Unit-II

Data Warehousing and Online Analytical Processing: Basic Concepts

What Is a DataWarehouse?

Loosely speaking, a data warehouse refers to a data repository that is maintained separately from an organization's operational databases.

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".

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.

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*.

data warehousing as the process of constructing and using data warehouses. The construction of a data warehouse requires data cleaning, data integration, and data consolidation.

"*How are organizations using the information from data warehouses?*" Many organizations use this information to support business decision-making activities, including (1) increasing customer focus, which includes the analysis of customer buying patterns (such as buying preference, buying time, budget cycles, and appetites for
spending); (2) repositioning products and managing product portfolios by comparing the performance of sales by quarter, by year, and by geographic regions in order to fine-tune production strategies; (3) analyzing operations and looking for sources of profit; and (4) managing customer relationships, making environmental corrections, and managing the cost of corporate assets.

The traditional database approach to heterogeneous database integration is to build **wrappers** and **integrators** (or **mediators**) on top of multiple, heterogeneous databases. When a query is posed to a client site, a metadata dictionary is used to translate the query into queries appropriate for the individual heterogeneous sites involved. These queries are then mapped and sent to local query processors. The results returned from the different sites are integrated into a global answer set. This **query-driven approach** requires complex information filtering and integration processes, and competes with local sites for processing resources. It is inefficient and potentially expensive for frequent queries, especially queries requiring aggregations.

Data warehousing provides an interesting alternative to this traditional approach. Rather than using a query-driven approach, data warehousing employs an **update-driven** approach in which information from multiple, heterogeneous sources is integrated in advance and stored in a warehouse for direct querying and analysis.

Differences between Operational Database Systems and Data Warehouses

The major task of online operational database systems is to perform online transaction and query processing. These systems are called **online transaction processing (OLTP)** systems. They cover

most of the day-to-day operations of an organization such as purchasing, inventory, manufacturing, banking, payroll, registration, and accounting. Data warehouse systems, on the other hand, serve users or knowledge workers in the role of data analysis and decision making. Such systems can organize and present data in various formats in order to accommodate the diverse needs of different users. These systems are known as **online analytical processing (OLAP)** systems.
The major distinguishing features of OLTP and OLAP are summarized as follows:

Users and system orientation: An OLTP system is *customer-oriented* and is used for transaction and query processing by clerks, clients, and information technology professionals. An OLAP system is *market-oriented* and is used for data analysis by knowledge workers, including managers, executives, and analysts.

Data contents: An OLTP systemmanages current data that, typically, are too detailed to be easily used for decision making. An OLAP system manages large amounts of historic data, provides facilities for summarization and aggregation, and stores and manages information at different levels of granularity.

Database design: An OLTP system usually adopts an entity-relationship (ER) data model and an application-oriented database design. An OLAP system typically adopts either a *star* or a *snowflake* model and a subject-oriented database design.

View: An OLTP system focuses mainly on the current data within an enterprise or department, without referring to historic data or data in different organizations. In contrast, an OLAP system often spans multiple versions of a database schema, due to the evolutionary process of an organization.

Access patterns: The access patterns of an OLTP system consist mainly of short, atomic transactions. Such a systemrequires concurrency control and recovery mechanisms. However, accesses to OLAP systems are mostly read-only operations (because most data warehouses store historic rather than up-to-date information), although many could be complex queries.

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

But, Why Have a Separate Data Warehouse?

"Why not perform online analytical processing directly on such databases instead of spending additional time and resources to construct a separate data warehouse?"

A major reason for such a separation is to help promote the *high performance of both systems*. An operational database is designed and tuned from known tasks and workloads like indexing and hashing using primary keys, searching for particular records, and optimizing "canned" queries. On the other hand, data warehouse queries are often complex. They involve the computation of large data groups at summarized levels, and may require the use of special data organization, access, and implementation methods based on multidimensional views. Processing OLAP queries in operational databases would substantially degrade the performance of operational tasks.

Moreover, an operational database supports the concurrent processing of multiple transactions. Concurrency control and recovery mechanisms (e.g., locking and logging) are required to ensure the consistency and robustness of transactions. An OLAP query often needs read-only access of data records for summarization and aggregation. Decision support requires historic data, whereas operational databases do not typically maintain historic data.

In this context, the data in operational databases, though abundant, are usually far from complete for decision making. Decision support requires consolidation (e.g., aggregation and summarization) of data from heterogeneous sources, resulting in high-quality, clean, integrated data. In contrast, operational databases contain only detailed raw data, such as transactions, which need to be consolidated before analysis.

Data Warehousing: A Multitiered Architecture

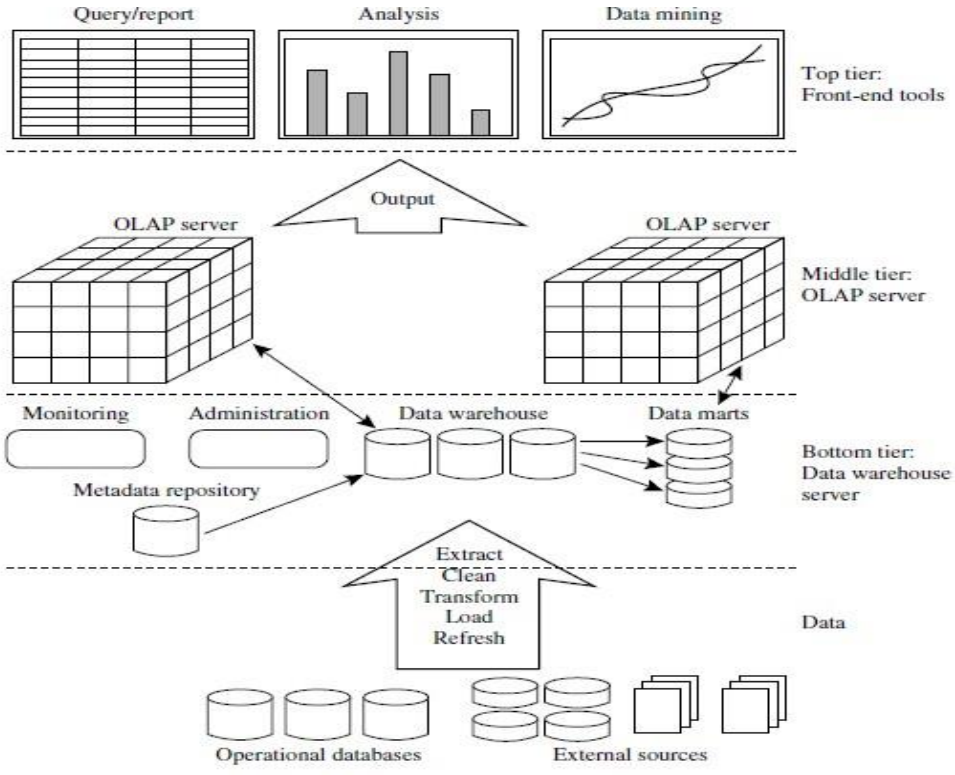

Figure 4.1 A three-tier data warehousing architecture.

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. 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. The data are extracted using application program interfaces known as **gateways**. A gateway is supported by the underlying DBMS and allows client programs to generate SQL code to be executed at a server. Examples of gateways include ODBC (Open Database Connection) and OLEDB (Object Linking and Embedding Database) by Microsoft and JDBC (Java Database Connection). 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 **relationalOLAP(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).

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.

Data marts are two types. They are

1. Independent data mart
2. Dependent data mart

1. *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.
2. *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.

"What are the pros and cons of the top-down and bottom-up approaches to data warehouse development?"

top-down development of an enterprise warehouse

pros

minimizes integration problems

cons

1. it is expensive
2. it takes a long time to develop,
3. lacks flexibility

bottom-up approach

pros

flexibility, low cost, and rapid return of investment.

Cons

lead to problems when integrating various disparate data marts into a consistent enterprise data warehouse.

A recommended method for the development of data warehouse systems is to implement the warehouse in an incremental and evolutionary manner. First, a high-level corporate data model is defined within a reasonably short period (such as one or two months) that provides a corporate-wide, consistent, integrated view of data among different subjects and potential usages. Second, independent data marts can be implemented in parallel with the enterprise warehouse based on the same corporate data model set noted before. Third, distributed data marts can be constructed to integrate different data marts via hub servers. Finally, a **multitier data warehouse** is constructed where the enterprise warehouse is the sole custodian of all warehouse data, which is then distributed to the various dependent data marts.

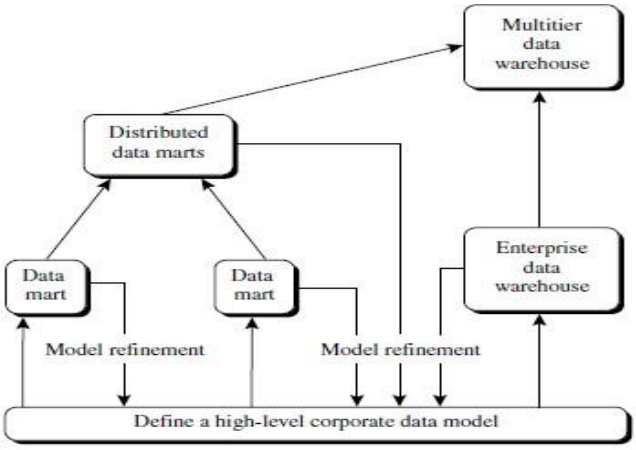

Figure 4.2 A recommended approach for data warehouse development.

Extraction, Transformation, and Loading

Data warehouse systems use back-end tools and utilities to populate and refresh their data. These tools and utilities include the following functions:

Data extraction, which typically gathers data from multiple, heterogeneous, and external sources.
Data cleaning, which detects errors in the data and rectifies them when possible.
Data transformation, which converts data from legacy or host format to warehouse format.
Load, which sorts, summarizes, consolidates, computes views, checks integrity, and builds indices and partitions.
Refresh, which propagates the updates from the data sources to the warehouse.

Metadata Repository

Metadata are data about data. When used in a data warehouse, metadata are the data that define warehouse objects. Metadata are created for the data names and definitions of the given warehouse.

