Introduction

A Database Management System (DBMS) stores data in the form of tables and uses an ER model and the goal is ACID properties. For example, a DBMS of a college has tables for students, faculty, etc.

A **Data Warehouse** is separate from DBMS, it stores a huge amount of data, which is typically collected from multiple heterogeneous sources like files, DBMS, etc. The goal is to produce statistical results that may help in decision-making. For example, a college might want to see quick different results, like how the placement of CS students has improved over the last 10 years, in terms of salaries, counts, etc.

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.

Need for data warehousing

An ordinary Database can store MBs to GBs of data and that too for a specific purpose. For storing data of TB size, the storage shifted to the Data Warehouse. Besides this, a transactional database doesn't offer itself to analytics. To effectively perform analytics, an organization keeps a central Data Warehouse to closely study its business by organizing, understanding, and using its historical data for making strategic decisions and analyzing trends.

Advantages of Data Warehouse:

Decision-Making Support: Data warehouses empower businesses with valuable insights and actionable information. By analyzing historical data, trends, and patterns, organizations can make informed decisions and develop effective strategies.

Data Consistency and Quality: Data warehouses ensure data consistency and integrity by integrating and cleansing data during the ETL process. This results in more reliable and accurate information for analysis.

Improved Performance: Data warehouses are specifically optimized for analytical tasks. Their structured design and indexing enable faster query response times, enhancing overall performance.

Historical Analysis: The ability to analyze historical data helps organizations understand long-term trends, spot patterns, and anticipate future developments.

Enhanced Business Intelligence(BI): Data warehouses act as the foundation for Business Intelligence (BI) initiatives. They provide the data required for creating reports, dashboards, and visualizations that aid in data-driven decision-making.

Data Warehouse vs DBMS

Database	Data Warehouse
A common Database is based on operational or transactional processing. Each operation is an indivisible transaction.	A data Warehouse is based on analytical processing.
Generally, a Database stores current and up-to-date data which is used for daily operations.	A Data Warehouse maintains historical data over time. Historical data is the data kept over years and can used for trend analysis, make future predictions and decision support.
A database is generally application specific. **Example** – A database stores related data, such as the student details in a school.	A Data Warehouse is integrated generally at the organization level, by combining data from different databases. **Example** – A data warehouse integrates the data from o analysis can be done to get results , such as the best perf
Constructing a Database is not so expensive.	Constructing a Data Warehouse can be expensive.

Components of Data Warehousing

A data warehouse comprises several essential components that work together to store, organize, and provide access to data for analytical purposes. Let's delve into each component:

Data Sources:
Data warehouses collect data from multiple sources within an organization. These sources may include transactional databases, customer relationship management (CRM) systems, supply chain records, financial applications, and more. Additionally, data from external sources, such as market research reports or public datasets, may also be integrated into the warehouse. Gathering data from diverse sources ensures a comprehensive and holistic view of an organization's operations.

ETL (Extract, Transform, Load):
ETL is the backbone of the data warehousing process. It involves three fundamental steps:

Extract: Data is extracted from the various source systems. This could involve identifying relevant data, reading it from the sources, and transferring it to the data warehouse environment.

Data cleaning, which detects errors in the data and rectifies them when possible.

Transform: The extracted data is then transformed into a consistent format suitable for storage and analysis. This step includes data cleansing, where inaccuracies, duplicates, and inconsistencies are resolved. Data is also converted into a standardized structure to enable effective querying.

Load: Finally, the transformed data is loaded into the data warehouse, where it becomes accessible for analytical purposes. This loading process can be periodic or real-time, depending on the organization's needs.

Data Storage
The data warehouse stores data in a manner optimized for analytical processing. It employs a dimensional data model, often represented using star or snowflake schemas. In this model, data is organized into fact tables (containing quantitative data or measures) and dimension tables (providing descriptive attributes or context). Such a structure allows for efficient querying and reporting, making it easier for users to navigate and retrieve relevant data.

Metadata Management
Metadata refers to data about data. In the context of a data warehouse, it includes information about the data stored, its origin, meaning, and relationships with other data elements. Proper metadata management ensures that users can understand the data and its context, facilitating effective data analysis. It also aids in data governance, data lineage tracking, and data quality assessment.

Data Access Tools
Data warehouses are accessed by end-users, such as analysts, business managers, and executives, through various reporting and analysis tools. These tools provide an intuitive and user-friendly interface for querying the data, creating custom reports, visualizing trends, and generating insights. Common data access tools include Business Intelligence (BI) platforms, SQL-based interfaces, and data visualization tools.

Security and Access Control
Given the sensitivity of the data stored in a data warehouse, robust security measures are essential. Access to the data must be restricted and controlled to prevent unauthorized usage or data breaches. Implementing encryption, role-based access controls, and data masking are some of the security practices employed to safeguard the data warehouse.

Characteristics of data warehousing

Subject-Oriented: A data warehouse can be used to analyze a particular subject area. Forexample, "sales" can be a particular subject.

Time-Variant: A crucial aspect of data warehouses is their ability to store historical data. They maintain a historical record of data changes, enabling trend analysis, performance comparisons over time, and other historical insights.

Non-Volatile: Data in a warehouse is not altered in real-time. Once data is loaded into the warehouse, it becomes read-only and remains unchanged. This ensures data consistency and integrity for analytical purposes.

Integrated: Data warehouses integrate data from multiple sources and systems, providing a unified and consistent view of the organization's data. This integration eliminates data silos and enhances data quality.

Optimized for Analytics: Unlike operational databases optimized for transaction processing, data warehouses are designed for complex queries and data analysis. They are structured to support efficient analytical operations.

Difference between OLAP and OLTP:

Category	OLAP (Online Analytical Processing)	OLTP (Online Transaction Processing)
Data source	Consists of historical data from various Databases.	Consists of only operational current data.
Method used	It makes use of a data warehouse.	It makes use of a standard database management system (DBMS).
Application	It is subject-oriented. Used for Data Mining, Analytics, Decisions making, etc.	It is application-oriented. Used for business tasks.
Usage of data	The data is used in planning, problem-solving, and decision-making.	The data is used to perform day-to-day fundamental operations.
Task	It provides a multi-dimensional view of different business tasks.	It reveals a snapshot of present business tasks.
Purpose	It serves the purpose to extract information for analysis and decision-making.	It serves the purpose to Insert, Update, and Delete information from the database.
Volume of data	A large amount of data is stored typically in TB, PB	The size of the data is relatively small as the historical data is archived in MB, and GB.
Queries	Relatively slow as the amount of data involved is large. Queries may take hours.	Very Fast as the queries operate on 5% of the data.
Update	The OLAP database is not often updated. As a result, data integrity is unaffected.	The data integrity constraint must be maintained in an OLTP database.
Backup and Recovery	It only needs backup from time to time as compared to OLTP.	The backup and recovery process is maintained rigorously
Processing time	The processing of complex queries can take a lengthy time.	It is comparatively fast in processing because of simple and straightforward queries.
Types of users	This data is generally managed by CEO, MD, and GM.	This data is managed by clerks and managers.

Category	OLAP (Online Analytical Processing)	OLTP (Online Transaction Processing)
Operations	Only read and rarely write operations.	Both read and write operations.
Productivity	Improves the efficiency of business analysts.	Enhances the user's productivity.

Data Warehouse Models: Enterprise Warehouse, Data Mart, and Virtual Warehouse

From the architecture point of view, there are three data warehouse models: the *enterprise warehouse*, the *data mart*, and the *virtual warehouse*.

Enterprise warehouse: An enterprise warehouse collects all of the information about subjects spanning the entire organization. It provides corporate-wide data integration, usually from one or more operational systems or external information providers, and is cross-functional in scope. It typically contains detailed data as well as summarized data, and can range in size from a few gigabytes to hundreds of gigabytes, terabytes, or beyond. An enterprise data warehouse may be implemented on traditional mainframes, computer superservers, or parallel architecture platforms. It requires extensive business modeling and may take years to design and build.

Data mart: A data mart contains a subset of corporate-wide data that is of value to a specific group of users. The scope is confined to specific selected subjects. For example, a marketing data mart may confine its subjects to customer, item, and sales. The data contained in data marts tend to be summarized.

