DATA WAREHOUSE AND OLAP TECHNOLOGY

BY

DR. PREETHAM KUMAR

HOD

DEPT. OF INFORMATION & COMMUNICATION TECHNOLOGY

Ref : Jiawei Han & Micheline Kamber

- According toWilliam 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"

- The four keywords, *subject-oriented, integrated, time-variant,* and *nonvolatile*, distinguish data warehouses from other data repository systems, such as
- Relational database systems,
- transaction processing systems, and
- file systems

Subject-oriented

- A data warehouse is organized around major subjects, such as customer, supplier, product, and sales.
- Rather than concentrating on the day-to-day operations and transaction processing of an organization, a data warehouse focuses on the **modeling and analysis of data for decision makers**.
- Hence, data warehouses typically provide a simple and concise view around particular subject issues by excluding data that are not useful in the decision support process

Integrated

- A data warehouse is usually constructed by integrating multiple heterogeneous sources, such as relational databases, flat files, and on-line 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 a historical perspective (e.g., the past 5–10 years).
- Every key structure in the data warehouse contains, either implicitly or explicitly, an element of time

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*

- A data warehouse is also often viewed as an architecture, constructed by integrating data from multiple heterogeneous sources to support structured and/or ad hoc queries, analytical reporting, and decision making.

- The construction of a data warehouse requires data cleaning, data integration, and data consolidation.
- The utilization of a data warehouse often necessitates a collection of *decision support* technologies.
- This allows "knowledge workers" (e.g., managers, analysts, and executives) to use the warehouse to quickly and conveniently obtain an overview of the data, and to make sound decisions based on information in the warehouse.

- Some authors use the term “data warehousing” to refer only to the process of data warehouse *construction*, while the term “warehouse DBMS” is used to refer to the *management and utilization* of data warehouses.

"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 the customer relationships, making environmental corrections, and managing the cost of corporate assets.

Traditional database approach

- 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 for resources with processing at local sources. It is inefficient and potentially expensive for frequent queries, especially for queries requiring aggregations

Update-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.
- Unlike on-line transaction processing databases, data warehouses do not contain the most current information.

- However, a data warehouse brings high performance to the integrated heterogeneous database system because data are copied, preprocessed, integrated, annotated, summarized, and restructured into one semantic data store
- Furthermore, query processing in data warehouses does not interfere with the processing at local sources.
- Moreover, data warehouses can store and integrate historical information and support complex multidimensional queries.

Differences between Operational Database Systems and Data Warehouses

- The major task of on-line operational database systems is to perform on-line transaction and query processing.
- These systems are called on-line 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 the different users.
- These systems are known as on-line analytical processing (OLAP) systems

OLTP and OLAP

- 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 systemis *market-oriented* and is used for data analysis by knowledge workers, including managers, executives, and analyst

- Data contents:
  - An OLTP system manages current data that, typically, are too detailed to be easily used for decision making.
  - An OLAP system manages large amounts of historical data, provides facilities for summarization and aggregation, and stores and manages information at different levels of granularity.
  - These features make the data easier to use in informed decision making.

- 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 *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 historical 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.
  - OLAP systems also deal with information that originates from different organizations, integrating information from many data stores.
  - Because of their huge volume, OLAP data are stored on multiple storage media.

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

- Other features that distinguish between OLTP and OLAP systems include database size,

frequency of operations, and performance metrics. These are summarized in the following Table.

Table 3.1 Comparison between 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	historical; 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	100 MB to GB	100 GB to TB
Priority	high performance, high availability	high flexibility, end-user autonomy
Metric	transaction throughput	query throughput, response time

NOTE: Table is partially based on [CD97].

A Multidimensional Data Model

- *"What is a data cube?"* data cube allows data to be modeled and viewed in multiple dimensions.
- It is defined by dimensions and facts.
- 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*.

- These dimensions allow the store to keep track of things like monthly sales of items and the branches and locations at which the items were sold.
- 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*.
- Dimension tables can be specified by users or experts, or automatically generated and adjusted based on data distributions.

- A multidimensional data model is typically organized around a central theme, like *sales*, for instance.
- This theme is represented by a fact table.
- Facts are numerical measures.
- Think of them as the quantities by which we want to analyze relationships between dimensions.

- 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

- 2-D data cube that is, in fact, a table or spreadsheet for sales data from *AllElectronics* is shown in the following Table .
- 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)