Additional metadata are created and captured for timestamping any extracted data, the source of the extracted data, and missing fields that have been added by data cleaning or integration processes.
A metadata repository should contain the following:

- A description of the *data warehouse structure*, which includes the warehouse schema, view, dimensions, hierarchies, and derived data definitions, as well as data mart locations and contents.
- *Operational metadata*, which include data lineage (history of migrated data and the sequence of transformations applied to it), currency of data (active, archived, or purged), and monitoring information (warehouse usage statistics, error reports, and audit trails).
- The *algorithms used for summarization*, which include measure and dimension definition algorithms, data on granularity, partitions, subject areas, aggregation, summarization, and predefined queries and reports.
- *Mapping from the operational environment to the data warehouse*, which includes source databases and their contents, gateway descriptions, data partitions, data extraction, cleaning, transformation rules and defaults, data refresh and purging rules, and security (user authorization and access control).
- *Data related to system performance*, which include indices and profiles that improve data access and retrieval performance, in addition to rules for the timing and scheduling of refresh, update, and replication cycles.
- *Business metadata*, which include business terms and definitions, data ownership information, and charging policies.

A data warehouse contains different levels of summarization, of which metadata is one. Other types include current detailed data (which are almost always on disk), older detailed data (which are usually on tertiary storage), lightly summarized data, and highly summarized data (which may or may not be physically housed).

DataWarehouse Modeling: Data Cube and OLAP

Data warehouses and OLAP tools are based on a **multidimensional data model**. This model views data in the form of a *data cube*.

Data Cube: A Multidimensional Data Model

"What is a data cube?" A **data cube** allows data to be modeled and viewed in multiple dimensions. It is defined by dimensions and facts.
In general terms, **dimensions** are the perspectives or entities with respect to which an organization wants to keep records. For example, *AllElectronics* may create a *sales* data warehouse in order to keep records of the store's sales with respect to the dimensions *time*, *item*, *branch*, and *location*. Each dimension may have a table associated with it, called a **dimension table**, which further describes the dimension. For example, a dimension table for *item* may contain the attributes *item name*, *brand*, and *type*.
A multidimensional data model is typically organized around a central theme, such as *sales*. This theme is represented by a fact table. **Facts** are numeric measures. Examples of facts for a sales data warehouse include *dollars_sold* (sales amount in dollars), *units_sold* (number of units sold), and *amount_budgeted*. The **fact table** contains the names of the *facts*, or measures, as well as keys to each of the related dimension tables.
In 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). 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. 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, as shown below,

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

Note: The sales are from branches located in the city of Vancouver. The measure displayed is *dollars_sold* (in thousands).

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

Note: The measure displayed is *dollars_sold* (in thousands).

4D cube 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.

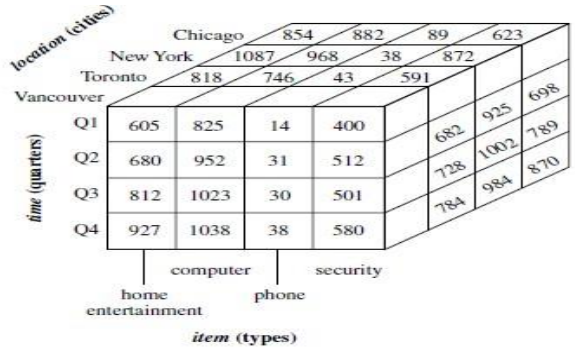

Figure 4.3 A 3-D data cube representation of the data in Table 4.3, according to *time*, *item*, and *location*. The measure displayed is *dollars_sold* (in thousands).

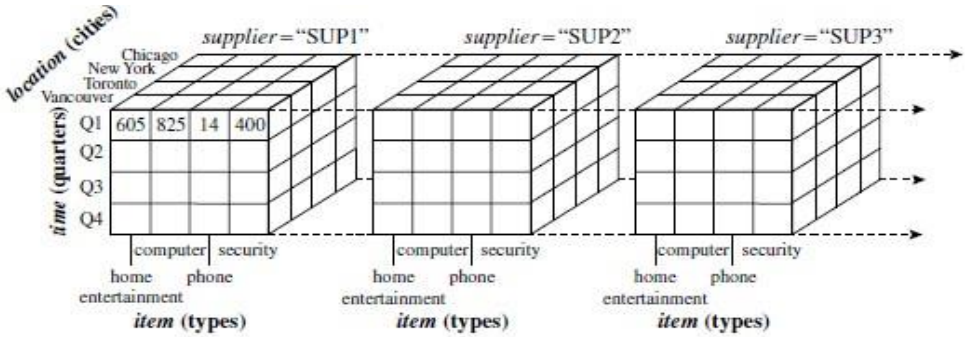

Figure 4.4 A 4-D data cube representation of sales data, according to *time*, *item*, *location*, and *supplier*. The measure displayed is *dollars_sold* (in thousands). For improved readability, only some of the cube values are shown.

lattice of cuboids

The following figure shows the forming of 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.

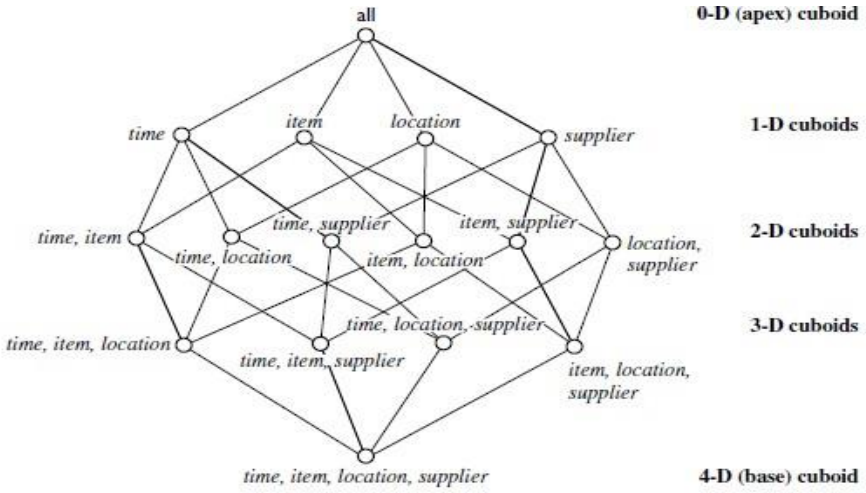

Figure 4.5 Lattice of cuboids, making up a 4-D data cube for *time*, *item*, *location*, and *supplier*. Each cuboid represents a different degree of summarization.

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

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**.

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.

Example 4.1 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.

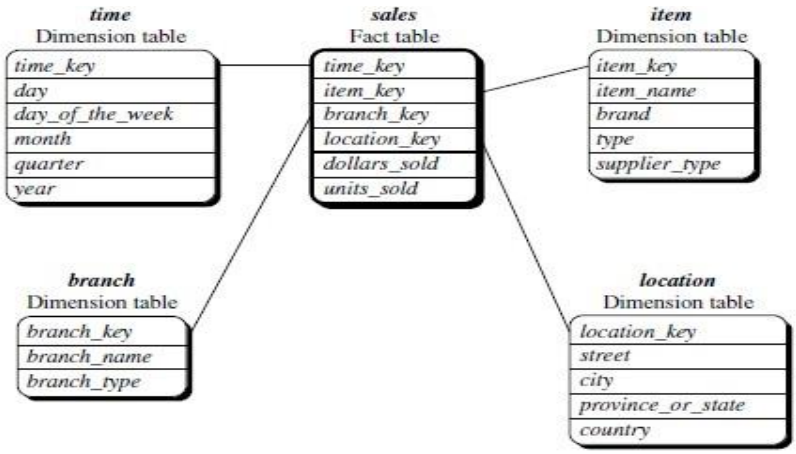

Figure 4.6 Star schema of *sales* data warehouse.

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 {*location_key*, *street*, *city*, *province_or_state*, *country*}. 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 major difference between the snowflake and star schema models is

1. Dimension tables of the snowflake model may be kept in normalized form to reduce redundancies.
2. Table is easy to maintain.
3. Saves storage space.

Disadvantage

1. The snowflake structure can reduce the effectiveness of browsing.
2. More joins will be needed to execute a query.
3. 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.

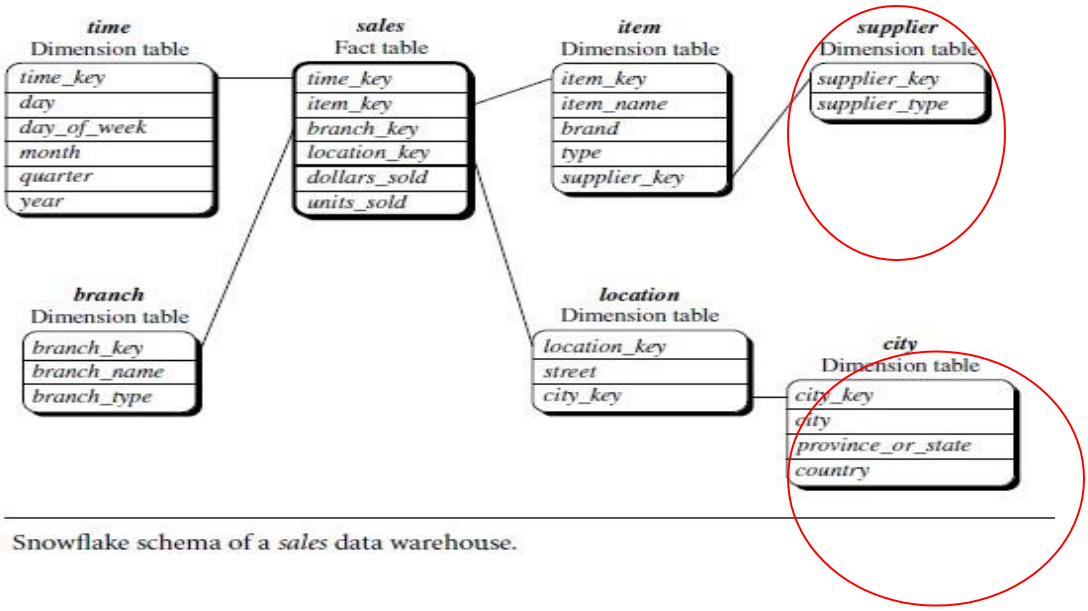

Figure 4.7 Snowflake schema of a *sales* data warehouse.

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.

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**.