Data marts are usually implemented on low-cost departmental servers that are Unix/Linux or Windows based. The implementation cycle of a data mart is more likely to be measured in weeks rather than months or years. However, it may involve complex integration in the long run if its design and planning were not enterprise-wide.

Depending on the source of data, data marts can be categorized as independent or dependent. *Independent* data marts are sourced from data captured from one or more operational systems or external information providers, or from data generated locally within a particular department or geographic area. *Dependent* data marts are sourced directly from enterprise data warehouses.

Virtual warehouse: A virtual warehouse is a set of views over operational databases. For efficient query processing, only some of the possible summary views may be materialized. A virtual warehouse is easy to build but requires excess capacity on operational database servers.

Data Warehouse Architecture

The following architecture properties are necessary for a data warehouse system:

Separation: Analytical and transactional processing should be keep apart as much as possible.

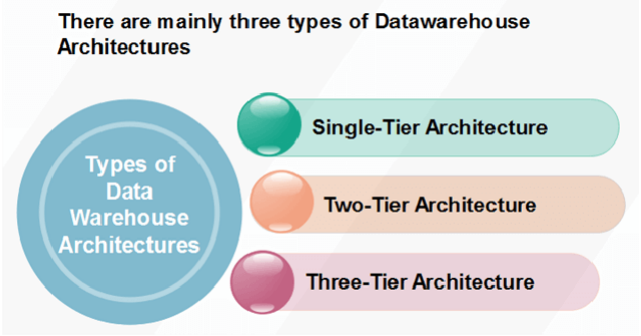
Scalability: Hardware and software architectures should be simple to upgrade the data volume, which has to be managed and processed, and the number of user's requirements, which have to be met, progressively increase.

Extensibility: The architecture should be able to perform new operations and technologies without redesigning the whole system.

Security: Monitoring accesses are necessary because of the strategic data stored in the data warehouses.

Administerability: Data Warehouse management should not be complicated.

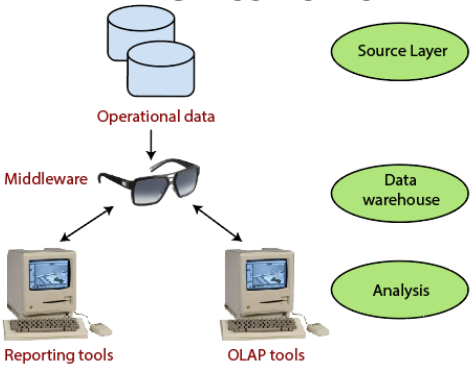
Data warehouses and their architectures very depending upon the elements of an organization's situation.
Three common architectures are:



Single-Tier Architecture:

Single-Tier architecture is not periodically used in practice. Its purpose is to minimize the amount of data stored to reach this goal; it removes data redundancies.

The figure shows the only layer physically available is the source layer. In this method, data warehouses are virtual. This means that the data warehouse is implemented as a multidimensional view of operational data created by specific middleware, or an intermediate processing layer.

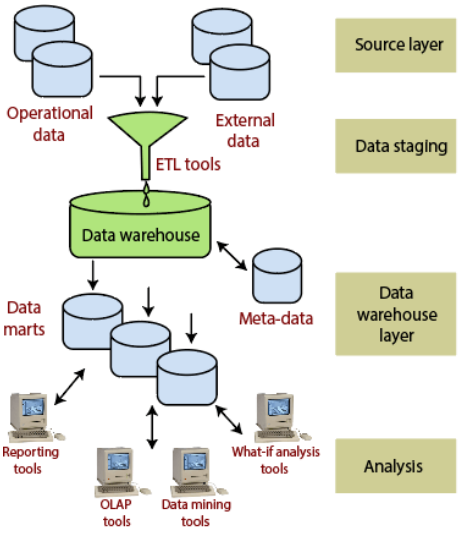


Single-Tier Data Warehouse Architecture

The vulnerability of this architecture lies in its failure to meet the requirement for separation between analytical and transactional processing. Analysis queries are agreed to operational data after the middleware interprets them. In this way, queries affect transactional workloads.

Two-Tier Architecture:

The requirement for separation plays an essential role in defining the two-tier architecture for a data warehouse system, as shown in fig:



Two-Tier Data Warehouse Architecture

Although it is typically called two-layer architecture to highlight a separation between physically available sources and data warehouses, in fact, consists of four subsequent data flow stages:

1. **Source layer:** A data warehouse system uses a heterogeneous source of data. That data is stored initially to corporate relational databases or legacy databases, or it may come from an information system outside the corporate walls.

2. **Data Staging:** The data stored to the source should be extracted, cleansed to remove inconsistencies and fill gaps, and integrated to merge heterogeneous sources into one standard schema. The so-named **Extraction**, **Transformation**, and **Loading Tools (ETL)** can combine heterogeneous schemata, extract, transform, cleanse, validate, filter, and load source data into a data warehouse.
3. **Data Warehouse layer:** Information is saved to one logically centralized individual repository: a data warehouse. The data warehouses can be directly accessed, but it can also be used as a source for creating data marts, which partially replicate data warehouse contents and are designed for specific enterprise departments. Meta-data repositories store information on sources, access procedures, data staging, users, data mart schema, and so on.
4. **Analysis:** In this layer, integrated data is efficiently, and flexible accessed to issue reports, dynamically analyze information, and simulate hypothetical business scenarios. It should feature aggregate information navigators, complex query optimizers, and customer-friendly GUIs.

Three-Tier Data Warehouse Architecture

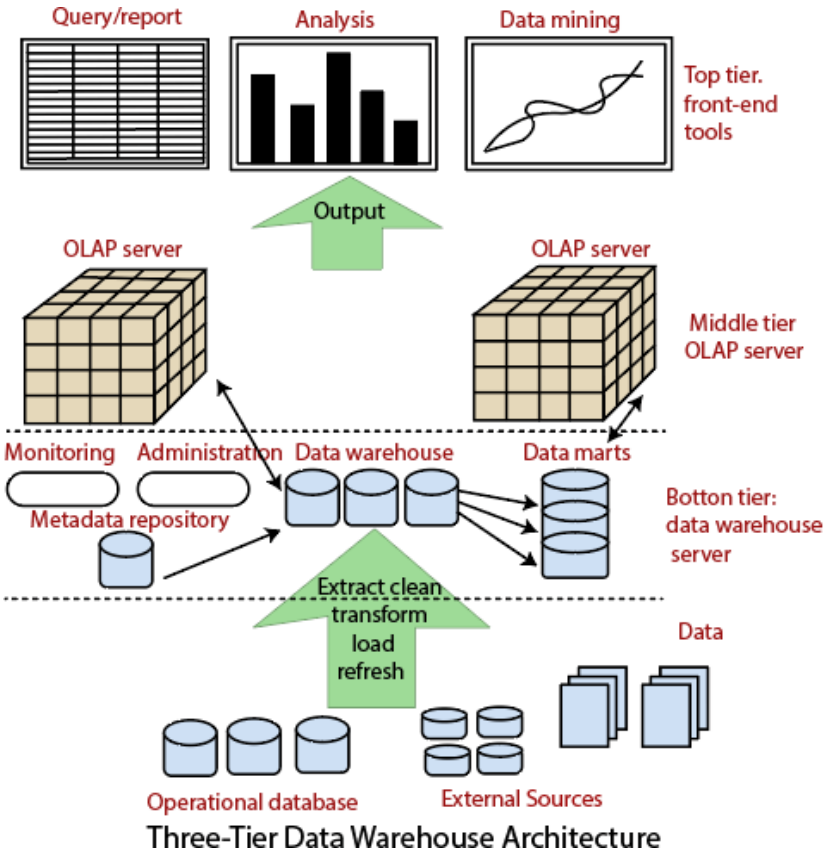
Data Warehouses usually have a three-level (tier) architecture that includes:

1. Bottom Tier (Data Warehouse Server)
2. Middle Tier (OLAP Server)
3. Top Tier (Front end Tools).

A **bottom-tier** that consists of the **Data Warehouse server**, which is almost always an RDBMS. It may include several specialized data marts and a metadata repository. A **middle-tier** which consists of an **OLAP server** for fast querying of the data warehouse.