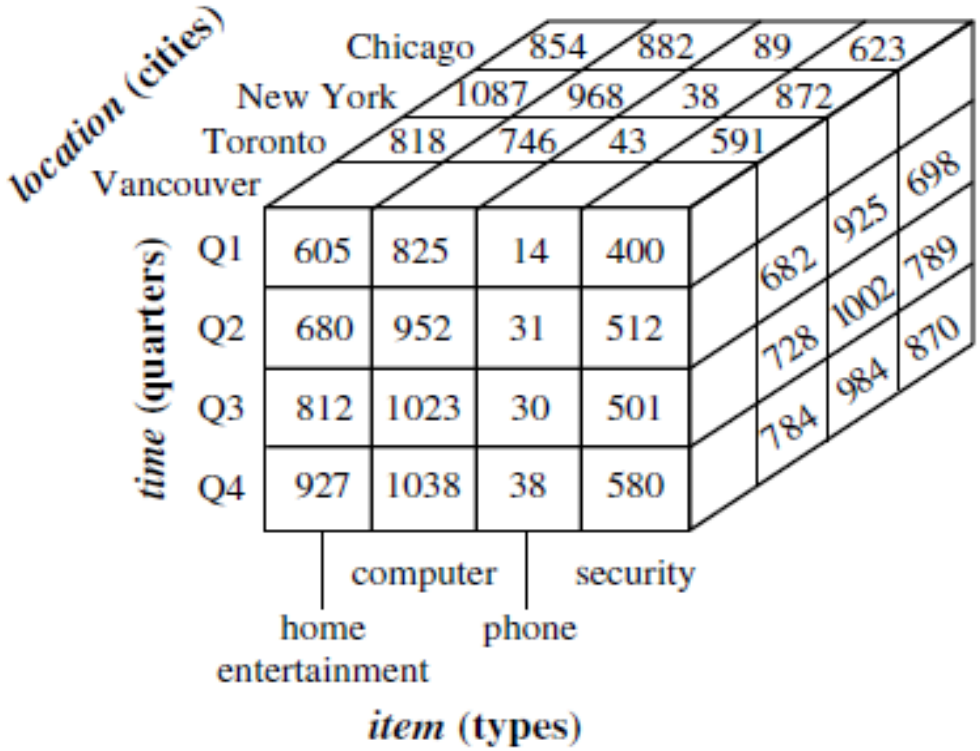
A 2-D view of sales data for *AllElectronics* according to the dimensions *time* and *item*, where the sales are from branches located in the city of Vancouver. The measure displayed is *dollars_sold* (in thousands).

	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

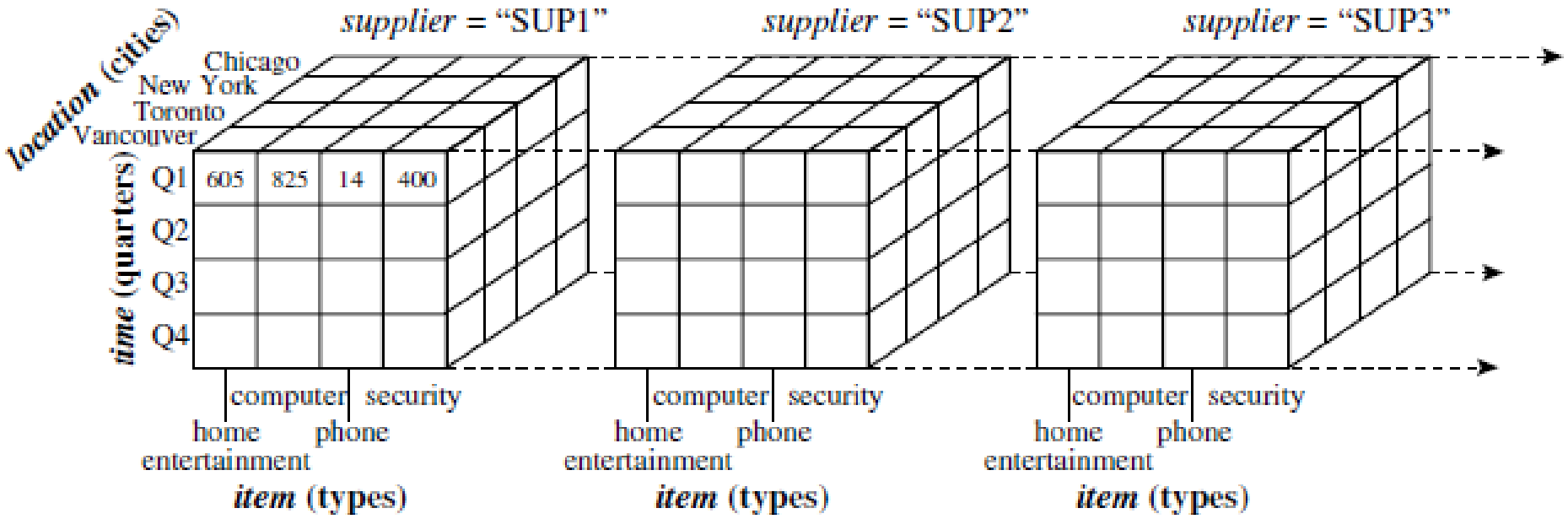
3D Data

Table 3.3 A 3-D view of sales data for *AllElectronics*, according to the dimensions *time*, *item*, and *location*. The measure displayed is *dollars_sold* (in thousands).

	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



gure 3.1 A 3-D data cube representation of the data in Table 3.3, according to the dimensions *time*, *item*, and *location*. The measure displayed is *dollars_sold* (in thousands).