Example 4.3 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 (Figure 4.6). 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.

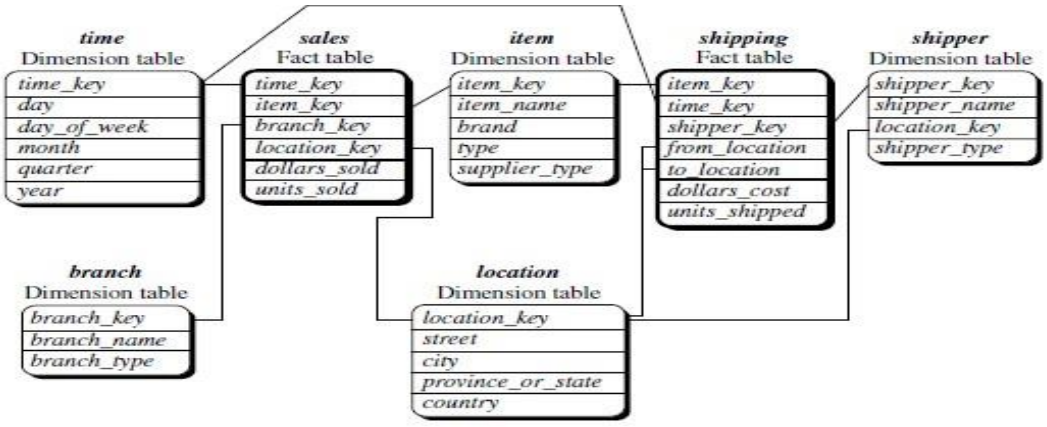

Figure 4.8 Fact constellation schema of a sales and shipping data warehouse.

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.

Dimensions: The Role of 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).

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 4.10(a). Alternatively, the attributes of a dimension may

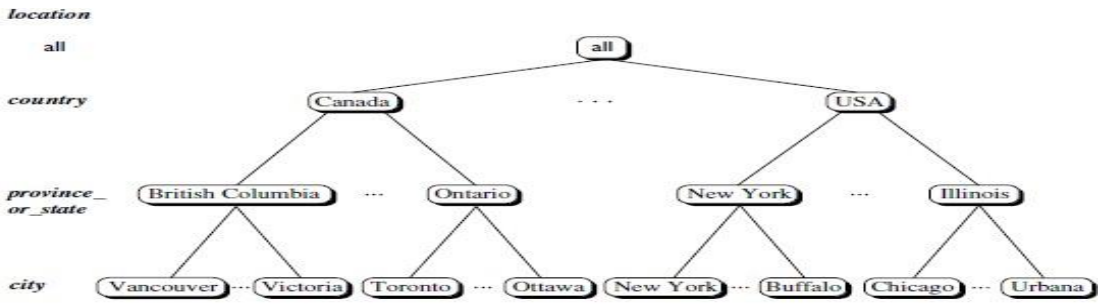

Figure 4.9 A concept hierarchy for *location*. Due to space limitations, not all of the hierarchy nodes are shown, indicated by ellipses between nodes.

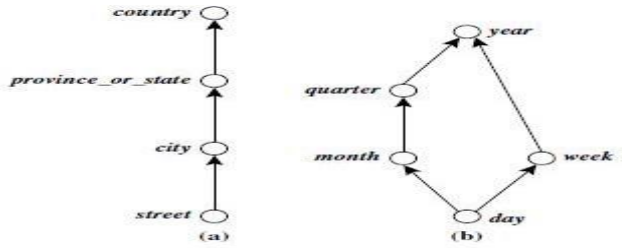

Figure 4.10 Hierarchical and lattice structures of attributes in warehouse dimensions: (a) a hierarchy for *location* and (b) a lattice for *time*.

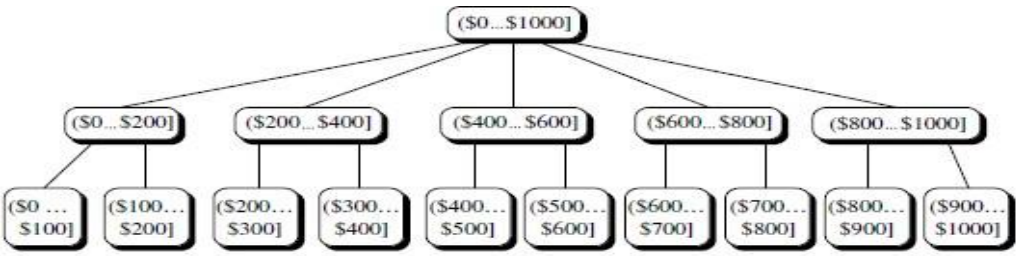

Figure 4.11 A concept hierarchy for *price*.

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* < {*month* < *quarter; week*} < *year*."1 This lattice structure is shown in Figure 4.10(b). 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.

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 Figure 4.11 for the dimension *price*, where an interval ($*X* ...$*Y*] denotes the range from $*X* (exclusive) to $*Y* (inclusive).

There may be more than one concept hierarchy for a given attribute or dimension, based on different user viewpoints. For instance, a user may prefer to organize *price* by defining ranges for *inexpensive, moderately priced*, and *expensive*.

Concept hierarchies may be provided manually by system users, domain experts, or knowledge engineers, or may be automatically generated based on statistical analysis of the data distribution.

Measures: Their Categorization and Computation

"How are measures computed?"

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.

for example, <*time* = "Q1", *location* = "Vancouver", *item* = "computer">. – set of dimension- value pairs

Measures can be organized into three categories—distributive, algebraic, and holistic— based on the kind of aggregate functions used.

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. 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.

A measure is ***algebraic*** if it is obtained by applying an algebraic aggregate function. 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 ***holistic*** if it is obtained by applying a holistic aggregate function.
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().

Most large data cube applications require efficient computation of distributive and algebraic measures. Many efficient techniques for this exist. In contrast, it is difficult to compute holistic measures efficiently. Efficient techniques to *approximate* the computation of some holistic measures, however, do exist.

Typical 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.

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*.
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*
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*. concept hierarchy for *time* defined as "*day* < *month* < *quarter* < *year*." Drill-down occurs by descending the *time* hierarchy fromthe level of *quarter* to the more detailed level of *month.* Because a drill-down adds more detail to the given data, it can also be performed by adding new dimensions to a cube.

Slice and dice: The *slice* operation performs a selection on one dimension of the given cube, resulting in a subcube. Figure 4.12 shows a slice operation where the sales data are selected from the central cube for the dimension *time* using the criterion *time* = "Q1." The *dice* operation defines a subcube by performing a selection on two or more dimensions. Figure 4.12 shows a dice operation on the central cube based on the following selection criteria that involve three dimensions: (*location* = "Toronto" or "Vancouver") and (*time* = "Q1" or "Q2") and (item = "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. Figure 4.12 shows a pivot operation where the *item* and *location* axes in a 2-D slice are rotated.

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.

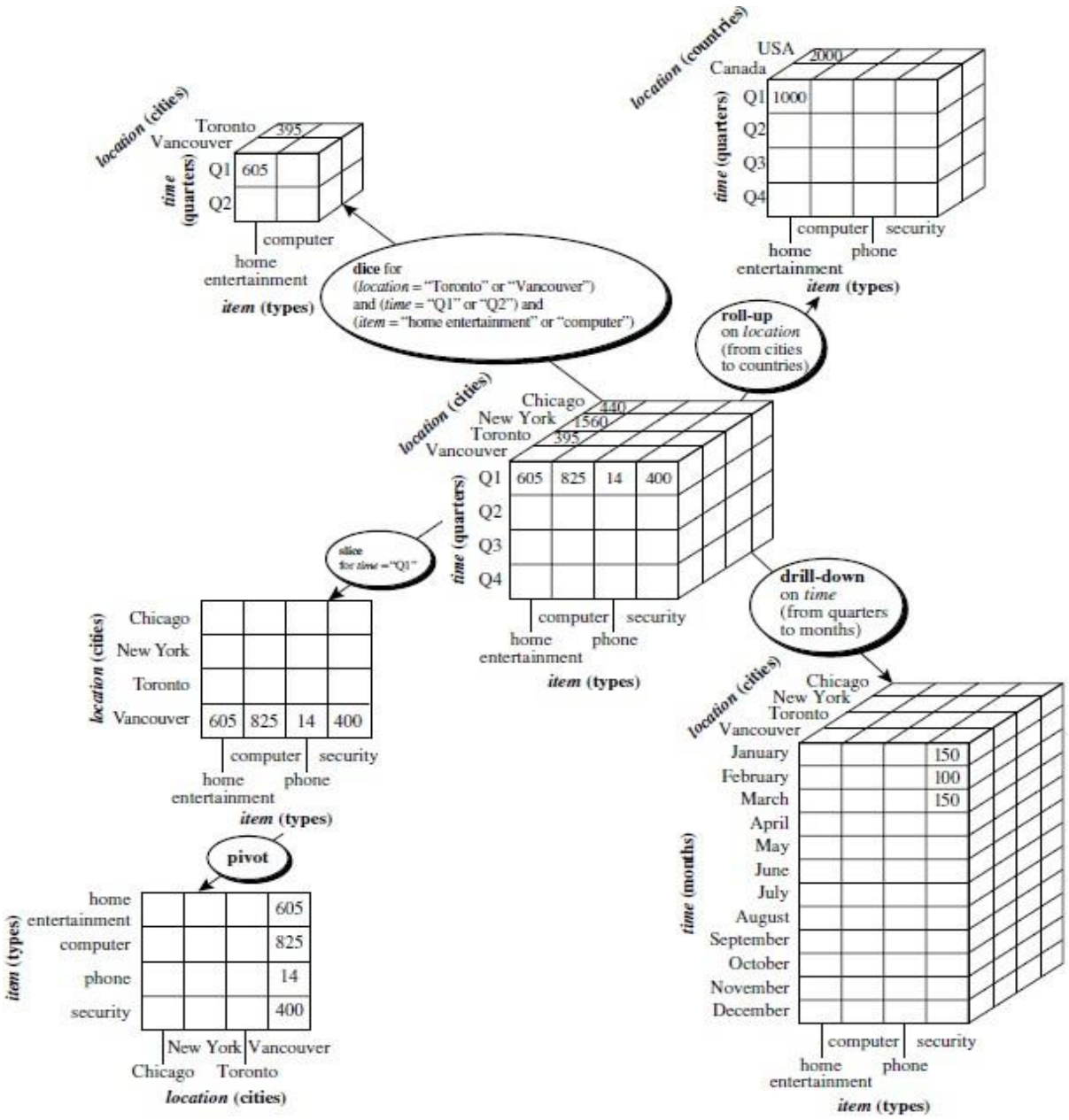

Figure 4.12 Examples of typical OLAP operations on multidimensional data.