The OLAP server is implemented using either **(1)** A **Relational OLAP (ROLAP) model**, i.e., an extended relational DBMS that maps functions on multidimensional data to standard relational operations. **(2)** A **Multidimensional OLAP (MOLAP) model**, i.e., a particular purpose server that directly implements multidimensional information and operations.

A **top-tier** that contains **front-end tools** for displaying results provided by OLAP, as well as additional tools for data mining of the OLAP-generated data. The overall Data Warehouse Architecture is shown in fig:



Three-Tier Data Warehouse Architecture

Dimensional Modeling

Dimensional modeling represents data with a cube operation, making more suitable logical data representation with OLAP data management. The perception of Dimensional Modeling was developed by **Ralph Kimball** and is consist of **"fact"** and **"dimension"** tables.

In dimensional modeling, the transaction record is divided into either **"facts,"** which are frequently numerical transaction data, or **"dimensions,"** which are the reference information that gives context to the facts.

Objectives of Dimensional Modeling:

1. To produce database architecture that is easy for end-clients to understand and write queries.
2. To maximize the efficiency of queries. It achieves these goals by minimizing the number of tables and relationships between them.

Elements of Dimensional Modeling:

Fact: It is a collection of associated data items, consisting of measures and context data. It typically represents business items or business transactions.

Dimensions: It is a collection of data which describe one business dimension. Dimensions decide the contextual background for the facts, and they are the framework over which OLAP is performed.

Measure: It is a numeric attribute of a fact, representing the performance or behavior of the business relative to the dimensions.

Data Cube: Multi-Dimensional Data Model

A multidimensional model views data in the form of a data-cube. A data cube enables data to be modeled and viewed in multiple dimensions. It is defined by dimensions and facts.

The dimensions are the perspectives or entities concerning which an organization keeps records. For example, a shop may create a sales data warehouse to keep records of the store's sales for the dimension time, item, and location. These dimensions allow the save to keep track of things, for example, monthly sales of items and the locations at which the items were sold. Each dimension has a table related to it, called a dimensional table, which describes the dimension further. For example, a dimensional table for an item may contain the attributes item_name, brand, and type.

Although we usually think of cubes as 3-D geometric structures, in data warehousing the data cube is *n*-dimensional. To gain a better understanding of data cubes and the multidimensional data model, let's start by looking at a simple 2-D data cube that is, in fact, a table or spreadsheet for sales data from *AllElectronics*. In particular, we will look at the *AllElectronics* sales data for items sold per quarter in the city of Vancouver. These data are shown in Table 4.2. In this 2-D representation, the sales for Vancouver are shown with respect to the *time* dimension (organized in quarters) and the *item*dimension (organized according to the types of items sold). The fact or measure displayed is *dollars sold* (in thousands).

Table 4.2 2-D View of Sales Data for *AllElectronics* According to *time* and *item*

	location = *"Vancouver"*			
	item *(type)*			
time *(quarter)*	*home entertainment*	*computer*	*phone*	*security*
Q1	605	825	14	400
Q2	680	952	31	512
Q3	812	1023	30	501
Q4	927	1038	38	580

Now, suppose that we would like to view the sales data with a third dimension. For instance, suppose we would like to view the data according to *time* and *item*, as well as *location*, for the cities Chicago, New York, Toronto, and Vancouver. These 3-D data are shown in Table 4.3. The 3-D data in the table are

represented as a series of 2-D tables. Conceptually, we may also represent the same data in the form of a 3-D data cube, as in Figure 4.3.

Table 4.3 3-D View of Sales Data for *AllElectronics* According to *time*, *item*, and *location*

	location = "Chicago"				***location*** = "New York"				***location*** = "Toronto"				***location*** = "Vancouver"			
	item				***item***				***item***				***item***			
	home				*home*				*home*				*home*			
time	*ent.*	*comp.*	*phone*	*sec.*	*ent.*	*comp.*	*phone*	*sec.*	*ent.*	*comp.*	*phone*	*sec.*	*ent.*	*comp.*	*phone*	*sec.*
Q1	854	882	89	623	1087	968	38	872	818	746	43	591	605	825	14	400
Q2	943	890	64	698	1130	1024	41	925	894	769	52	682	680	952	31	512
Q3	1032	924	59	789	1034	1048	45	1002	940	795	58	728	812	1023	30	501
Q4	1129	992	63	870	1142	1091	54	984	978	864	59	784	927	1038	38	580

Suppose that we would now like to view our sales data with an additional fourth dimension such as *supplier*. Viewing things in 4-D becomes tricky. However, we can think of a 4-D cube as being a series of 3-D cubes. If we continue in this way, we may display any *n*-dimensional data as a series of (*n*-1)-dimensional "cubes." The data cube is a metaphor for multidimensional data storage. The actual physical storage of such data may differ from its logical representation. The important thing to remember is that data cubes are *n*-dimensional and do not confine data to 3-D. The above tables show the data at different degrees of summarization. In the data warehousing research literature, a data cube like those shown in Figures 4.3 and 4.4 is often referred to as a **cuboid**. Given a set of dimensions, we can generate a cuboid for each of the possible subsets of the given dimensions. The result would form a *lattice* of cuboids, each showing the data at a different level of summarization, or group-by. The lattice of cuboids is then referred to as a data cube. Figure shows a lattice of cuboids forming a data cube for the dimensions *time*, *item*, *location*, and *supplier*.

The cuboid that holds the lowest level of summarization is called the **base cuboid**. For example, the 4-D cuboid in Figure 4.4 is the base cuboid for the given *time*, *item*, *location*, and *supplier* dimensions. Figure 4.3 is a 3-D (nonbase) cuboid for *time*, *item*, and *location*, summarized for all suppliers. The 0-D cuboid, which holds the highest level of summarization, is called the **apex cuboid**. In our example, this is the total sales, or *dollars sold*, summarized over all four dimensions. The apex cuboid is typically denoted by all.

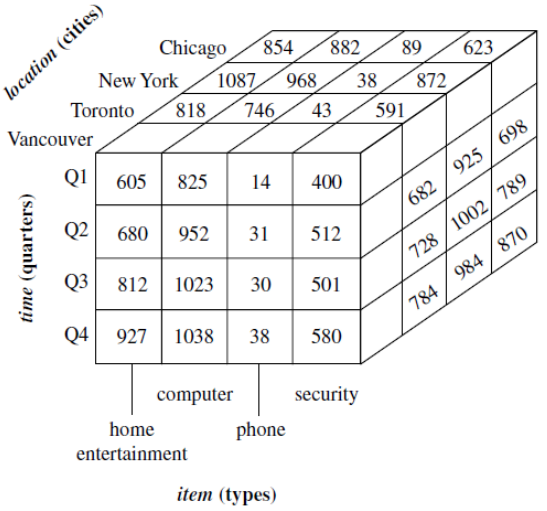


Figure 4.3 A 3-D data cube representation of the data in Table 4.3, according to *time*, *item*, and *location*. The measure displayed is *dollars sold* (in thousands).