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

- Suppose that we would now like to view our sales data with an additional fourth dimension, such as *supplier*.
- we can think of a 4-D cube as being a series of 3-D cubes, as shown in the above Figure

- If we continue in this way, we may display any *n*-D data as a series of (*n*-1)-D "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 such as each of the above 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 forma *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.

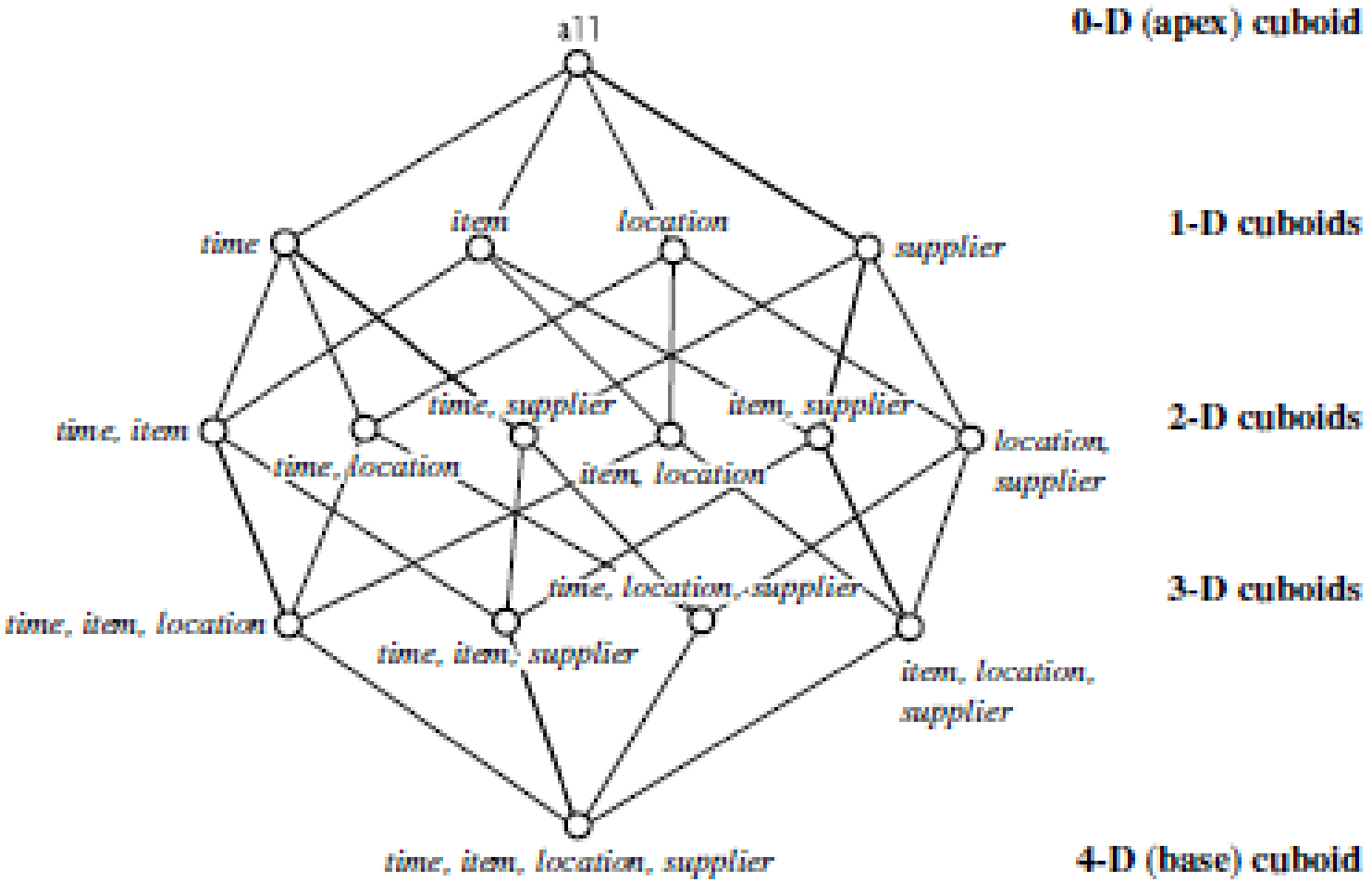


Figure 3.3 Lattice of cuboids, making up a 4-D data cube for the dimensions *time*, *item*, *location*, and *supplier*. Each cuboid represents a different degree of summarization.

- 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.
- 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.