OLAP offers analytical modeling capabilities, including a calculation engine for deriving ratios, variance, and so on, and for computing measures across multiple dimensions. OLAP also supports functional models for
forecasting, trend analysis, and statistical analysis.

Reference: Data Mining – Concepts and Techniques – 3rd Edition, Jiawei Han, Micheline Kamber & Jian Pei-Elsevier

OLAP Systems versus Statistical Databases

A **statistical database** is a database system that is designed to support statistical applications.

OLAP and SDB systems, however, have distinguishing differences. While SDBs tend to focus on socioeconomic applications, OLAP has been targeted for business applications. Privacy issues regarding concept hierarchies are a major concern for SDBs. For example, given summarized socioeconomic data, it is controversial to allow users to view the corresponding low-level data. Finally, unlike SDBs, OLAP systems are designed for efficiently handling huge amounts of data.

A Starnet Query Model for Querying Multidimensional Databases

The querying of multidimensional databases can be based on a **starnet model**, which consists of radial lines emanating from a central point, where each line represents a concept hierarchy for a dimension. Each abstraction level in the hierarchy is called a **footprint**. These represent the granularities available for use by OLAP operations such as drill-down and roll-up.

Example 4.5 Starnet This starnet consists of four radial lines, representing concept hierarchies for the dimensions *location*, *customer*, *item*, and *time*, respectively. Each line consists of footprints representing abstraction levels of the dimension. For example, the *time* line has four footprints: "day," "month," "quarter," and "year."

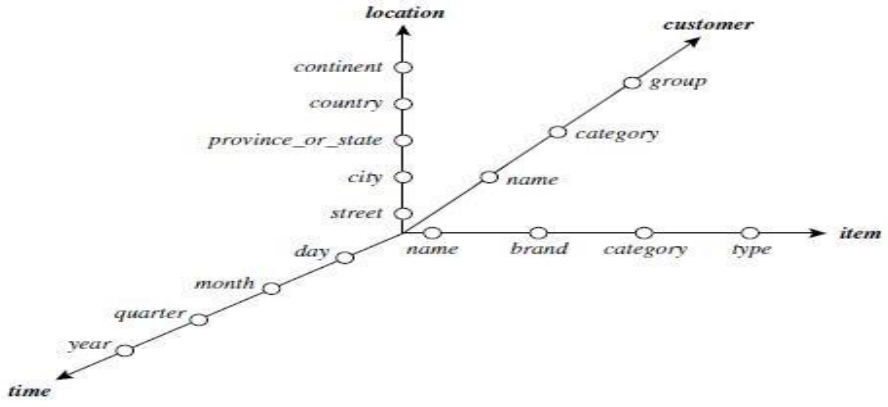

Figure 4.13 A starnet model of business queries.

Concept hierarchies can be used to **generalize** data by replacing low-level values (such as "day" for the *time* dimension) by higher-level abstractions (such as "year"), or to **specialize** data by replacing higher-level abstractions with lower-level values.

Data Objects and Attribute Types

Data sets are made up of data objects. A **data object** represents an entity—in a sales database, the objects may be customers, store items, and sales; in a medical database, the objects may be patients;

Data objects are typically described by attributes. Data objects can also be referred to as *samples, examples, instances, data points*, or *objects*. If the data objects are stored in a database, they are *data tuples*. That is, the rows of a database correspond to the data objects, and the columns correspond to the attributes.

What Is an Attribute?

An **attribute** is a data field, representing a characteristic or feature of a data object. The nouns *attribute, dimension, feature*, and *variable* are often used interchangeably in the literature. The term*dimension* is commonly used in data warehousing.Machine learning literature tends to use the term *feature*, while statisticians prefer the term *variable*.

The **type** of an attribute is determined by the set of possible values—nominal, binary, ordinal, or numeric—the attribute can have.

Attributes are of two types. They are
1. Quantitative(Numeric attributes) – they describe quantity.
2. Qualitative(Nominal, binary, Ordinal attributes) - they *describe* a feature of an object without giving an actual size or quantity.

Nominal Attributes
Nominal means "relating to names." The values of a **nominal attribute** are symbols or *names of things*. Each value represents some kind of category, code, or state, and so nominal attributes are also referred to as **categorical**. The values do not have any meaningful order.
Example 2.1 Nominal attributes. Suppose that *hair color* and *marital status* are two attributes describing *person* objects. In our application, possible values for *hair color* are *black*, *brown*, *blond*, *red*, *auburn*, *gray*, and *white*. The attribute *marital status* can take on the values *single*, *married*, *divorced*, and *widowed*. Both *hair color* and *marital status* are nominal attributes. Another example of a nominal attribute is *occupation*, with the values *teacher*, *dentist*, *programmer*, *farmer*, and so on.

Binary Attributes
A **binary attribute** is a nominal attribute with only two categories or states: 0 or 1, where 0 typically means that the attribute is absent, and 1 means that it is present. Binary attributes are referred to as **Boolean** if the two states correspond to *true* and *false*.
Example 2.2 Binary attributes. Given the attribute *smoker* describing a *patient* object, 1 indicates that the patient smokes, while 0 indicates that the patient does not. Similarly, suppose the patient undergoes a medical test that has two possible outcomes. The attribute *medical test* is binary, where a value of 1 means the result of the test for the patient is positive, while 0 means the result is negative.

A binary attribute is **symmetric** if both of its states are equally valuable and carry the same weight; that is, there is no preference on which outcome should be coded as 0 or 1. One such example could be the attribute *gender* having the states *male* and *female*.
A binary attribute is **asymmetric** if the outcomes of the states are not equally important, such as the *positive* and *negative* outcomes of a medical test for HIV. By convention, we code the most important outcome, which is usually the rarest one, by 1 (e.g., *HIV positive*) and the other by 0 (e.g., *HIV negative*).

Ordinal Attributes

An **ordinal attribute** is an attribute with possible values that have a meaningful order or *ranking* among them, but the magnitude between successive values is not known.
For example, Drink size- *small, medium*, and *large*.
Grade- S, *A+*, *A*, *B+*, B, C, D, F
professional rank- assistant, associate, and *full* for professors.
army ranks- *private, private first class, specialist, corporal, and sergeant.*
Customer satisfaction- 0*: very dissatisfied,1: somewhat dissatisfied, 2: neutral, 3: satisfied,* and *4: very satisfied.*
Ordinal attributes may also be obtained fromthe discretization of numeric quantities by splitting the value range into a finite number of ordered categories. The central tendency of an ordinal attribute can be represented by its mode and its median (the middle value in an ordered sequence), but the mean cannot be defined.

Numeric Attributes

A **numeric attribute** is *quantitative*; that is, it is a measurable quantity, represented in integer or real values. Numeric attributes can be *interval-scaled* or *ratio-scaled*.

Interval-scaled attributes are measured on a scale of equal-size units. The values of interval-scaled attributes have order and can be positive, 0, or negative.
Example Interval-scaled attributes. A *temperature* attribute is interval-scaled. Suppose that we have the outdoor *temperature* value for a number of different days, where each day is an object. By ordering the values, we obtain a ranking of the objects with respect to *temperature*. Temperatures in Celsius and Fahrenheit do not have a true zero-point, that is, neither 0^0C nor 0^0F indicates "no temperature."
Because interval-scaled attributes are numeric, we can compute their mean value, in addition to the median and mode measures of central tendency.
A **ratio-scaled attribute** is a numeric attribute with an inherent zero-point. That is, if a measurement is ratio-scaled, we can speak of a value as being a multiple (or ratio) of another value.
For example, Unlike temperatures in Celsius and Fahrenheit, the Kelvin (K) temperature scale has what is considered a true zero-point (0^0K = -273.15^{0}C): It is the point at which the particles that comprise matter have zero kinetic energy. Other examples of ratio-scaled attributes include *count* attributes such as *years of experience* (e.g., the objects are employees) and *number of words* (e.g., the objects are documents).

Discrete versus Continuous Attributes

A **discrete attribute** has a finite or countably infinite set of values, which may or may not be represented as integers. The attributes *hair_color*, *smoker*, *medical_test*, and *drink_size* each have a finite number of values, and so are discrete. Note that discrete attributes may have numeric values, such as 0 and 1 for binary attributes or, the values 0 to 110 for the attribute *age*.
If an attribute is not discrete, it is **continuous**.

Basic Statistical Descriptions of Data

For data preprocessing to be successful, it is essential to have an overall picture of your data. Basic statistical descriptions can be used to identify properties of the data and highlight which data values should be treated as noise or outliers.

Measuring the Central Tendency: Mean, Median, and Mode

Suppose that we have some attribute *X*, like *salary*, which has been recorded for a set of objects.
Let *x*1,*x*2, ...,*xN* be the set of *N* observed values or *observations* for *X*. Here, these values may also be referred to as the data set (for *X*). If we were to plot the observations for *salary*, where would most of the values fall? This gives us an idea of the central tendency of the data. Measures of central tendency include the mean, median, mode, and midrange.
The most common and effective numeric measure of the "center" of a set of data is the *(arithmetic) mean*. Let *x*1,*x*2, .. ,*xN* be a set of *N* values or *observations*, such as for some numeric attribute *X*, like *salary*. The **mean** of this set of values is

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N} = \frac{x_1 + x_2 + \cdots + x_N}{N}.$$

Example 2.6 Mean. Suppose we have the following values for *salary* (in thousands of dollars), shown in increasing order: 30, 36, 47, 50, 52, 52, 56, 60, 63, 70, 70, 110.using the above equation, we get

$$\bar{x} = \frac{30+36+47+50+52+52+56+60+63+70+70+110}{12}$$
$$= \frac{696}{12} = 58.$$

Thus, the mean salary is $58,000.
the **weighted arithmetic mean** or the **weighted average**.

$$\bar{x} = \frac{\sum_{i=1}^{N} w_i x_i}{\sum_{i=1}^{N} w_i} = \frac{w_1 x_1 + w_2 x_2 + \cdots + w_N x_N}{w_1 + w_2 + \cdots + w_N}.$$

A major problem with the mean is its sensitivity to extreme (e.g., outlier) values. Even a small number of extreme values can corrupt the mean. For example, the mean salary at a company may be substantially pushed up by that of a few highly paid managers. Similarly, the mean score of a class in an exam could be pulled down quite a bit by a few very low scores. To offset the effect caused by a small number of extreme values, we can instead use the **trimmed mean**, which is the mean obtained after chopping off values at the high and low extremes. For example, we can sort the values observed for *salary* and remove the top and bottom 2% before computing the mean. We should avoid trimming too large a portion (such as 20%) at both ends, as this can result in the loss of valuable information.