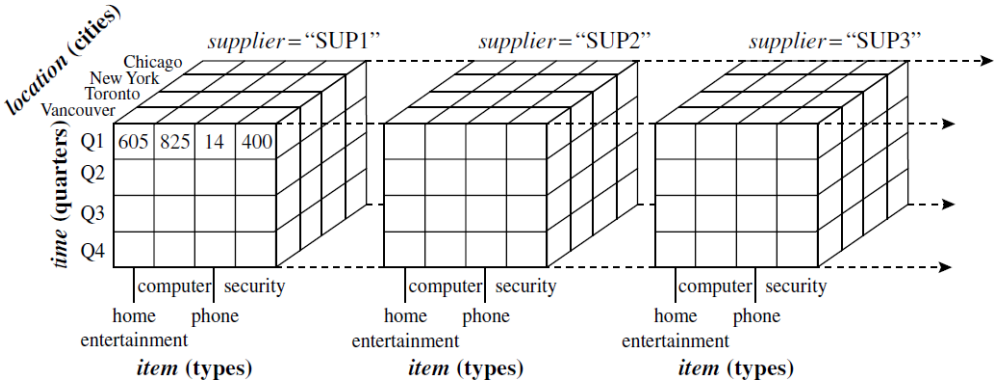
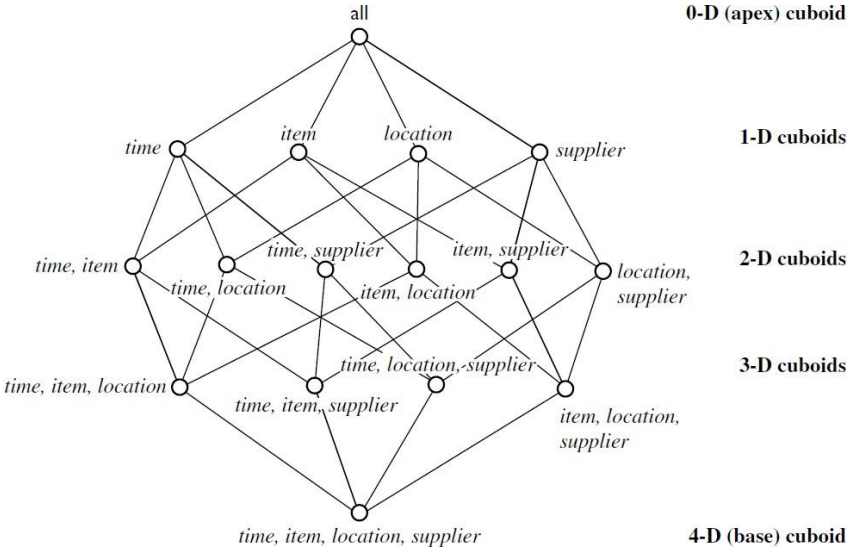


Figure 4.4 A 4-D data cube representation of sales data, according to *time*, *item*, *location*, and *supplier*. The measure displayed is *dollars sold* (in thousands). For improved readability, only some of the cube values are shown.



Stars, Snowflakes, and Fact Constellations: Schemas for Multidimensional Data Models

The entity-relationship data model is commonly used in the design of relational databases, where a database schema consists of a set of entities and the relationships between them. Such a data model is appropriate for online transaction processing. A data warehouse, however, requires a concise, subject-oriented schema that facilitates online data analysis.
The most popular data model for a data warehouse is a **multidimensional model**, which can exist in the formof a **star schema**, a **snowflake schema**, or a **fact constellation schema**. Let's look at each of these.

Star schema: The most common modeling paradigm is the star schema, in which the data warehouse contains (1) a large central table (**fact table**) containing the bulk of the data, with no redundancy, and (2) a set of smaller attendant tables (**dimension tables**), one for each dimension. The schema graph resembles a starburst, with the dimension tables displayed in a radial pattern around the central fact table.

EX 1: Star schema. A star schema for *AllElectronics* sales is shown in Figure 4.6. Sales are considered along four dimensions: *time*, *item*, *branch*, and *location*. The schema contains a central fact table for *sales* that contains keys to each of the four dimensions, along with two measures: *dollars sold* and *units sold*. To minimize the size of the fact table, dimension identifiers (e.g., *time key* and *item key*) are system-generated identifiers.

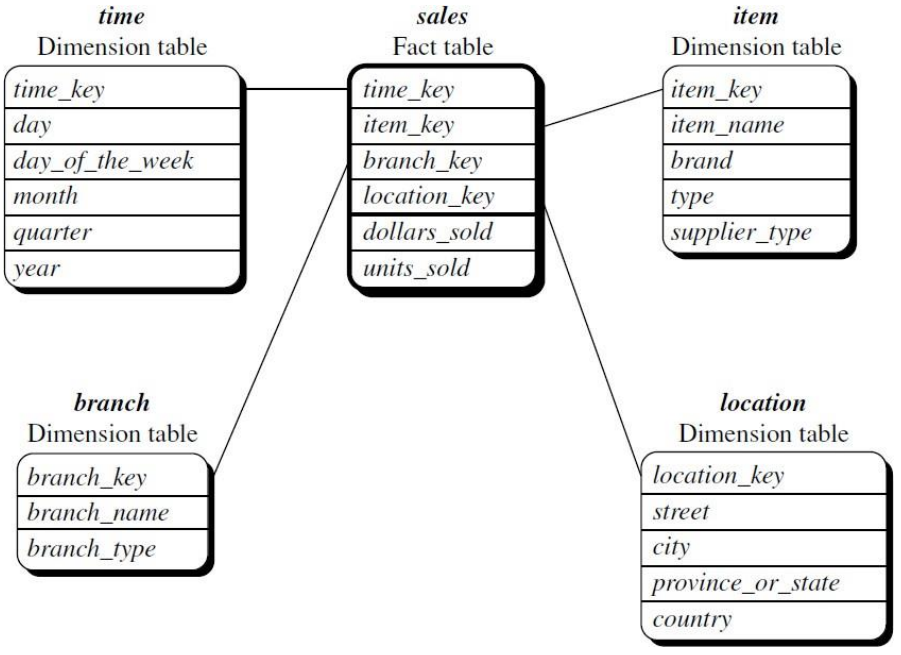


Figure Star schema of *sales* data warehouse.

Notice that in the star schema, each dimension is represented by only one table, and each table contains a set of attributes. For example, the *location* dimension table contains the attribute set f*location key, street, city,*

*province or state, country*g. This constraint may introduce some redundancy. For example, "Urbana" and "Chicago" are both cities in the state of Illinois, USA. Entries for such cities in the *location* dimension table will create redundancy among the attributes *province or state* and *country*; that is,.... , Urbana, IL, USA/ and.... , Chicago, IL, USA/.Moreover, the attributes within a dimension table may form either a hierarchy (total order) or a lattice (partial order).

Snowflake schema: The snowflake schema is a variant of the star schema model, where some dimension tables are *normalized*, thereby further splitting the data into additional tables. The resulting schema graph forms a shape similar to a snowflake.
The major difference between the snowflake and star schema models is that the dimension tables of the snowflake model may be kept in normalized form to reduce redundancies. Such a table is easy to maintain and saves storage space. However, this space savings is negligible in comparison to the typical magnitude of the fact table. Furthermore, the snowflake structure can reduce the effectiveness of browsing, since more joins will be needed to execute a query. Consequently, the system performance may be adversely impacted.Hence, although the snowflake schema reduces redundancy, it is not as popular as the star schema in data warehouse design.

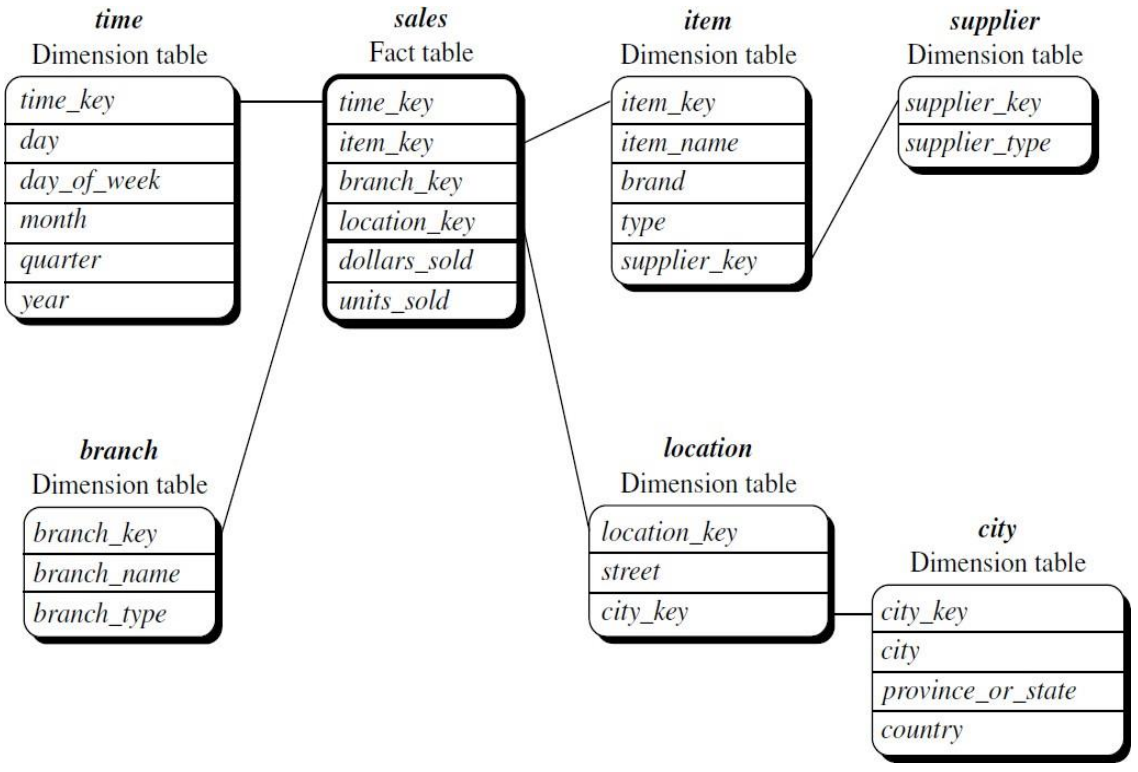


Figure Snowflake schema of a *sales* data warehouse.