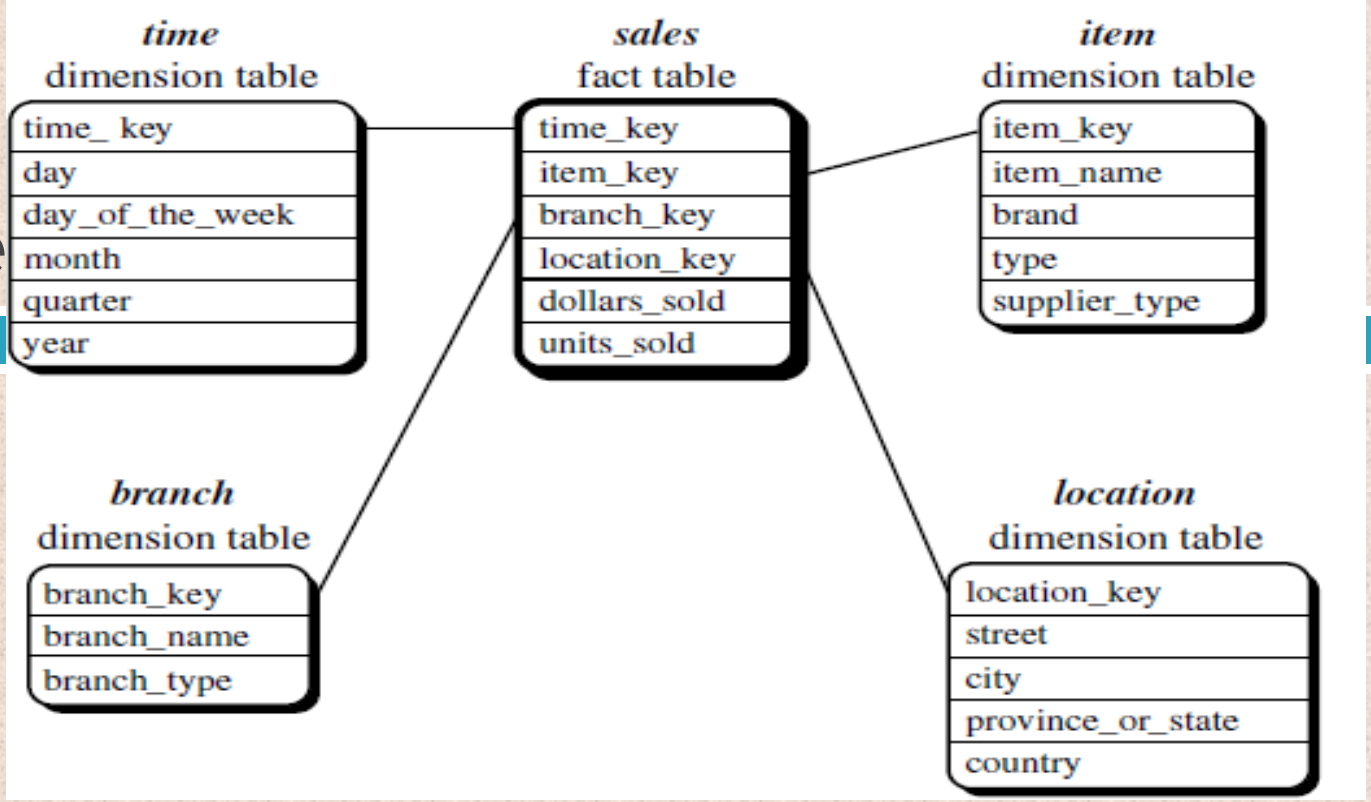
Stars, Snowflakes, and Fact Constellations: Schemas for Multidimensional Databases

- 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 on-line transaction processing.
- A data warehouse, however, requires a concise, subject-oriented schema that facilitates on-line data analysis.
- The most popular data model for a data warehouse is a multidimensional model.
- Such a model can exist in the form of 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.
  - The schema graph resembles a starburst, with the dimension tables displayed in a radial pattern around the central fact table

Example



- A star schema for *AllElectronics* sales is shown in the above Figure.
- Sales are considered along four dimensions, namely, *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 (such as *time key* and *item key*) are system-generated identifiers.

