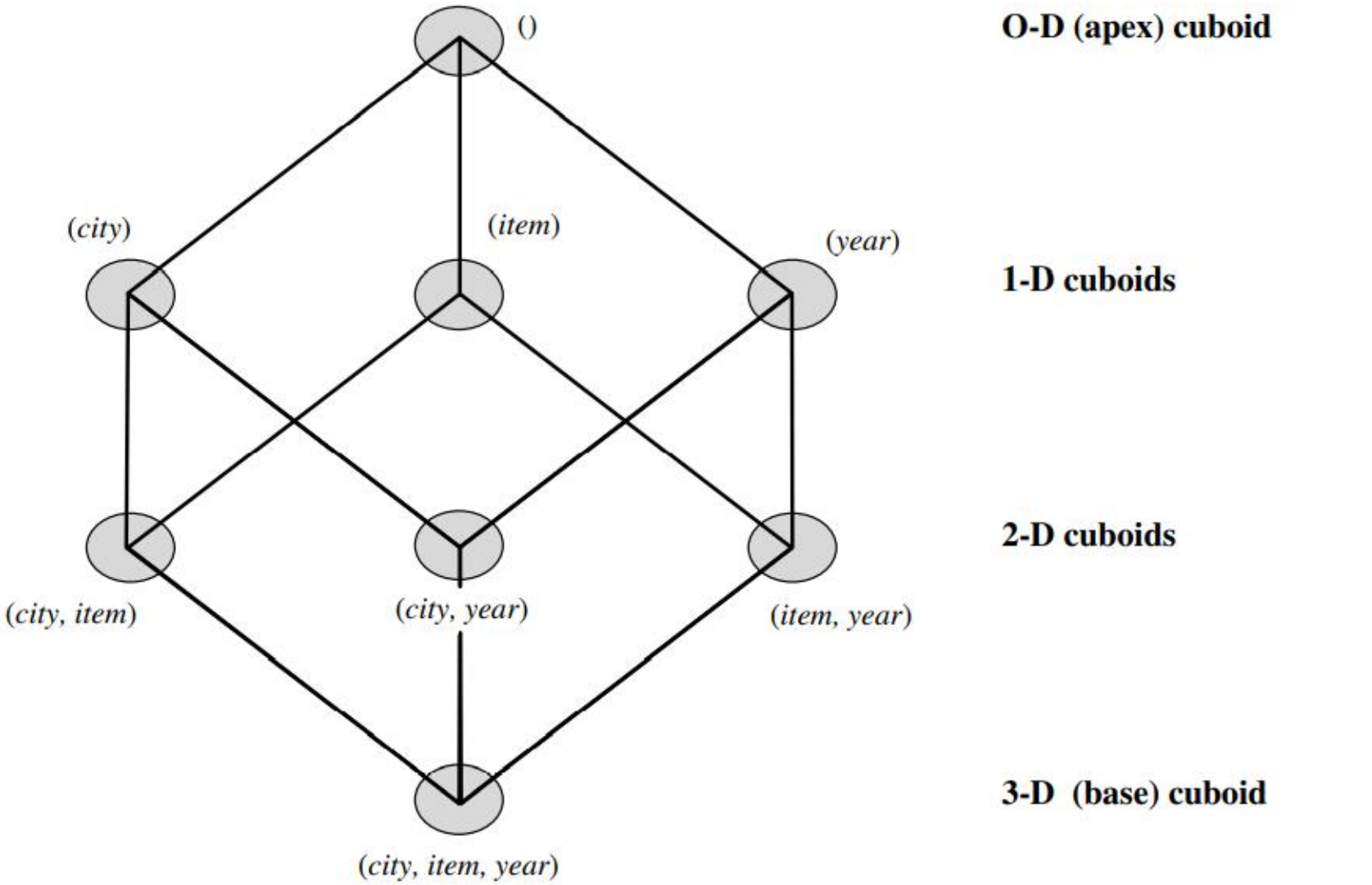
1. The compute cube Operator and the Curse of Dimensionality

One approach to cube computation extends SQL so as to include a compute cube operator. The compute cube operator computes aggregates over all subsets of the dimensions specified in the operation. This can require excessive storage space, especially for large numbers of dimensions. We start with an intuitive look at what is involved in the efficient computation of data cubes.