EX. 2: Snowflake schema. A snowflake schema for *AllElectronics* sales is given in the above Figure. Here, the *sales* fact table is identical to that of the star schema in Figure 4.6. The main difference between the two schemas is in the definition of dimension tables. The single dimension table for *item* in the star schema is normalized in the snowflake schema, resulting in new *item* and *supplier* tables. For example, the *item* dimension table now contains the attributes *item key*, *item name*, *brand*, *type*, and *supplier key*, where

supplier key is linked to the *supplier* dimension table, containing *supplier key* and *supplier type* information. Similarly, the single dimension table for *location* in the star schema can be normalized into two new tables: *location* and *city*. The *city key* in the new *location* table links to the *city* dimension. Notice that, when desirable, further normalization can be performed on *province or state* and *country* in the snowflake schema.

Fact constellation: Sophisticated applications may require multiple fact tables to *share* dimension tables. This kind of schema can be viewed as a collection of stars, and hence is called a **galaxy schema** or a **fact constellation**.

EX 3: Fact constellation. A fact constellation schema is shown in Figure 4.8. This schema specifies two fact tables, *sales* and *shipping*. The *sales* table definition is identical to that of the star schema. The *shipping* table has five dimensions, or keys—*item key*, *time key*, *shipper key*, *from location*, and *to location*—and two measures—*dollars cost* and *units shipped*. A fact constellation schema allows dimension tables to be shared between fact tables. For example, the dimensions tables for *time*, *item*, and *location* are shared between the *sales* and *shipping* fact tables.

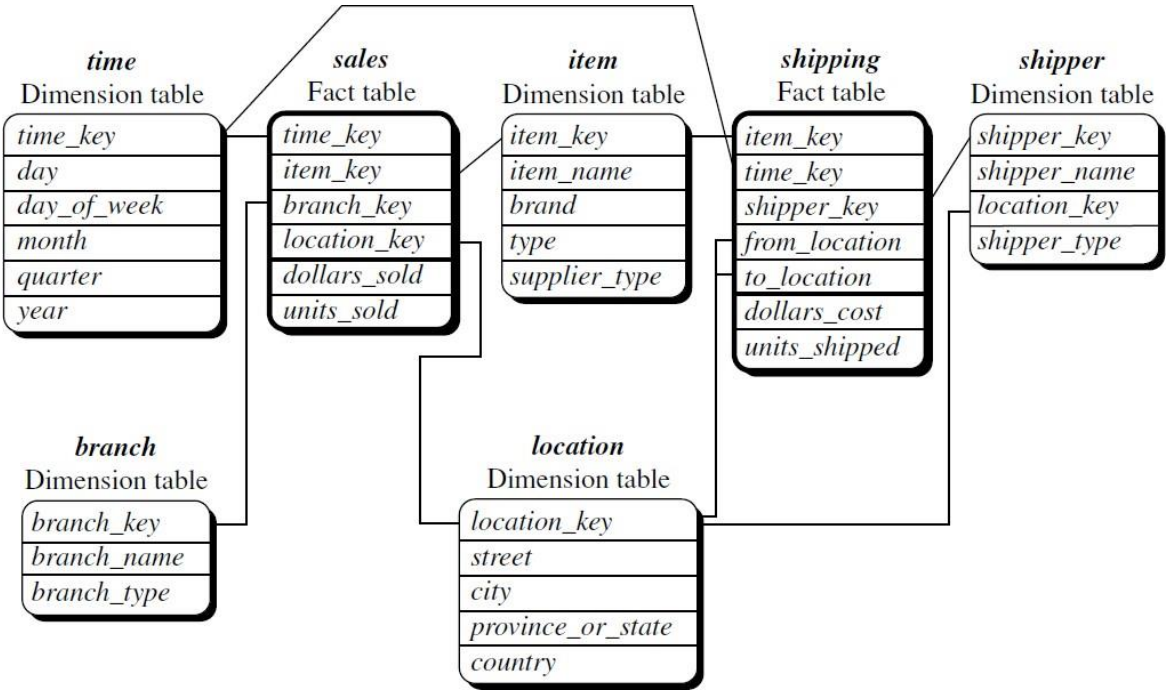


Figure Fact constellation schema of a sales and shipping data warehouse.

In data warehousing, there is a distinction between a data warehouse and a data mart. A data warehouse collects information about subjects that span the *entire organization*, such as *customers*, *items*, *sales*, *assets*, and *personnel*, and thus its scope is *enterprise-wide*. For data warehouses, the fact constellation schema is commonly used, since it can model multiple, interrelated subjects. A **data mart**, on the other hand, is a department subset of the data warehouse that focuses on selected subjects, and thus its scope is *departmentwide*. For data marts, the *star* or *snowflake* schema is commonly used, since both are geared toward modeling single subjects, although the star schema is more popular and efficient.

Concept Hierarchies

A **concept hierarchy** defines a sequence of mappings from a set of low-level concepts to higher-level, more general concepts. Consider a concept hierarchy for the dimension *location*. City values for *location* include Vancouver, Toronto, New York, and Chicago. Each city, however, can be mapped to the province or state to which it belongs. For example, Vancouver can be mapped to British Columbia, and Chicago to Illinois. The provinces and states can in turn be mapped to the country (e.g., Canada or the United States) to which they belong. These mappings form a concept hierarchy for the dimension *location*, mapping a set of low-level concepts (i.e., cities) to higher-level, more general concepts (i.e., countries). This concept hierarchy is illustrated in the bellow Figure.

Many concept hierarchies are implicit within the database schema. For example, suppose that the dimension *location* is described by the attributes *number*, *street*, *city*, *province or state*, *zip code*, and *country*. These attributes are related by a total order, forming a concept hierarchy such as "*street* < *city* < *province or state* < *country*." This hierarchy is shown in Figure (a). Alternatively, the attributes of a dimension may be organized in a partial order, forming a lattice. An example of a partial order for the *time* dimension based on the attributes *day*, *week*, *month*, *quarter*, and *year* is "*day* < f*month* < *quarter; week*g < *year*." This lattice structure is shown in Figure (b).

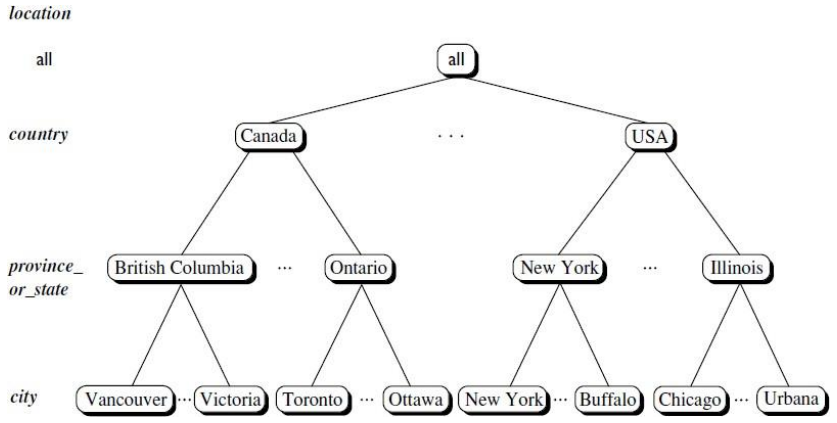


Figure A concept hierarchy for *location*. Due to space limitations, not all of the hierarchy nodes are shown, indicated by ellipses between nodes.