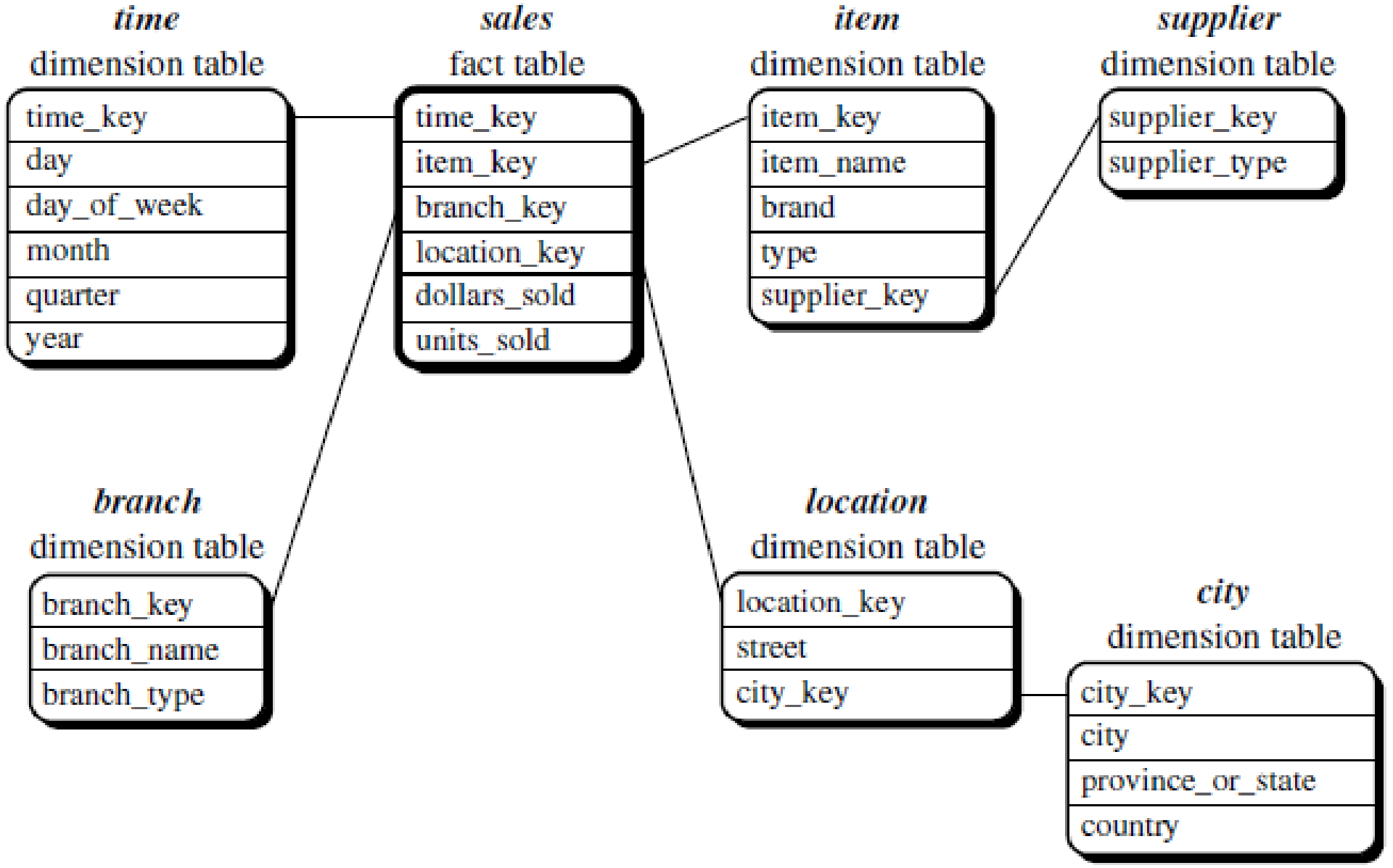
- in the star schema, each dimension is represented by only one table, and each table contains a set of attributes.
- For example, the *location* dimension table contains the attribute set f*location key, street, city, province or state, country*g. This constraint may introduce some redundancy.

- For example, "*Vancouver*" and "*Victoria*" are both cities in the Canadian province of British Columbia.
- Entries for such cities in the *location* dimension table will create redundancy among the attributes *province or state* and *country*, that is, (..., *Vancouver, British Columbia, Canada*) and (..., *Victoria, British Columbia, Canada*).
- 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.

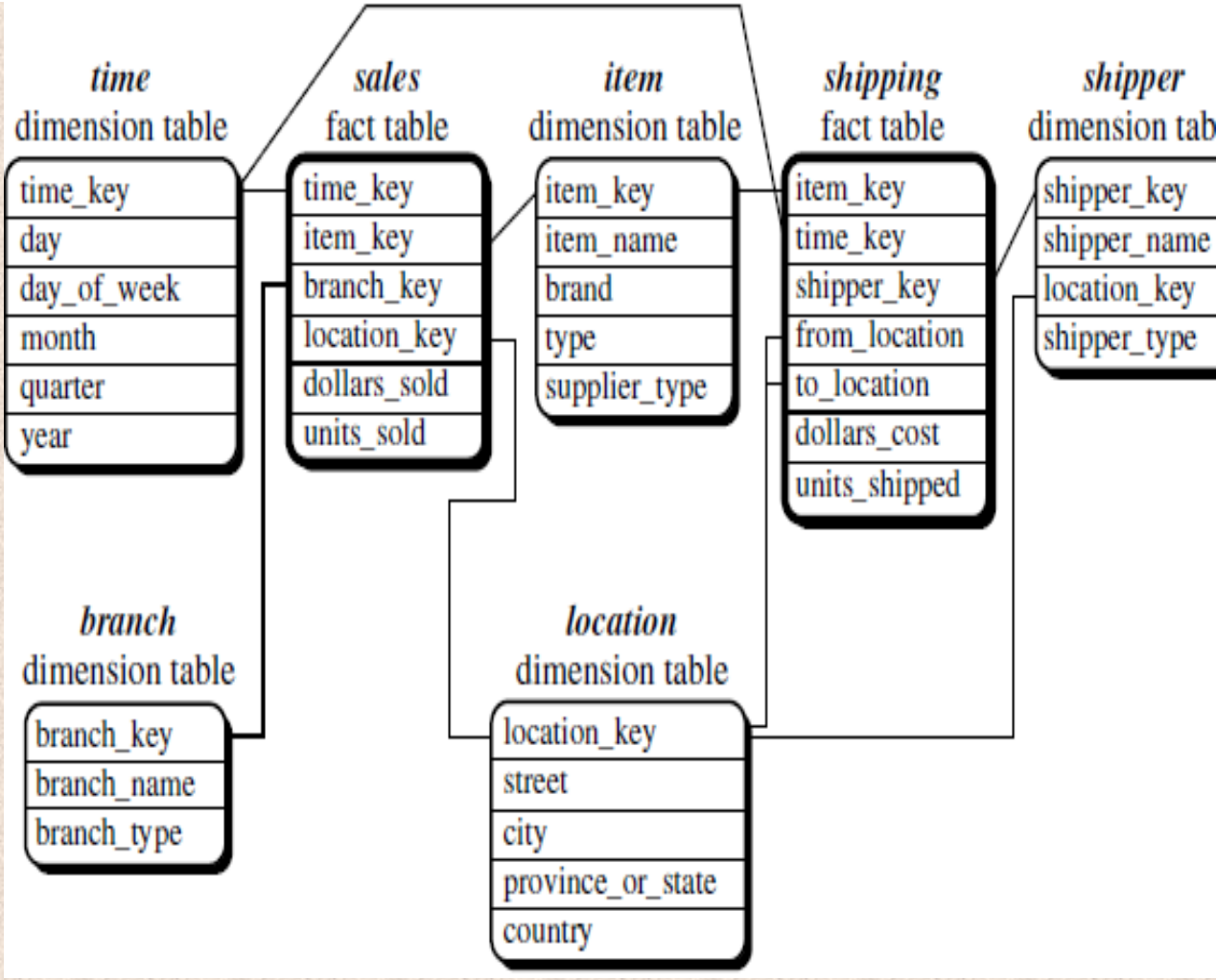
- This saving of space is negligible in comparison to the typical magnitude of the fact table.
- Furthermore, the snowflake structure can reduce the effectiveness of browsing, since more joins will be needed to execute a query.
- Consequently, the system performance may be adversely impacted.
- Hence, although the snowflake schema reduces redundancy, it is not as popular as the star schema in data warehouse design.