For skewed (asymmetric) data, a better measure of the center of data is the **median**, which is the middle value in a set of ordered data values. If *N* is odd, then the median is the *middle value* of the ordered set. If *N* is even, then the median is not unique; it is the two middlemost values and any value in between.
In the above given example, that is, (52+56)/2=108/2=54. Thus, the median is $54,000.

Suppose that we had only the first 11 values in the list. Given an odd number of values, the median is the middlemost value. This is the sixth value in this list, which has a value of $52,000.

The median is expensive to compute when we have a large number of observations. For numeric attributes, however, we can easily *approximate* the value. Assume that data are grouped in intervals according to their *xi* data values and that the frequency (i.e., number of data values) of each interval is known. For example, employees may be grouped according to their annual salary in intervals such as $10–20,000, $20–30,000, and so on. We can approximate the median of the entire data set (e.g., the median salary) by interpolation using the formula

$$median = L_1 + \left(\frac{N/2 - (\sum freq)_l}{freq_{median}} \right) width,$$

where *L*1 is the lower boundary of the median interval, *N* is the number of values in the entire data set, $(\sum freq)_l$ is the sum of the frequencies of all of the intervals that are lower than the median interval, *freqmedian* is the frequency of the median interval, and *width* is the width of the median interval.

The **mode** for a set of data is the value that occurs most frequently in the set. Data sets with one, two,

or three modes are respectively called **unimodal**, **bimodal**, and **trimodal**. In general, a data set with two or more modes is**multimodal**. At the other extreme, if each data value occurs only once, then there is no mode.

In the above example, the data given is bimodal. The two modes are $52,000 and $70,000.
For unimodal numeric data that are moderately skewed (asymmetrical), we have the following empirical relation:

$$mean - mode \approx 3 \times (mean - median).$$

This implies that the mode for unimodal frequency curves that are moderately skewed can easily be approximated if the mean and median values are known.
The **midrange** can also be used to assess the central tendency of a numeric data set.
In the above example, the midrange is (30,000+1,10,000)/2=$70,000.

In a unimodal frequency curve with perfect **symmetric** data distribution, the mean, median, and mode are all at the same center value, as shown in Figure 2.1(a) Data in most real applications are not symmetric. They may instead be either **positively skewed**, where the mode occurs at a value that is smaller than the median (Figure 2.1b), or **negatively skewed**, where the mode occurs at a value greater than the median (Figure 2.1c).

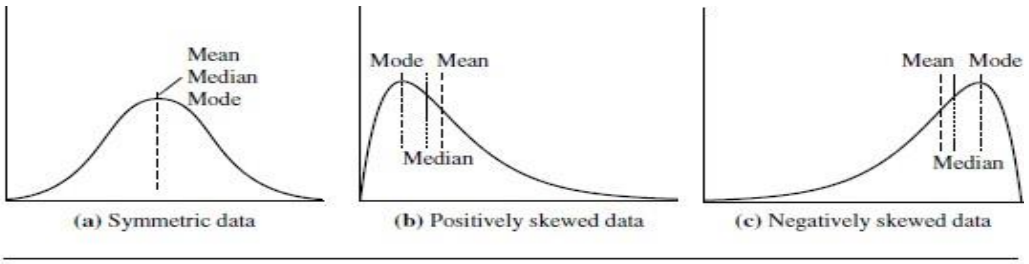

Figure 2.1 Mean, median, and mode of symmetric versus positively and negatively skewed data.

Measuring the Dispersion of Data: Range, Quartiles, Variance, Standard Deviation, and Interquartile Range

The measures include range, quantiles, quartiles, percentiles, and the interquartile range. The five-number summary, which can be displayed as a boxplot, is useful in identifying outliers. Variance and standard deviation also indicate the spread of a data distribution.
Let $x1, x2, \ldots, xN$ be a set of observations for some numeric attribute, X. The **range** of the set is the difference between the largest (max()) and smallest (min()) values.
Suppose that the data for attribute X are sorted in increasing numeric order. Imagine that we can pick certain data points so as to split the data distribution into equal-size consecutive sets, as in Figure 2.2. These data points are called *quantiles*. **Quantiles** are points taken at regular intervals of a data distribution, dividing it into essentially equalsize consecutive sets.
The 2-quantile is the data point dividing the lower and upper halves of the data distribution. It corresponds to the median. The 4-quantiles are the three data points that split the data distribution into four equal parts; each part represents one-fourth of the data distribution. They are more commonly referred to as **quartiles**. The 100-quantiles are more commonly referred to as **percentiles**; they divide the data distribution into 100 equal-sized consecutive sets.

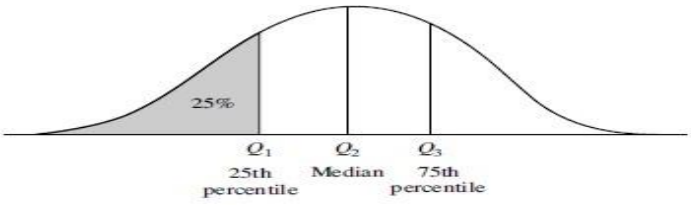

Figure 2.2 A plot of the data distribution for some attribute X. The quantiles plotted are quartiles. The three quartiles divide the distribution into four equal-size consecutive subsets. The second quartile corresponds to the median.

The quartiles give an indication of a distribution's center, spread, and shape. The **first quartile**, denoted by *Q*1, is the 25th percentile. It cuts off the lowest 25% of the data. The **third quartile**, denoted by *Q*3, is the 75th percentile—it cuts off the lowest 75% (or highest 25%) of the data. The second quartile is the 50th percentile. As the median, it gives the center of the data distribution.
The distance between the first and third quartiles is a simple measure of spread that gives the range covered by the middle half of the data. This distance is called the **interquartile range** (**IQR**) and is defined as

$$IQR = Q3 - Q1$$

Thus, the quartiles for this data are the third, sixth, and ninth values, respectively, in the sorted list. Therefore, *Q*1 = $47,000 and *Q*3 is $63,000. Thus, the interquartile range is *IQR* = 63-47 = $16,000.

Five-Number Summary, Boxplots, and Outliers

In the symmetric distribution, the median (and other measures of central tendency) splits the data into equal-size halves. This does not occur for skewed distributions. Therefore, it is more informative to also provide the two quartiles *Q*1 and *Q*3, along with the median. A common rule of thumb for identifying suspected **outliers** is to
single out values falling at least 1.5 * *IQR* above the third quartile or below the first quartile.

Because *Q*1, the median, and *Q*3 together contain no information about the endpoints (e.g., tails) of the data, a fuller summary of the shape of a distribution can be obtained by providing the lowest and highest data values as well. This is known as the *five-number summary*. The **five-number summary** of a distribution consists of the
median (*Q*2), the quartiles *Q*1 and *Q*3, and the smallest and largest individual observations, written in the order of *Minimum*, *Q*1, *Median*, *Q*3, *Maximum.*
Boxplots are a popular way of visualizing a distribution. A boxplot incorporates the five-number summary as follows:

- Typically, the ends of the box are at the quartiles so that the box length is the interquartile range.
- The median is marked by a line within the box.
- Two lines (called *whiskers*) outside the box extend to the smallest (*Minimum*) and largest (*Maximum*) observations.

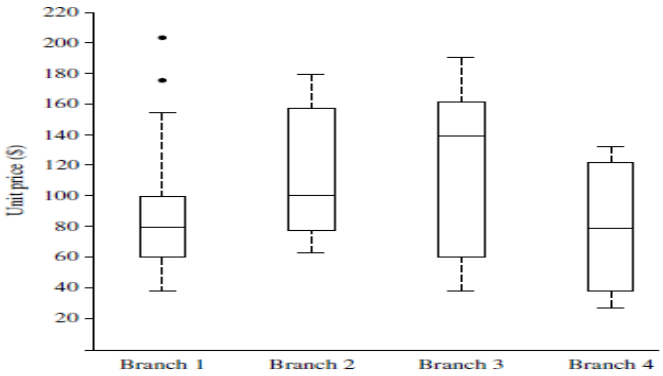

Figure 2.3 Boxplot for the unit price data for items sold at four branches of *AllElectronics* during a given time period.

Boxplots can be used in the comparisons of several sets of compatible data.

Example 2.11 Boxplot. Figure 2.3 shows boxplots for unit price data for items sold at four branches of *AllElectronics* during a given time period. For branch 1, we see that the median price of items sold is \$80, *Q*1 is \$60, and *Q*3 is \$100. Notice that two outlying observations for this branch were plotted individually, as their values of 175 and 202 are more than 1.5 times the IQR here of 40.

Variance and Standard Deviation

Variance and standard deviation are measures of data dispersion. They indicate how spread out a data distribution is. A low standard deviation means that the data observations tend to be very close to the mean, while a high standard deviation indicates that the data are spread out over a large range of values.

The **variance** of *N* observations, *x*1,*x*2, ... ,*xN*, for a numeric attribute *X* is

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 = \left(\frac{1}{N}\sum_{i=1}^{N}x_i^2\right) - \bar{x}^2,$$

where $\bar{x}$ is the mean value of the observations. The **standard deviation**, σ, of the observations is the square root of the variance, σ^2.