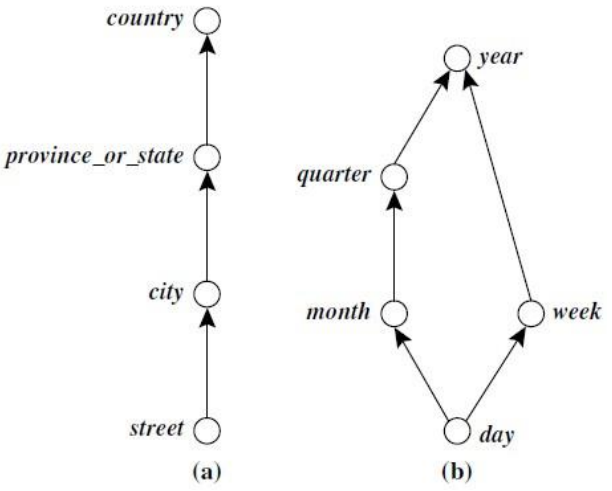


Figure Hierarchical and lattice structures of attributes in warehouse dimensions: (a) a hierarchy for *location* and (b) a lattice for *time*.

A concept hierarchy that is a total or partial order among attributes in a database schema is called a **schema hierarchy**. Concept hierarchies that are common to many applications (e.g., *for time*) may be predefined in the data mining system. Data mining systems should provide users with the flexibility to tailor predefined hierarchies according to their particular needs. For example, users may want to define a fiscal year starting on April 1 or an academic year starting on September 1.

Concept hierarchies may also be defined by discretizing or grouping values for a given dimension or attribute, resulting in a **set-grouping hierarchy**. A total or partial order can be defined among groups of values. An example of a set-grouping hierarchy is shown in thje bellow Figure for the dimension *price*, where an interval .$*X* : : :$*Y*] denotes the range from $*X* (exclusive) to $*Y* (inclusive).

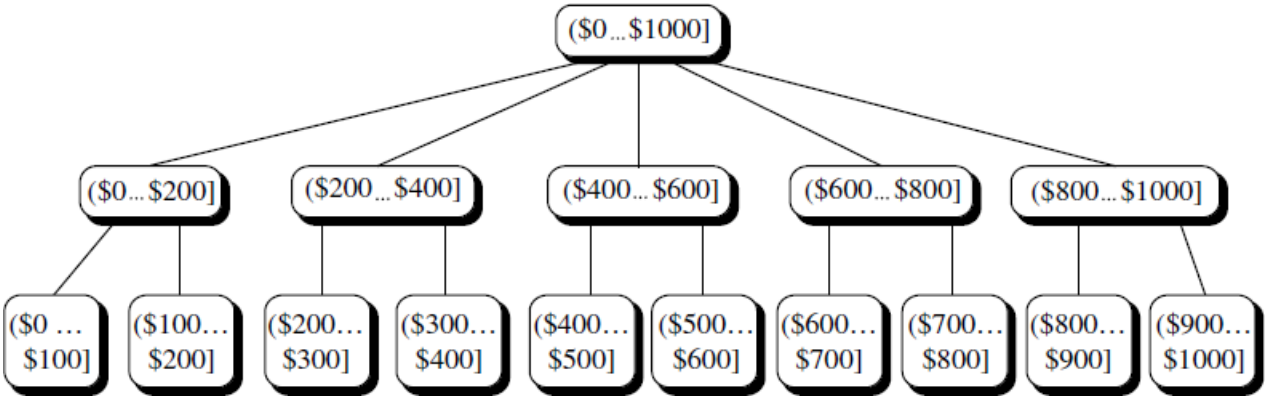


Figure : A concept hierarchy for *price*.

Measures:

A data cube **measure** is a numeric function that can be evaluated at each point in the data cube space. A measure value is computed for a given point by aggregating the data corresponding to the respective dimension–value pairs defining the given point. Measures can be organized into three categories—distributive, algebraic, and holistic— based on the kind of aggregate functions used.

Distributive: An aggregate function is *distributive* if it can be computed in a distributed manner as follows. Suppose the data are partitioned into *n* sets. We apply the function to each partition, resulting in *n* aggregate values. If the result derived by applying the function to the *n* aggregate values is the same as that derived by applying the function to the entire data set (without partitioning), the function can be computed in a distributed manner.
For example, sum() can be computed for a data cube by first partitioning the cube into a set of subcubes, computing sum() for each subcube, and then summing up the counts obtained for each subcube. Hence, sum() is a distributive aggregate function. For the same reason, count(), min(), and max() are distributive aggregate functions. By treating the count value of each nonempty base cell as 1 by default, count() of any cell in a cube can be viewed as the sum of the count values of all of its corresponding child cells in its subcube. Thus, count() is distributive. A measure is *distributive* if it is obtained by applying a distributive aggregate function. Distributive measures can be computed efficiently because of the way the computation can be partitioned.

Algebraic: An aggregate function is *algebraic* if it can be computed by an algebraic function with *M* arguments (where *M* is a bounded positive integer), each of which is obtained by applying a distributive aggregate function. For example, avg() (average) can be computed by sum()/count(), where both sum() and count() are distributive aggregate functions. Similarly, it can be shown that min N() and max N() (which find the *N* minimum and *N* maximum values, respectively, in a given set) and standard deviation() are algebraic aggregate functions. A measure is *algebraic* if it is obtained by applying an algebraic aggregate function.

Holistic: An aggregate function is *holistic* if there is no constant bound on the storage size needed to describe a subaggregate. That is, there does not exist an algebraic function with *M* arguments (where *M* is a constant) that characterizes the computation. Common examples of holistic functions include median(), mode(), and rank(). A measure is *holistic* if it is obtained by applying a holistic aggregate function.

Most large data cube applications require efficient computation of distributive and algebraic measures.

Data Warehouse Implementation

The big data which is to be analyzed and handled to draw insights from it will be stored in data warehouses. These warehouses are run by OLAP servers which require processing of a query with seconds. So, a data warehouse should need highly efficient cube computation techniques, access methods, and query processing techniques.

The core of multidimensional data analysis is the efficient computation of aggregations across many sets of dimensions.

In SQL aggregations are referred to as group-by's.

Each group-by can be represented as a cuboid.

Set of group-by's forms a lattice of a cuboid defining a data cube.

Efficient Data Cube Computation:

The compute cube operator computes aggregates over all subsets of the dimensions specified in the operation. It requires excessive storage space, especially for a large number of dimensions. A data cube is a lattice of cuboids.

Suppose that we create a data cube for ProElectronics(Company) sales that contains the following: city, item, year, and sales_in_dollars.

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?

Three attributes:
city, item, year (dimensions), sales_in_dollars (measure).

The total number of cuboids or group-by's computed for this cube is 2^3=8.

Group-by's: {(city,item,year), (city, item), (city, year), (item, year), (city), (item), (year),()}.
() : group-by is empty i.e. the dimensions are not grouped. The base cuboid contains all three dimensions. Apex cuboid is empty.

On-line analytical processing may need to access different cuboids for different queries. A major challenge related to precomputation would be storage space if all the cuboids in the data cube are computed, especially when the cube has many dimensions.The storage requirements are even more

excessive when many of the dimensions have associated concept hierarchies, each with multiple levels.This problem is referred to as the **Curse of Dimensionality**.

Cube Operation:

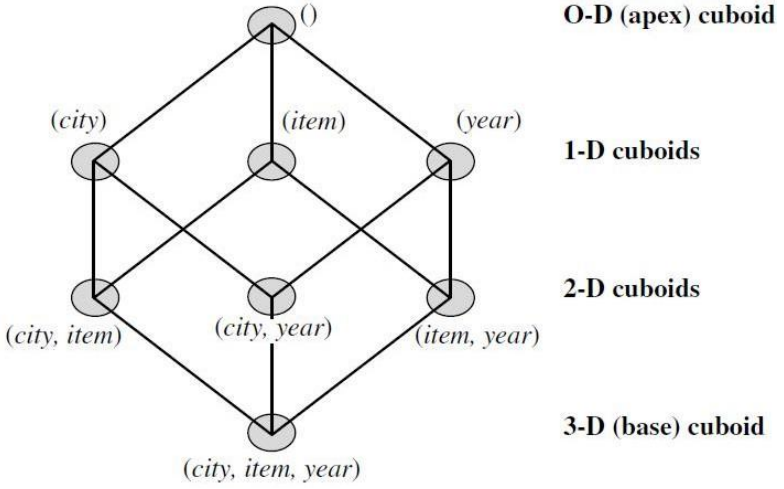


Figure Lattice of cuboids, making up a 3-D data cube. Each cuboid represents a different group-by. The base cuboid contains *city, item,* and *year* dimensions.