Snowflake schema of a data warehouse for sales.

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.



Data Warehouse & Data Mart

- 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.

Data Mart

- A data mart, is a department subset of the data warehouse that focuses on selected subjects, and thus its scope is *department wide*.
- For data marts, the *star* or *snowflake* schema are commonly used, since both are geared toward modeling single subjects, although the star schema is more popular and efficient.

"How can we define a multidimensional schema?"

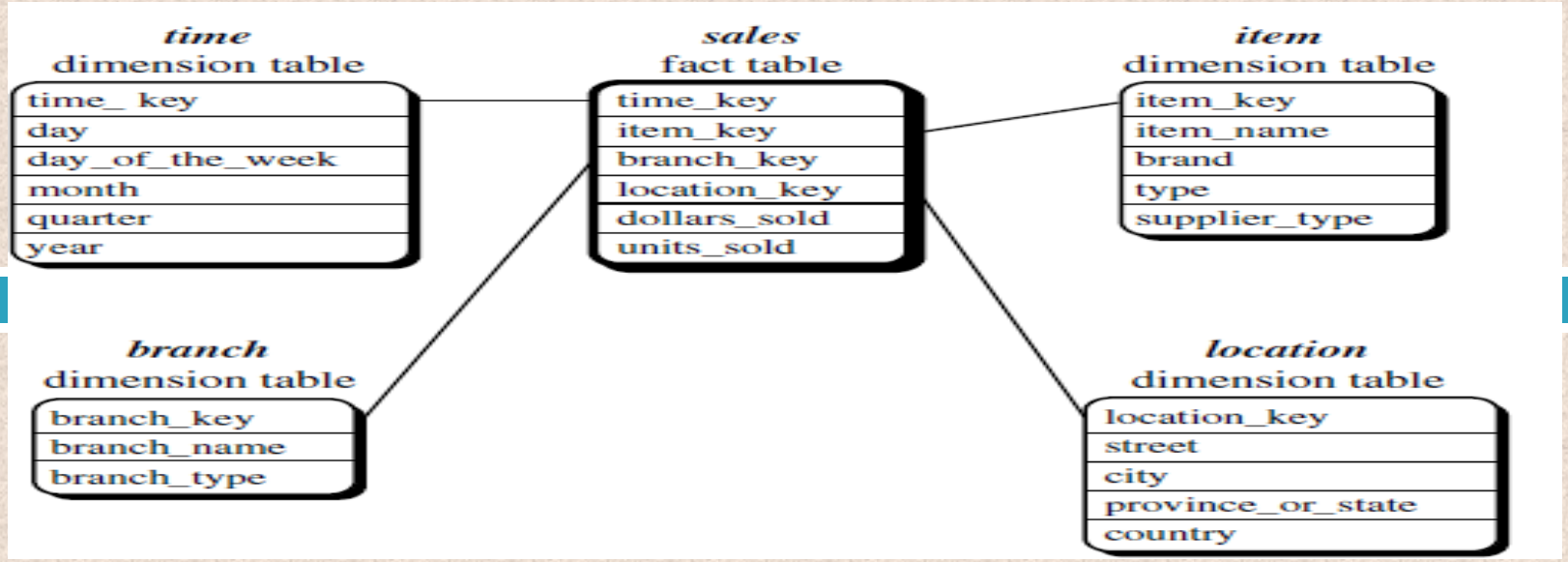
- Just as relational query languages like SQL can be used to specify relational queries, a data mining query language can be used to specify data mining tasks.
- In particular, we examine how to define data warehouses and data marts in our SQL-based data mining query language, DMQL.
- Data warehouses and data marts can be defined using two language primitives, one for *cube definition* and one for *dimension definition*.