Example 2.12 Variance and standard deviation. In Example 2.6, we found $\bar{x} = \$58{,}000$ using Eq. (2.1) for the mean. To determine the variance and standard deviation of the data from that example, we set $N = 12$ and use Eq. (2.6) to obtain

$$\sigma^2 = \frac{1}{12}(30^2 + 36^2 + 47^2 \ldots + 110^2) - 58^2$$
$$\approx 379.17$$
$$\sigma \approx \sqrt{379.17} \approx 19.47.$$

■

Graphic Displays of Basic Statistical Descriptions of Data

These include *quantile plots*, *quantile–quantile plots*, *histograms*, and *scatter plots*. Such graphs are helpful for the visual inspection of data, which is useful for data preprocessing. The first three of these show univariate distributions (i.e., data for one attribute), while scatter plots show bivariate distributions (i.e., involving two attributes).

Quantile Plot

A **quantile plot** is a simple and effective way to have a first look at a univariate data distribution. First, it displays all of the data for the given attribute (allowing the user to assess both the overall behavior and unusual occurrences). Second, it plots quantile information . Let *xi* , for *i* = 1 to *N*, be the data sorted in increasing order so that *x*1 is the smallest observation and *xN* is the largest for some ordinal or numeric attribute *X*. Each observation, *xi* , is paired with a percentage, *fi* , which indicates that approximately *fi* * 100% of the data are below the value, *xi* . We say "approximately" because there may not be a value with exactly a fraction, *fi* , of the data below *xi* . Note that the 0.25 percentile corresponds to quartile *Q*1, the 0.50 percentile is the median,
and the 0.75 percentile is *Q*3.
Let

$$f_i = \frac{i - 0.5}{N}.$$

On a quantile plot, *xi* is graphed against *fi* . This allows us to compare different distributions based on their quantiles. For example, given the quantile plots of sales data for two different time periods, we can compare
their *Q*1, median, *Q*3, and other *fi* values at a glance.

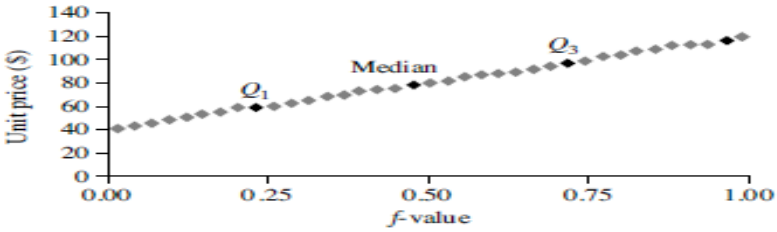

Figure 2.4 A quantile plot for the unit price data of Table 2.1.

Quantile–Quantile Plot

A **quantile–quantile plot**, or **q-q plot**, graphs the quantiles of one univariate distribution against the corresponding quantiles of another.
Suppose that we have two sets of observations for the attribute or variable *unit price*, taken from two different branch locations. Let *x*1, ...,*xN* be the data from the first branch, and *y*1, ... , *yM* be the data from the second, where each data set is sorted in increasing order. If *M* = *N* (i.e., the number of points in each set is the same), then we simply plot *yi* against *xi* , where *yi* and *xi* are both (*i* -0.5)/*N* quantiles of their respective data sets. If *M* < *N* (i.e., the second branch has fewer observations than the first), there can be only *M* points on the q-q plot. Here, *yi* is the (*i* -0.5)/*M* quantile of the *y* data, which is plotted against the (*i* - 0.5)/*M* quantile of the *x* data. This computation typically involves interpolation.

Table 2.1 A Set of Unit Price Data for Items Sold at a Branch of *AllElectronics*

Unit price ($)	*Count of items sold*
40	275
43	300
47	250
—	—
74	360
75	515
78	540
—	—
115	320
117	270
120	350

Example 2.14 Quantile–quantile plot. Figure 2.5 shows a quantile–quantile plot for *unit price* data of items sold at two branches of *AllElectronics* during a given time period. Each point corresponds to the same quantile for each data set and shows the unit price of items sold at branch 1 versus branch 2 for that quantile.(The darker points correspond to the data for $Q1$, the median, and $Q3$, respectively.) for example, that at $Q1$, the unit price of items sold at branch 1 was slightly less than that at branch 2. In other words, 25% of items sold at branch 1 were less than or equal to $60, while 25% of items sold at branch 2 were less than or equal to $64. At the 50th percentile (marked by the median, which is also $Q2$), we see that 50% of items sold at branch 1 were less than $78, while 50% of items at branch 2 were less than $85. In general, we note that there is a shift in the distribution of branch 1 with respect to branch 2 in that the unit prices of items sold at branch 1 tend to be lower than those at branch 2.

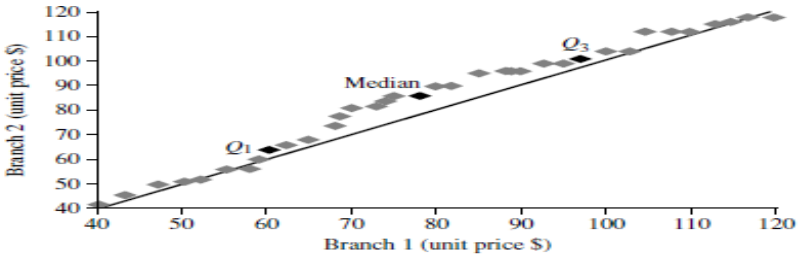

Figure 2.5 A q-q plot for unit price data from two *AllElectronics* branches.

Histograms

"Histos" means pole or mast, and "gram" means chart, so a histogram is a chart of poles. Plotting histograms is a graphical method for summarizing the distribution of a given attribute, X. If X is nominal, such as *automobile_ model* or *item_type*, then a pole or vertical bar is drawn for each known value of X. The height of the bar indicates the frequency (i.e., count) of that X value. The resulting graph is more commonly known as a **bar chart**.

If X is numeric, the term *histogram* is preferred. The range of values for X is partitioned into disjoint consecutive subranges. The subranges, referred to as *buckets* or *bins*, are disjoint subsets of the data distribution for X. The range of a bucket is known as the **width**. Typically, the buckets are of equal width. For example, a *price* attribute with a value range of $1 to $200 (rounded up to the nearest dollar) can be partitioned into subranges 1 to 20, 21 to 40, 41 to 60, and so on.

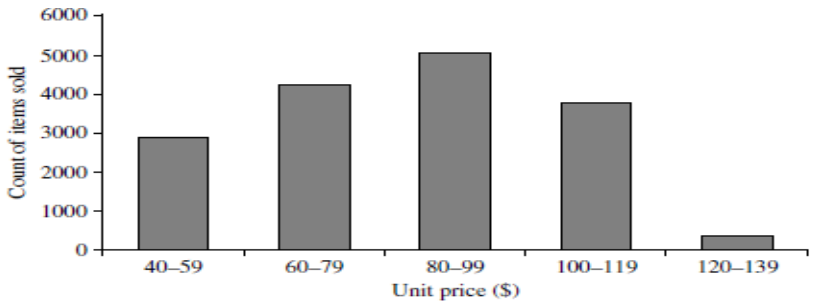

Figure 2.6 A histogram for the Table 2.1 data set.

Although histograms are widely used, they may not be as effective as the quantile plot, q-q plot, and boxplot methods in comparing groups of univariate observations.

Scatter Plots and Data Correlation

A **scatter plot** is one of the most effective graphical methods for determining if there appears to be a relationship, pattern, or trend between two numeric attributes.

The scatter plot is a useful method for providing a first look at bivariate data to see clusters of points and outliers, or to explore the possibility of correlation relationships. Two attributes, *X*, and *Y*, are **correlated** if one attribute implies the other. Correlations can be positive, negative, or null (uncorrelated). If the plotted points pattern slopes from lower left to upper right, this means that the values of *X* increase as the values of *Y* increase, suggesting a *positive correlation* (Figure 2.8a). If the pattern of plotted points slopes from upper left to lower right, the values of *X* increase as the values of *Y* decrease, suggesting a *negative correlation* (Figure 2.8b).

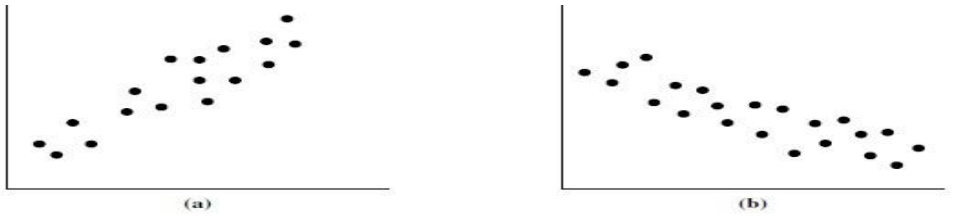

Figure 2.8 Scatter plots can be used to find (a) positive or (b) negative correlations between attributes.

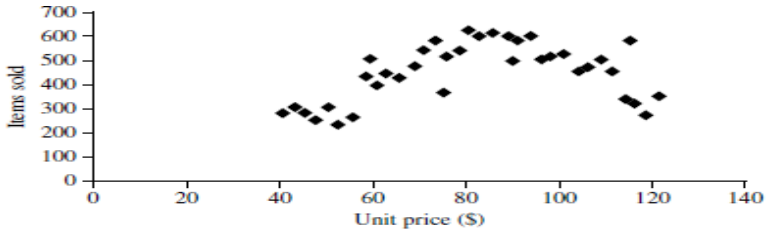

Figure 2.7 A scatter plot for the Table 2.1 data set.

Figure 2.9 shows three cases for which there is no correlation relationship between the two attributes in each of the given data sets.

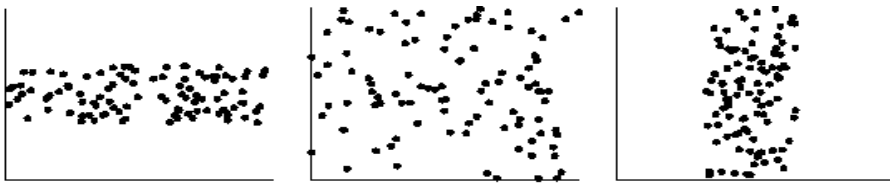

Figure 2.9 Three cases where there is no observed correlation between the two plotted attributes in each of the data sets.