Cube definition and computation in DMQL

- define cube sales_cube[city, item, year] (sales_in_dollars)
- compute cube sales_cube

Transform it into a SQL-like language (with a new operator cube by, introduced by Gray et al.'96)

- SELECT item, city, year, SUM (amount) FROM SALES CUBE BY item, city, year

Data cube can be viewed as a lattice of cuboids

- The bottom-most cuboid is the base cuboid.
- The top-most cuboid (apex) contains only one cell.
- How many cuboids in an n-dimensional cube with L levels? ($T=\Pi(Li+1)$)

Partial Materialization: Selected Computation of Cuboids

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

1. No materialization: Do not precompute any of the "nonbase" cuboids. This leads to computing expensive multidimensional aggregates on-the-fly, which can be extremely slow.

2. Full materialization: Precompute 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. We will use the term *subcube* to refer to the latter case, where only some of the cells may be precomputed for various cuboids. Partial materialization represents an interesting trade-off between storage space and response time.

The partial materialization of cuboids or subcubes should consider three factors: (1) identify the subset of cuboids or subcubes to materialize; (2) exploit the materialized cuboids or subcubes during query processing; and (3) efficiently update the materialized cuboids or subcubes during load and refresh.

OLAP Operations

"How are concept hierarchies useful in OLAP?" In the multidimensional model, data are organized into multiple dimensions, and each dimension contains multiple levels of abstraction defined by concept hierarchies. This organization provides users with the flexibility to view data from different perspectives. A number of OLAP data cube operations exist to materialize these different views, allowing interactive querying and analysis of the data at hand. Hence, OLAP provides a user-friendly environment for interactive data analysis.

Example of OLAP operations. Let's look at some typical OLAP operations for multidimensional data. Each of the following operations described is illustrated in the bellow Figure. At the center of the figure is a data cube for *AllElectronics* sales. The cube contains the dimensions *location*, *time*, and *item*, where *location* is aggregated with respect to city values, *time* is aggregated with respect to quarters, and *item* is aggregated with respect to item types. To aid in our explanation, we refer to this cube as the central cube. The measure displayed is *dollars sold* (in thousands). (For improved readability, only some of the cubes' cell values are shown.) The data examined are for the cities Chicago, New York, Toronto, and Vancouver.

Roll-up: The roll-up operation (also called the *drill-up* operation by some vendors) performs aggregation on a data cube, either by *climbing up a concept hierarchy* for a dimension or by *dimension reduction*. Figure 4.12 shows the result of a roll-up operation performed on the central cube by climbing up the concept hierarchy for *location* given in Figure 4.9. This hierarchy was defined as the total order "*street* < *city* <

province or state < *country*." The roll-up operation shown aggregates the data by ascending the *location* hierarchy from the level of *city* to the level of *country*. In other words, rather than grouping the data by city, the resulting cube groups the data by country.

When roll-up is performed by dimension reduction, one or more dimensions are removed from the given cube. For example, consider a sales data cube containing only the *location* and *time* dimensions. Roll-up may be performed by removing, say, the *time* dimension, resulting in an aggregation of the total sales by location, rather than by location and by time.

Drill-down: Drill-down is the reverse of roll-up. It navigates from less detailed data to more detailed data. Drill-down can be realized by either *stepping down a concept hierarchy* for a dimension or *introducing additional dimensions*. Figure shows the result of a drill-down operation performed on the central cube by stepping down a concept hierarchy for *time* defined as "*day* < *month* < *quarter* < *year*."

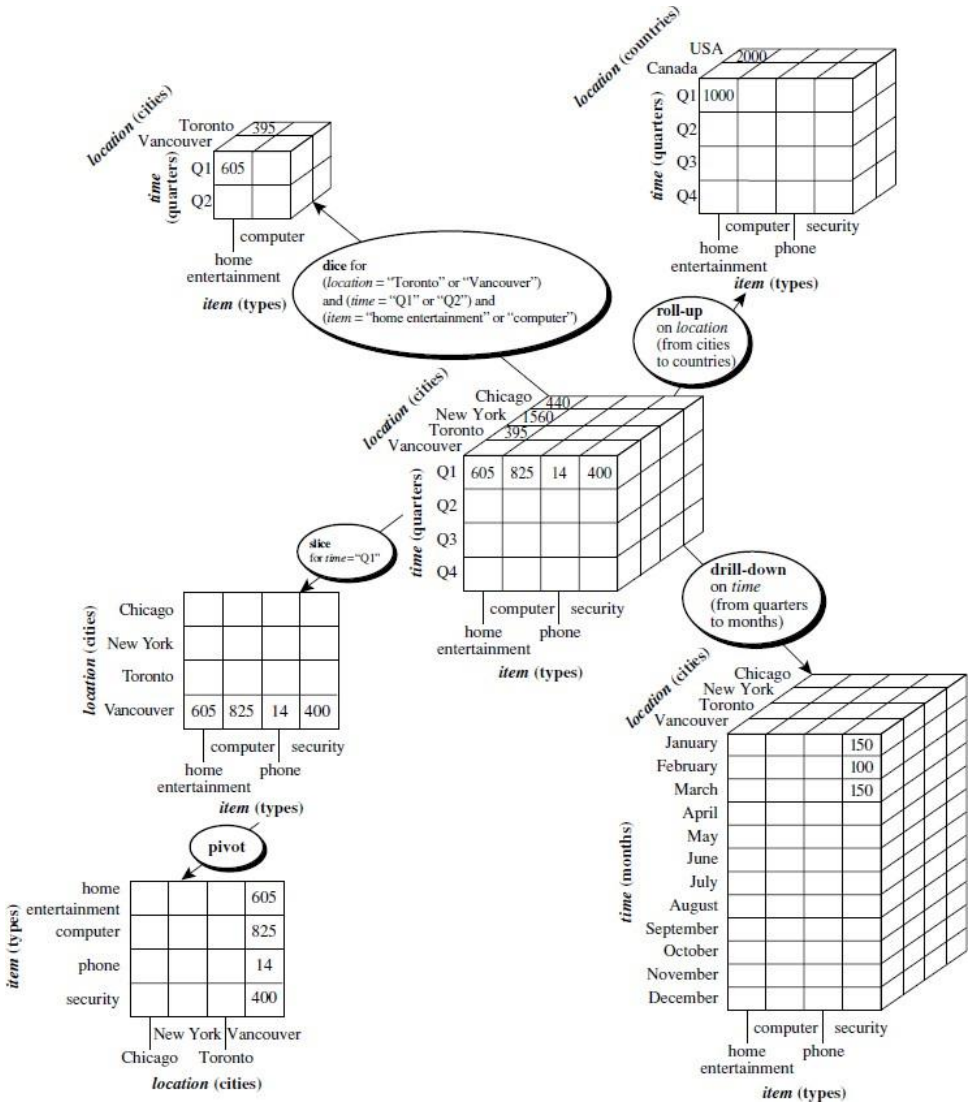


Figure Examples of typical OLAP operations on multidimensional data.

Drill-down occurs by descending the *time* hierarchy from the level of *quarter* to the more detailed level of *month*. The resulting data cube details the total sales per month rather than summarizing them by quarter.

Because a drill-down adds more detail to the given data, it can also be performed by adding new dimensions to a cube. For example, a drill-down on the central cube of the above Figure can occur by introducing an additional dimension, such as *customer group.*

Slice and dice: The *slice* operation performs a selection on one dimension of the given cube, resulting in a subcube. Above figure shows a slice operation where the sales data are selected from the central cube for the dimension *time* using the criterion *time* D "Q1." The *dice* operation defines a subcube by performing a selection on two or more dimensions. Figure shows a dice operation on the central cube based on the following selection criteria that involve three dimensions: (*location* D "Toronto" or "Vancouver") and (*time* D "Q1" or "Q2") and (item D "home entertainment" or "computer").