- The *cube definition* statement has the following syntax

define cube ⟨cube_name⟩ [⟨dimension_list⟩]: ⟨measure_list⟩

The *dimension definition* statement has the following syntax:

define dimension ⟨dimension_name⟩ as (⟨attribute_or_dimension_list⟩)

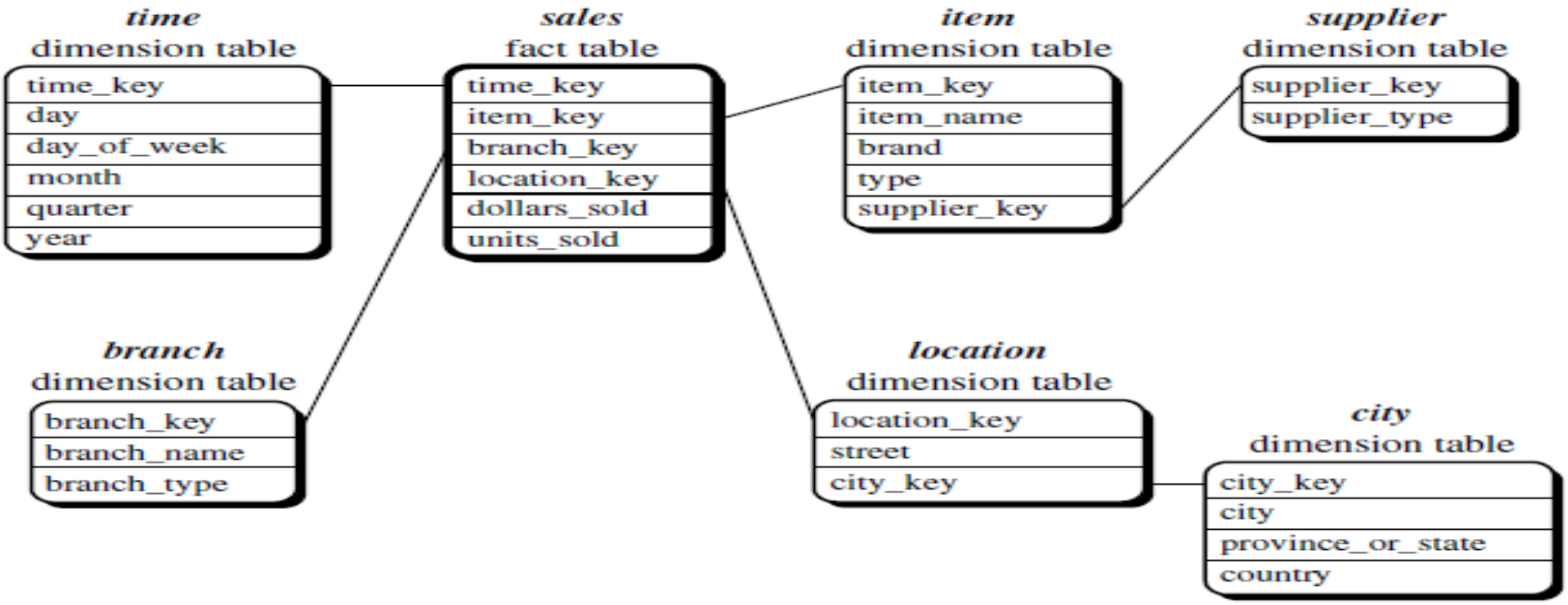


- The star schema of Figure is defined in DMQL as follows:

```
define cube sales_star [time, item, branch, location]:
        dollars_sold = sum(sales_in_dollars), units_sold = count(*)
define dimension time as (time_key, day, day_of_week, month, quarter, year)
define dimension item as (item_key, item_name, brand, type, supplier_type)
define dimension branch as (branch_key, branch_name, branch_type)
define dimension location as (location_key, street, city, province_or_state,
        country)
```