Example 3.11 A data cube is a lattice of cuboids. Suppose that you would like to create a data cube for AllElectronics sales that contains the following: city, item, year, and sales in dollars. You would like to be able to analyze the data, with queries such as the following:

- "Compute the sum of sales, grouping by city and item."
- "Compute the sum of sales, grouping by city."
- "Compute the sum of sales, grouping by item."

- What is the total number of cuboids, or group-by's, that can be computed for this data cube? Taking the three attributes, city, item, and year, as the dimensions for the data cube, and sales in dollars as the measure, the total number of cuboids, or groupby's, that can be computed for this data cube is $2^3 = 8$.
- The possible group-by's are the following: {(city, item, year), (city, item), (city, year), (item, year), (city), (item), (year), ()}, where () means that the group-by is empty (i.e., the dimensions are not grouped). These group-by's form a lattice of cuboids for the data cube, as shown in Figure 3.14.
- The base cuboid contains all three dimensions, city, item, and year. It can return the total sales for any combination of the three dimensions. The apex cuboid, or 0-D cuboid, refers to the case where the group-by is empty. It contains the total sum of all sales. The base cuboid is the least generalized (most specific) of the cuboids.
- The apex cuboid is the most generalized (least specific) of the cuboids, and is often denoted as all. If we start at the apex cuboid and explore downward in the lattice, this is equivalent to drilling down within the data cube. If we start at the base cuboid and explore upward, this is akin to rolling up.



Lattice of cuboids, making up a 3-D data cube. Each cuboid represents a different group-by. The base cuboid contains the three dimensions *city*, *item*, and *year*.

- An SQL query containing no group-by, such as "compute the sum of total sales," is a zero-dimensional operation. An SQL query containing one group-by, such as "compute the sum of sales, group by city," is a one-dimensional operation. A cube operator on n dimensions is equivalent to a collection of group by statements, one for each subset of the n dimensions. Therefore, the cube operator is the n-dimensional generalization of the group by operator.

2. Indexing OLAP Data: Bitmap Index and Join Index

- To facilitate efficient data accessing, most data warehouse systems support index structures and materialized views (using cuboids).
- In this subsection, we examine how to index OLAP data by bitmap indexing and join indexing. The bitmap indexing method is popular in OLAP products because it allows quick searching in data cubes.
- The bitmap index is an alternative representation of the record ID (RID) list. In the bitmap index for a given attribute, there is a distinct bit vector, Bv, for each value v in the attribute's domain. If a given attribute's domain consists of n values, then n bits are needed for each entry in the bitmap index (i.e., there are n bit vectors).
- If the attribute has the value v for a given row in the data table, then the bit representing that value is set to 1 in the corresponding row of the bitmap index. All other bits for that row are set to 0

Example 4.7 **Bitmap indexing.** In the *AllElectronics* data warehouse, suppose the dimension *item* at the top level has four values (representing item types): "*home entertainment,*" "*computer,*" "*phone,*" and "*security.*" Each value (e.g., "*computer*") is represented by a bit vector in the *item* bitmap index table. Suppose that the cube is stored as a relation table with 100,000 rows. Because the domain of *item* consists of four values, the bitmap index table requires four bit vectors (or lists), each with 100,000 bits. Figure 4.15 shows a base (data) table containing the dimensions *item* and *city*, and its mapping to bitmap index tables for each of the dimensions. ■

Base table

RID	*item*	*city*
R1	H	V
R2	C	V
R3	P	V
R4	S	V
R5	H	T
R6	C	T
R7	P	T
R8	S	T

item bitmap index table

RID	H	C	P	S
R1	1	0	0	0
R2	0	1	0	0
R3	0	0	1	0
R4	0	0	0	1
R5	1	0	0	0
R6	0	1	0	0
R7	0	0	1	0
R8	0	0	0	1

city bitmap index table

RID	V	T
R1	1	0
R2	1	0
R3	1	0
R4	1	0
R5	0	1
R6	0	1
R7	0	1
R8	0	1

Note: H for "home entertainment," C for "computer," P for "phone," S for "security," V for "Vancouver," T for "Toronto."