Pivot (rotate): *Pivot* (also called *rotate*) is a visualization operation that rotates the data axes in view to provide an alternative data presentation. The above Figure shows a pivot operation where the *item* and *location* axes in a 2-D slice are rotated. Other examples include rotating the axes in a 3-D cube, or transforming a 3-D cube into a series of 2-D planes.

Other OLAP operations: Some OLAP systems offer additional drilling operations. For example, **drill-across** executes queries involving (i.e., across) more than one fact table. The **drill-through** operation uses relational SQL facilities to drill through the bottom level of a data cube down to its back-end relational tables.

Other OLAP operations may include ranking the top *N* or bottom *N* items in lists, as well as computing moving averages, growth rates, interests, internal return rates, depreciation, currency conversions, and statistical functions.

OLAP offers analytical modeling capabilities, including a calculation engine for deriving ratios, variance, and so on, and for computing measures acrossmultiple dimensions. It can generate summarizations, aggregations, and hierarchies at each granularity level and at every dimension intersection. OLAP also supports functional models for forecasting, trend analysis, and statistical analysis. In this context, an OLAP engine is a powerful data analysis tool.

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.
Many MOLAP servers adopt a two-level storage representation to handle dense and sparse data sets: Denser subcubes are identified and stored as array structures, whereas sparse subcubes employ compression technology for efficient storage utilization.

Hybrid OLAP (HOLAP) servers:

The hybrid OLAP approach combines ROLAP and MOLAP technology, benefiting from the greater calability 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.

Example of A ROLAP data store. Table bellow shows a summary fact table that contains both base fact data and aggregated data. The schema is "*record identifier (RID), item, . . . , day, month,quarter, year, dollars sold*i," where *day*, *month*, *quarter*, and *year* define the sales date, and *dollars sold* is the sales amount. Consider the tuples with an *RID* of 1001 and 1002, respectively. The data of these tuples are at the base fact level, where the sales dates are October 15, 2010, and October 23, 2010, respectively. Consider the tuple with an *RID* of 5001. This tuple is at a more general level of abstraction than the tuples 1001 and 1002. The *day* value has been generalized to all, so that the corresponding *time* value is October 2010. That is, the *dollars sold* amount shown is an aggregation representing the entire month of October 2010, rather than just October 15 or 23, 2010. The special value all is used to represent subtotals in summarized data.

MOLAP uses multidimensional array structures to store data for online analytical processing. Most data warehouse systems adopt a client-server architecture. A relational data store always resides at the data warehouse/data mart server site. A multidimensional data store can reside at either the database server site or the client site.

Single Table for Base and Summary Facts

RID	item	...	day	month	quarter	year	dollars_sold
1001	TV	...	15	10	Q4	2010	250.60
1002	TV	...	23	10	Q4	2010	175.00
...	...	...	...	...	...	...	...
5001	TV	...	all	10	Q4	2010	45,786.08
...	...	...	...	...	...	...	...

Difference between ROLAP and MOLAP:

S.NO	ROLAP	MOLAP
1.	ROLAP stands for **Relational Online Analytical Processing.**	While MOLAP stands for **Multidimensional Online Analytical Processing.**
2.	ROLAP is used for large data volumes.	While it is used for limited data volumes.
3.	The access of ROLAP is slow.	While the access of MOLAP is fast.
4.	In ROLAP, Data is stored in relation tables.	While in MOLAP, Data is stored in multidimensional array.
5.	In ROLAP, Data is fetched from data-warehouse.	While in MOLAP, Data is fetched from MDDBs database.
6.	In ROLAP, Complicated sql queries are used.	While in MOLAP, Sparse matrix is used.
7.	In ROLAP, Static multidimensional view of data is created.	While in MOLAP, Dynamic multidimensional view of data is created.

Indexing OLAP Data: Bitmap Index and Join Index

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 of 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.

Bitmap indexing is advantageous compared to hash and tree indices. It is especially useful for low-cardinality domains because comparison, join, and aggregation operations are then reduced to bit arithmetic, which substantially reduces the processing time. Bitmap indexing leads to significant reductions in space and input/output (I/O) since a string of characters can be represented by a single bit. For higher-cardinality domains, the method can be adapted using compression techniques.

The **join indexing** method gained popularity fromits use in relational database query processing. Traditional indexing maps the value in a given column to a list of rows having that value. In contrast, 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.

The star schema model of data warehouses makes join indexing attractive for crosstable search, because the linkage between a fact table and its corresponding dimension tables comprises the fact table's foreign key and the dimension table's primary key. Join indexing maintains relationships between attribute values of a dimension (e.g., within a dimension table) and the corresponding rows in the fact table. Join indices may span multiple dimensions to form **composite join indices**.We can use join indices to identify subcubes that are of interest.

Example of Join indexing. We defined a star schema for *AllElectronics* of the form "*sales star* [*time, item, branch, location*]: *dollars sold* D = 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 Figure1. 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 Figure2.

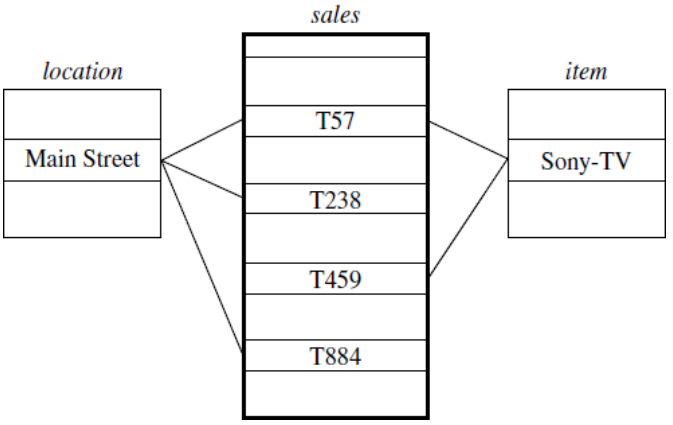


Figure1 Linkages between a *sales* fact table and *location* and *item* dimension tables.

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
. . .	. . .	. . .

Figure2 Join index tables based on the linkages between the *sales* fact table and the *location* and *item* dimension tables shown in the above Figure.

Suppose that there are 360 time values, 100 items, 50 branches, 30 locations, and 10 million sales tuples in the *sales star* data cube. If the *sales* fact table has recorded sales for only 30 items, the remaining 70 items will obviously not participate in joins. If join indices are not used, additional I/Os have to be performed to bring the joining portions of the fact table and the dimension tables together.

To further speed up query processing, the join indexing and the bitmap indexing methods can be integrated to form **bitmapped join indices**.

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.

Example of OLAP query processing. Suppose that we define a data cube for *AllElectronics* of the form"*sales cube* [*time, item, location*]: sum*(sales in dollars)*." The dimension hierarchies used are "*day* < *month* < *quarter* < *year*" for *time*; "*item name* < *brand* < *type*" for *item*; and "*street* < *city* < *province or state* < *country*" for *location*.
Suppose that the query to be processed is on f*brand, province or state*g, with the selection constant "*year = 2010*." Also, suppose that there are four materialized cuboids available, as follows:

cuboid 1: f*year, item name, city*g
cuboid 2: f*year, brand, country*g
cuboid 3: f*year, brand, province or state*g
cuboid 4: f*item name, province or state*g, where *year* = 2010

"Which of these four cuboids should be selected to process the query?"

Finer-granularity data cannot be generated from coarser-granularity data. Therefore, cuboid 2 cannot be used because *country* is a more general concept than *province or state*. Cuboids 1, 3, and 4 can be used to process the query because (1) they have the same set or a superset of the dimensions in the query, (2) the selection clause in the query can imply the selection in the cuboid, and (3) the abstraction levels for the *item* and *location* dimensions in these cuboids are at a finer level than *brand* and *province or state*, respectively.

"How would the costs of each cuboid compare if used to process the query?"

It is likely that using cuboid 1 would cost the most because both *item name* and *city* are at a lower level than the *brand* and *province or state* concepts specified in the query. If there are not many *year* values associated with *items* in the cube, but there are several *item names* for each *brand*, then cuboid 3 will be smaller than cuboid 4, and thus cuboid 3 should be chosen to process the query. However, if efficient indices are available for cuboid 4, then cuboid 4 may be a better choice. Therefore, some cost-based estimation is required to decide which set of cuboids should be selected for query processing.