Measuring Data Similarity and Dissimilarity

A similarity measure for two objects, *i* and *j*, will typically return the value 0 if the objects are unalike. The higher the similarity value, the greater the similarity between objects. (Typically, a value of 1 indicates complete similarity, that is, the objects are identical.) A dissimilarity measure, returns a value of 0 if the objects are the same (and therefore, far from being dissimilar). The higher the dissimilarity value, the more dissimilar the two objects are.

Data Matrix versus Dissimilarity Matrix

Suppose that we have *n* objects (e.g., persons, items, or courses) described by *p* attributes (also called *measurements* or *features*, such as age, height, weight, or gender).The objects are $x1 = (x11,x12,.. ,x1p)$, $x2= (x21,x22, ... ,x2p)$, and so on, where xij is the value for object xi of the jth attribute.

Data matrix (or *object-by-attribute structure*): This structure stores the *n* data objects in the form of a relational table, or *n*-by-*p* matrix (*n* objects * *p* attributes):

$$\begin{bmatrix} x_{11} & \cdots & x_{1f} & \cdots & x_{1p} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{i1} & \cdots & x_{if} & \cdots & x_{ip} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{n1} & \cdots & x_{nf} & \cdots & x_{np} \end{bmatrix}.$$

Each row corresponds to an object. we may use *f* to index through the *p* attributes.

Dissimilarity matrix (or *object-by-object structure*): This structure stores a collection of proximities that are available for all pairs of *n* objects. It is often represented by an *n*-by-*n* table:

$$\begin{bmatrix} 0 & & & & \\ d(2,1) & 0 & & & \\ d(3,1) & d(3,2) & 0 & & \\ \vdots & \vdots & \vdots & & \\ d(n,1) & d(n,2) & \cdots & \cdots & 0 \end{bmatrix},$$

where d(*i, j)* is the measured **dissimilarity** or "difference" between objects *i* and *j*. In general, *d(i, j)* is a non-negative number that is close to 0 when objects *i* and *j* are highly similar or "near" each other, and becomes larger the more they differ.Note that *d*(*i, i)* = 0; that is, the difference between an object and itself is 0.

Measures of similarity can often be expressed as a function of measures of dissimilarity. For example, for nominal data,

$$sim(i, j) = 1 - d(i, j),$$

where *sim*(*i, j)* is the similarity between objects *i* and *j*.

A data matrix is made up of two entities or "things," namely rows (for objects) and columns (for attributes). Therefore, the data matrix is often called a **two-mode** matrix. The dissimilarity matrix contains one kind of entity (dissimilarities) and so is called a **one-mode** matrix.

Proximity Measures for Nominal Attributes

A nominal attribute can take on two or more states. For example, *map color* is a nominal attribute that may have, say, five states: *red, yellow, green, pink*, and *blue*.

Let the number of states of a nominal attribute be *M*. The states can be denoted by letters, symbols, or a set of integers, such as 1, 2, ..., *M*. Notice that such integers are used just for data handling and do not represent any specific ordering.

"How is dissimilarity computed between objects described by nominal attributes?" The dissimilarity between two objects *i* and *j* can be computed based on the ratio of mismatches:

$$d(i, j) = \frac{p - m}{p},$$

where *m* is the number of *matches* (i.e., the number of attributes for which *i* and *j* are in the same state), and *p* is the total number of attributes describing the objects.

Table 2.2 A Sample Data Table Containing Attributes of Mixed Type

Object Identifier	*test-1 (nominal)*	*test-2 (ordinal)*	*test-3 (numeric)*
1	code A	excellent	45
2	code B	fair	22
3	code C	good	64
4	code A	excellent	28

Example 2.17 Dissimilarity between nominal attributes. Suppose that we have the sample data of Table 2.2, except that only the *object-identifier* and the attribute *test-1* are available, where *test-1* is nominal. Let's compute the dissimilarity matrix , that is,

$$\begin{bmatrix} 0 & & & \\ d(2,1) & 0 & & \\ d(3,1) & d(3,2) & 0 & \\ d(4,1) & d(4,2) & d(4,3) & 0 \end{bmatrix}.$$

Since here we have one nominal attribute, *test-1*, we set $p = 1$ in above equation, so that *d(i, j)* evaluates to 0 if objects *i* and *j* match, and 1 if the objects differ. Thus, we get

$$\begin{bmatrix} 0 & & & \\ 1 & 0 & & \\ 1 & 1 & 0 & \\ 0 & 1 & 1 & 0 \end{bmatrix}.$$

From this, we see that all objects are dissimilar except objects 1 and 4 (i.e., $d(4,1) = 0$).
Alternatively, similarity can be computed as

$$sim(i, j) = 1 - d(i, j) = \frac{m}{p}.$$

Proximity Measures for Binary Attributes

a binary attribute has only one of two states: 0 and 1, where 0 means that the attribute is absent, and 1 means that it is present. Given the attribute *smoker* describing a patient, for instance, 1 indicates that the patient smokes, while 0 indicates that the patient does not.
"So, how can we compute the dissimilarity between two binary attributes?"
symmetric binary dissimilarity - If objects *i* and *j* are described by symmetric binary attributes, then the dissimilarity between *i* and *j* is

$$d(i, j) = \frac{r + s}{q + r + s + t}.$$

For asymmetric binary attributes, the two states are not equally important, such as the *positive* (1) and *negative* (0) outcomes of a disease test. Given two asymmetric binary attributes, the agreement of two 1s (a positive match) is then considered more significant than that of two 0s (a negative match). Therefore, such binary attributes are often considered "monary" (having one state). The dissimilarity based on these attributes is called **asymmetric binary dissimilarity**, where the number of negative matches, *t* , is considered unimportant and is thus ignored in the following computation:

$$d(i, j) = \frac{r + s}{q + r + s}.$$

Table 2.3 Contingency Table for Binary Attributes

		Object *j*		
		1	0	sum
	1	q	r	$q+r$
Object *i*	0	s	t	$s+t$
	sum	$q+s$	$r+t$	p

Complementarily, we can measure the difference between two binary attributes based on the notion of similarity instead of dissimilarity. For example, the **asymmetric binary similarity** between the objects *i* and *j* can be computed as

$$sim(i, j) = \frac{q}{q + r + s} = 1 - d(i, j).$$

The coefficient *sim*(*i*, j) of above equation is called the **Jaccard coefficient.**
Example 2.18 Dissimilarity between binary attributes. Suppose that a patient record table (Table 2.4) contains the attributes *name, gender, fever, cough, test-1, test-2, test-3,* and *test-4,* where *name* is an object identifier, *gender* is a symmetric attribute, and the remaining attributes are asymmetric binary.

Table 2.4 Relational Table Where Patients Are Described by Binary Attributes

name	gender	fever	cough	test-1	test-2	test-3	test-4
Jack	M	Y	N	P	N	N	N
Jim	M	Y	Y	N	N	N	N
Mary	F	Y	N	P	N	P	N
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

For asymmetric attribute values, let the values *Y* (*yes*) and *P* (*positive*) be set to 1, and the value *N* (*no* or *negative*) be set to 0. Suppose that the distance between objects (patients) is computed based only on the asymmetric attributes. According to **asymmetric binary dissimilarity** Eq. , the distance between each pair of the three patients—Jack, Mary, and Jim—is

$$d(Jack, Jim) = \frac{1+1}{1+1+1} = 0.67,$$
$$d(Jack, Mary) = \frac{0+1}{2+0+1} = 0.33,$$
$$d(Jim, Mary) = \frac{1+2}{1+1+2} = 0.75.$$

These measurements suggest that Jim and Mary are unlikely to have a similar disease because they have the highest dissimilarity value among the three pairs. Of the three patients, Jack and Mary are the most likely to have a similar disease.

Dissimilarity of Numeric Data: Minkowski Distance

Minkowski distance is a generalization of the Euclidean and Manhattan distances. It is defined as

$$d(i, j) = \sqrt[h]{|x_{i1} - x_{j1}|^h + |x_{i2} - x_{j2}|^h + \cdots + |x_{ip} - x_{jp}|^h},$$

where *h* is a real number such that *h* >= 1. (Such a distance is also called *Lp* **norm** in some literature, where the symbol *p* refers to our notation of *h*.) It represents the Manhattan distance when *h* = 1 (i.e., *L*1 norm) and Euclidean distance when *h* = 2 (i.e., *L*2 norm).

The most popular distance measure is **Euclidean distance** (i.e., straight line or "as the crow flies"). Let *i* = (*xi*1, *xi*2, ... , *xip*) and *j* = (*xj*1, *xj*2, ... , *xjp*) be two objects described by *p* numeric attributes. The Euclidean distance between objects *i* and *j* is defined as

$$d(i, j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{ip} - x_{jp})^2}.$$

Another well-known measure is the **Manhattan (or city block) distance**, named so because it is the distance in blocks between any two points in a city (such as 2 blocks down and 3 blocks over for a total of 5 blocks). It is defined as

$$d(i, j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \cdots + |x_{ip} - x_{jp}|.$$

Both the Euclidean and the Manhattan distance satisfy the following mathematical properties:

Non-negativity: *d(i, j)*>= 0: Distance is a non-negative number.
Identity of indiscernibles: *d(i, i)*= 0: The distance of an object to itself is 0.
Symmetry: *d(i, j)*= *d(j, i)*: Distance is a symmetric function.
Triangle inequality: *d(i, j)*<= *d(i, k) + d(k, j)*: Going directly from object *i* to object *j* in space is no more than making a detour over any other object *k*.

A measure that satisfies these conditions is known as **metric**. Please note that the non-negativity property is implied by the other three properties.

The **supremum distance** (also referred to as ***Lmax***, ***L***1 **norm** and as the **Chebyshev distance**) is a generalization of the Minkowski distance for h --> ∞. To compute it, we find the attribute *f* that gives

the maximum difference in values between the two objects. This difference is the supremum distance, defined more formally as:

$$d(i,j) = \lim_{h\to\infty}\left(\sum_{f=1}^{p}|x_{if}-x_{jf}|^h\right)^{\frac{1}{h}} = \max_{f}^{p}|x_{if}-x_{jf}|.$$

The L^∞ norm is also known as the *uniform norm*.

Example 2.19 Euclidean distance and Manhattan distance. Let $\boldsymbol{x1}$ = (1, 2) and $\boldsymbol{x2}$ = (3, 5) represent two objects as shown in Figure 2.23. The Euclidean distance between the two is $\sqrt{2^2+3^2} = 3.61$. The Manhattan distance between the two is 2 + 3 = 5.