Figure 4.15 Indexing OLAP data using bitmap indices.

- Join indexing registers the joinable rows of two relations from a relational database. For example, if two relations R(RID, A) and S(B, SID) join on the attributes A and B, then the join index record contains the pair (RID, SID), where RID and SID are record identifiers from the R and S relations, respectively.
- Hence, the join index records can identify joinable tuples without performing costly join operations. Join indexing is especially useful for maintaining the relationship between a foreign key2 and its matching primary keys, from the joinable relation.
- We can use join indices to identify subcubes that are of interest.

Example 4.8 **Join indexing.** In Example 3.4, we defined a star schema for *AllElectronics* of the form "*sales_star* [*time, item, branch, location*]: *dollars_sold* = sum (*sales_in_dollars*)." An example of a join index relationship between the *sales* fact table and the *location* and *item* dimension tables is shown in Figure 4.16. For example, the "Main Street" value in the *location* dimension table joins with tuples T57, T238, and T884 of the *sales* fact table. Similarly, the "Sony-TV" value in the *item* dimension table joins with tuples T57 and T459 of the *sales* fact table. The corresponding join index tables are shown in Figure 4.17.

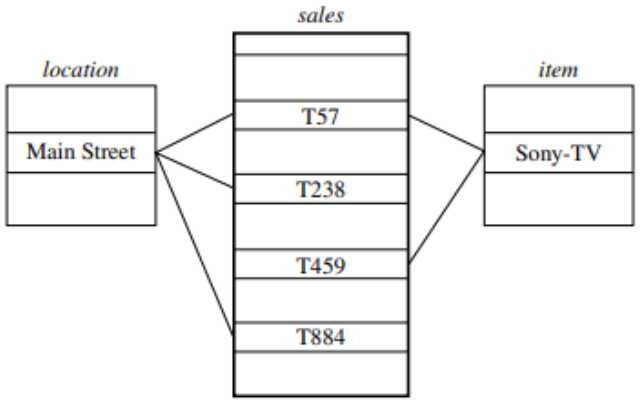


Figure 4.16 Linkages between a *sales* fact table and *location* and *item* dimension tables.

Join index table for *location/sales*

location	*sales_key*
...	...
Main Street	T57
Main Street	T238
Main Street	T884
...	...

Join index table for *item/sales*

item	*sales_key*
...	...
Sony-TV	T57
Sony-TV	T459
...	...

Join index table linking *location* and *item* to *sales*

location	*item*	*sales_key*
...	...	...
Main Street	Sony-TV	T57
...	...	...

Figure 4.17 Join index tables based on the linkages between the *sales* fact table and the *location* and *item* dimension tables shown in Figure 4.16.

3. Efficient Processing of OLAP Queries

The purpose of materializing cuboids and constructing OLAP index structures is to speed up query processing in data cubes. Given materialized views, query processing should proceed as follows:

1. Determine which operations should be performed on the available cuboids:

This involves transforming any selection, projection, roll-up (group-by), and drill-down operations specified in the query into corresponding SQL and/or OLAP operations. For example, slicing and dicing a data cube may correspond to selection and/or projection operations on a materialized cuboid.

2. Determine to which materialized cuboid(s) the relevant operations should be applied:

This involves identifying all of the materialized cuboids that may potentially be used to answer the query, pruning the set using knowledge of "dominance" relationships among the cuboids, estimating the costs of using the remaining materialized cuboids, and selecting the cuboid with the least cost

4. OLAP Server Architectures: ROLAP versus MOLAP versus HOLAP

Logically, OLAP servers present business users with multidimensional data from data warehouses or data marts, without concerns regarding how or where the data are stored. However, the physical architecture and implementation of OLAP servers must consider data storage issues. Implementations of a warehouse server for OLAP processing include the following:

Relational OLAP (ROLAP) servers: These are the intermediate servers that stand in between a relational back-end server and client front-end tools. They use a relational or extended-relational DBMS to store and manage warehouse data, and OLAP middleware to support missing pieces. ROLAP servers include optimization for each DBMS back end, implementation of aggregation navigation logic, and additional tools and services. ROLAP technology tends to have greater scalability than MOLAP technology. The DSS server of Microstrategy, for example, adopts the ROLAP approach.