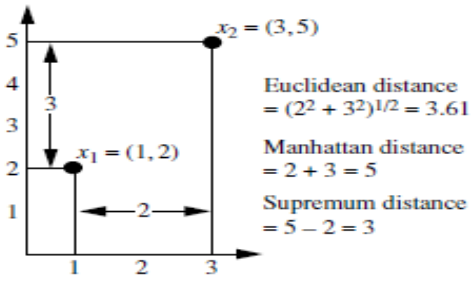

Figure 2.23 Euclidean, Manhattan, and supremum distances between two objects.

If each attribute is assigned a weight according to its perceived importance, the **weighted Euclidean distance** can be computed as

$$d(i,j) = \sqrt{w_1|x_{i1}-x_{j1}|^2 + w_2|x_{i2}-x_{j2}|^2 + \cdots + w_m|x_{ip}-x_{jp}|^2}.$$

Proximity Measures for Ordinal Attributes

The values of an ordinal attribute have a meaningful order or ranking about them, yet the magnitude between successive values is unknown. An example includes the sequence *small*, *medium*, *large* for a *size* attribute.

Let *M* represent the number of possible states that an ordinal attribute can have. These ordered states define the ranking 1, ... , *Mf* .

"How are ordinal attributes handled?" Suppose that *f* is an attribute from a set of ordinal attributes describing *n* objects. The dissimilarity computation with respect to *f* involves the following steps:

1. The value of *f* for the *i*th object is *xif* , and *f* has *Mf* ordered states, representing the ranking 1, ... , *Mf* . Replace each *xif* by its corresponding rank, *rif* $\in \{1, \ldots, Mf\}$.
2. Since each ordinal attribute can have a different number of states, it is often necessary to map the range of each attribute onto [0.0, 1.0] so that each attribute has equal weight. We perform such data normalization by replacing the rank *rif* of the *i*th object in the *f* th attribute by
   $$\mathbf{Z_{if} = (r_{if} - 1)/ (M_f\text{-}1)}$$
3. Dissimilarity can then be computed using any of the distance measures dissimilarity matrix for numeric attributes, using *zif* to represent the *f* value for the *i*th object.

Example 2.21 Dissimilarity between ordinal attributes. Suppose that we have the sample data shown earlier in Table 2.2, except that this time only the *object-identifier* and the continuous ordinal attribute, *test-2*, are available. There are three states for *test-2*: *fair*, *good*, and *excellent*, that is, *Mf* = 3. For step 1, if we replace each value for *test-2* by its rank, the four objects are assigned the ranks 3, 1, 2, and 3, respectively. Step 2 normalizes the ranking by mapping rank 1 to 0.0, rank 2 to 0.5, and rank

3 to 1.0. For step 3, we can use, say, the Euclidean distance Eq., which results in the following dissimilarity matrix:

$$\begin{bmatrix} 0 & & & \\ 1.0 & 0 & & \\ 0.5 & 0.5 & 0 & \\ 0 & 1.0 & 0.5 & 0 \end{bmatrix}$$

Therefore, objects 1 and 2 are the most dissimilar, as are objects 2 and 4 (i.e., $d(2,1) = 1.0$ and $d(4,2) = 1.0$). objects 1 and 4 are both *excellent*. Object 2 is *fair*, which is at the opposite end of the range of values for *test-2*.

Dissimilarity for Attributes of Mixed Types

"So, how can we compute the dissimilarity between objects of mixed attribute types?"
One approach is to group each type of attribute together, performing separate data mining (e.g., clustering) analysis for each type. This is feasible if these analyses derive compatible results.

A more preferable approach is to process all attribute types together, performing a single analysis. One such technique combines the different attributes into a single dissimilarity matrix, bringing all of the meaningful attributes onto a common scale of the interval [0.0, 1.0].
Suppose that the data set contains *p* attributes of mixed type. The dissimilarity *d(i, j)* between objects *i* and *j* is defined as

$$d(i,j) = \frac{\sum_{f=1}^{p} \delta_{ij}^{(f)} d_{ij}^{(f)}}{\sum_{f=1}^{p} \delta_{ij}^{(f)}},$$

where the indicator $\delta_{ij}^{(f)} = 0$ if either (1) *xif* or *xjf* is missing (i.e., there is no measurement of attribute *f* for object *i* or object *j*), or (2) *xif* = *xjf* = 0 and attribute *f* is asymmetric binary; otherwise, $\delta_{ij}^{(f)}$ = 1. The contribution of attribute *f* to the dissimilarity between *i* and *j* (i.e., $d_{ij}^{(f)}$) is computed dependent on its type:

- If f is numeric: $d_{ij}^{(f)} = \frac{|x_{if} - x_{jf}|}{max_h x_{hf} - min_h x_{hf}}$, where h runs over all nonmissing objects for attribute f.
- If f is nominal or binary: $d_{ij}^{(f)} = 0$ if $x_{if} = x_{jf}$; otherwise, $d_{ij}^{(f)} = 1$.
- If f is ordinal: compute the ranks r_{if} and $z_{if} = \frac{r_{if} - 1}{M_f - 1}$, and treat z_{if} as numeric.

These steps are identical to what we have already seen for each of the individual attribute types. The only difference is for numeric attributes, where we normalize so that the values map to the interval [0.0, 1.0]. Thus, the dissimilarity between objects can be computed even when the attributes describing the objects are of different types.
Example 2.22 Dissimilarity between attributes of mixed type. Let's compute a dissimilarity matrix for the objects in Table 2.2. Now we will consider *all* of the attributes, which are of different types. In Examples 2.17 and 2.21, we worked out the dissimilarity matrices for each of the individual attributes. The procedures we followed for *test-1* (which is nominal) and *test-2* (which is ordinal) are the same as outlined earlier for processing attributes of mixed types. Therefore, we can use the dissimilarity matrices obtained for *test-1* and *test-2* later when we compute Eq. (2.22). First, however, we need to compute the dissimilarity matrix for the third attribute, *test-3* (which is numeric). That is, we must

compute $d_{ij}^{(3)}$. Following the case for numeric attributes, we let *maxhxh* = 64 and *minhxh* = 22. The difference between the two is used in Eq. (2.22) to normalize the values of the dissimilarity matrix. The resulting dissimilarity matrix for *test-3* is

$$\begin{bmatrix} 0 & & & \\ 0.55 & 0 & & \\ 0.45 & 1.00 & 0 & \\ 0.40 & 0.14 & 0.86 & 0 \end{bmatrix}$$

We can now use the dissimilarity matrices for the three attributes in our computation of above d(i,j)Eq. . The indicator $\delta_{ij}^{(f)} = 1$. for each of the three attributes, f .We get, for example, $d(3,1) = \frac{1(1)+1(0.50)+1(0.45)}{3} = 0.65.$

The resulting dissimilarity matrix obtained for the data described by the three attributes of mixed types is:

$$\begin{bmatrix} 0 & & & \\ 0.85 & 0 & & \\ 0.65 & 0.83 & 0 & \\ 0.13 & 0.71 & 0.79 & 0 \end{bmatrix}$$

From Table 2.2, we can intuitively guess that objects 1 and 4 are the most similar, based on their values for *test*-1 and *test*-2. This is confirmed by the dissimilarity matrix, where $d(4, 1)$ is the lowest value for any pair of different objects.

Cosine Similarity

A document can be represented by thousands of attributes, each recording the frequency of a particular word (such as a keyword) or phrase in the document. Thus, each document is an object represented by what is called a *term-frequency vector*.

Term-frequency vectors are typically very long and **sparse** (i.e., they have many 0 values). Applications using such structures include information retrieval, text document clustering, biological taxonomy, and gene feature mapping.

Cosine similarity is a measure of similarity that can be used to compare documents or, say, give a ranking of documents with respect to a given vector of query words. Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be two vectors for comparison. Using the cosine measure as a similarity function, we have

$$sim(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{||\boldsymbol{x}||||\boldsymbol{y}||},$$

where $||\boldsymbol{x}||$ is the Euclidean norm of vector $\boldsymbol{x} = (x1, x2, \ldots, xp)$, defined as $\sqrt{x_1^2 + x_2^2 + \cdots + x_p^2}$. Conceptually, it is the length of the vector. Similarly, $||\boldsymbol{y}||$ is the Euclidean norm of vector $\boldsymbol{y}$. The measure computes the cosine of the angle between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. A cosine value of 0 means that the two vectors are at 90 degrees to each
other (orthogonal) and have no match. The closer the cosine value to 1, the smaller the angle and the greater the match between vectors.

Table 2.5 Document Vector or Term-Frequency Vector

Document	team	coach	hockey	baseball	soccer	penalty	score	win	loss	season
Document1	5	0	3	0	2	0	0	2	0	0
Document2	3	0	2	0	1	1	0	1	0	1
Document3	0	7	0	2	1	0	0	3	0	0
Document4	0	1	0	0	1	2	2	0	3	0

Example 2.23 Cosine similarity between two term-frequency vectors. Suppose that ***x*** and ***y*** are the first two term-frequency vectors in Table 2.5. That is, ***x*** = (5, 0, 3, 0, 2, 0, 0, 2, 0,0) and ***y=(***3, 0, 2, 0, 1, 1, 0, 1, 0,1). How similar are ***x*** and ***y***? Using Sim(x,y) Eq. to compute the cosine similarity between the two vectors, we get:

$$x^t \cdot y = 5 \times 3 + 0 \times 0 + 3 \times 2 + 0 \times 0 + 2 \times 1 + 0 \times 1 + 0 \times 0 + 2 \times 1 + 0 \times 0 + 0 \times 1 = 25$$

$$||x|| = \sqrt{5^2 + 0^2 + 3^2 + 0^2 + 2^2 + 0^2 + 0^2 + 2^2 + 0^2 + 0^2} = 6.48$$

$$||y|| = \sqrt{3^2 + 0^2 + 2^2 + 0^2 + 1^2 + 1^2 + 0^2 + 1^2 + 0^2 + 1^2} = 4.12$$

$$sim(x, y) = 0.94$$

Therefore, if we were using the cosine similarity measure to compare these documents, they would be considered quite similar.