Multidimensional OLAP (MOLAP) servers: These servers support multidimensional data views through array-based multidimensional storage engines. They map multidimensional views directly to data cube array structures. The advantage of using a data cube is that it allows fast indexing to precomputed summarized data. Notice that with multidimensional data stores, the storage utilization may be low if the data set is sparse. In such cases, sparse matrix compression techniques should be explored.

Hybrid OLAP (HOLAP) servers: The hybrid OLAP approach combines ROLAP and MOLAP technology, benefiting from the greater scalability of ROLAP and the faster computation of MOLAP. For example, a HOLAP server may allow large volumes of detailed data to be stored in a relational database, while aggregations are kept in a separate MOLAP store. The Microsoft SQL Server 2000 supports a hybrid OLAP server.

Specialized SQL servers: To meet the growing demand of OLAP processing in relational databases, some database system vendors implement specialized SQL servers that provide advanced query language and query processing support for SQL queries over star and snowflake schemas in a read-only environment.

FURTHER DEVELOPMENT OF DATA CUBE AND OLAP TECHNOLOGY

- It describes data mining by discovery-driven exploration of data cubes, where anomalies in the data are automatically detected and marked for the user with visual cues.
- It describes multifeature cubes for complex data mining queries involving multiple dependent aggregates at multiple granularity.
- It presents methods for constrained gradient analysis in data cubes, which identifies cube cells that have dramatic changes in value in comparison with their siblings, ancestors, or descendants

FROM DATA WAREHOUSING TO DATA MINING

- Refer reference book

DATA CUBE COMPUTATION AND DATA GENERALIZATION: EFFICIENT METHODS 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 section explores efficient methods for data cube computation. It introduces general concepts and computation strategies relating to cube materialization, detail specific computation algorithms, namely, MultiWay array aggregation, BUC, Star-Cubing, the computation of shell fragments, and the computation of cubes involving complex measures.

Cube Materialization: Full Cube, Iceberg Cube, Closed Cube, and Shell Cube

- Figure 4.1 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. For the purposes of our discussion in this chapter, we will always use the term data cube to refer to a lattice of cuboids rather than an individual cuboid.

- full cube : all the cells of all of the cuboids for a given data cube.
- The cells that cannot pass the threshold are likely to be too trivial to warrant further analysis. Such partially materialized cubes are known as iceberg cubes.
- A closed cube is a data cube consisting of only closed cells.

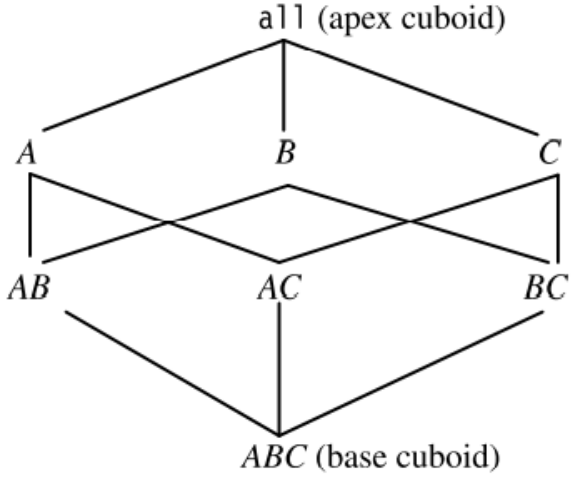


Figure 4.1 Lattice of cuboids, making up a 3-D data cube with the dimensions *A*, *B*, and *C* for some aggregate measure, *M*.

Multiway Array Aggregation for Full Cube 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.