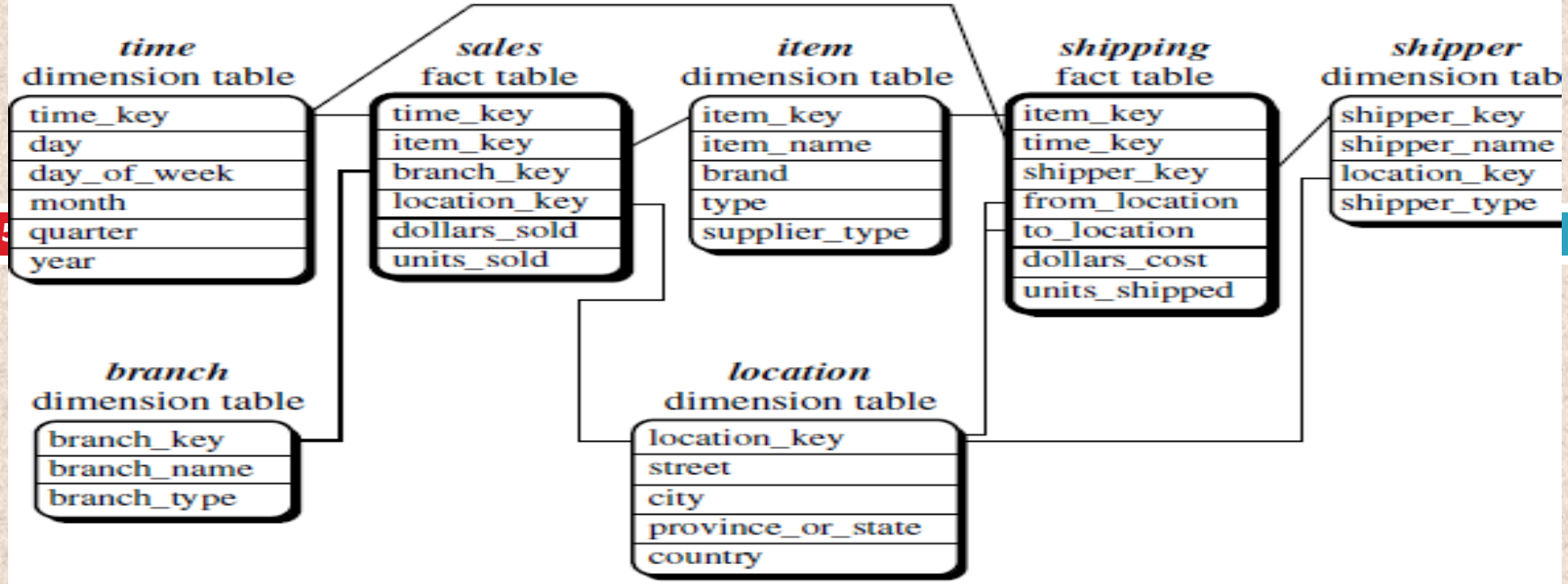
```
define cube sales_star [time, item, branch, location]:
        dollars_sold = sum(sales_in_dollars), units_sold = count(*)
define dimension time as (time_key, day, day_of_week, month, quarter, year)
define dimension item as (item_key, item_name, brand, type, supplier_type)
define dimension branch as (branch_key, branch_name, branch_type)
define dimension location as (location_key, street, city, province_or_state,
        country)
```

- The define cube statement defines a data cube called *sales star*, which corresponds to the central *sales* fact table.
- This command specifies the dimensions and the two measures, *dollars sold* and *units sold*.
- The data cube has four dimensions, namely, *time*, *item*, *branch*, and *location*. A define dimension statement is used to define each of the dimensions.




```
define cube sales_snowflake [time, item, branch, location]:
        dollars_sold = sum(sales_in_dollars), units_sold = count(*)
define dimension time as (time_key, day, day_of_week, month, quarter, year)
define dimension item as (item_key, item_name, brand, type, supplier
        (supplier_key, supplier_type))
define dimension branch as (branch_key, branch_name, branch_type)
define dimension location as (location_key, street, city
        (city_key, city, province_or_state, country))
```

- The snowflake schema of Figure is defined in DMQL is shown above



define cube sales [time, item, branch, location]:
dollars_sold = sum(sales_in_dollars), units_sold = count(*)
define dimension time as (time_key, day, day_of_week, month, quarter, year)
define dimension item as (item_key, item_name, brand, type, supplier_type)
define dimension branch as (branch_key, branch_name, branch_type)
define dimension location as (location_key, street, city, province_or_state, country)

define cube shipping [time, item, shipper, from_location, to_location]:
dollars_cost = sum(cost_in_dollars), units_shipped = count(*)
define dimension time as time in cube sales
define dimension item as item in cube sales
define dimension shipper as (shipper_key, shipper_name, location as location in cube sales, shipper_type)
define dimension from_location as location in cube sales
define dimension to_location as location in cube sales

Measures: Their Categorization and Computation

- Measures can be organized into three categories (i.e., distributive, algebraic, 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, count() can be computed for a data cube by first partitioning the cube into a set of subcubes, computing count() for each subcube, and then summing up the counts obtained for each subcube.
- Hence, count() is a distributive aggregate function.
- sum(), min(), and max() are distributive aggregate functions.
- A measure is *distributive* if it is obtained by applying a distributive aggregate function.
- Distributive measures can be computed efficiently because they can be computed in a distributive manner.

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.
- A measure is *algebraic* if it is obtained by applying an algebraic 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.

Holistic

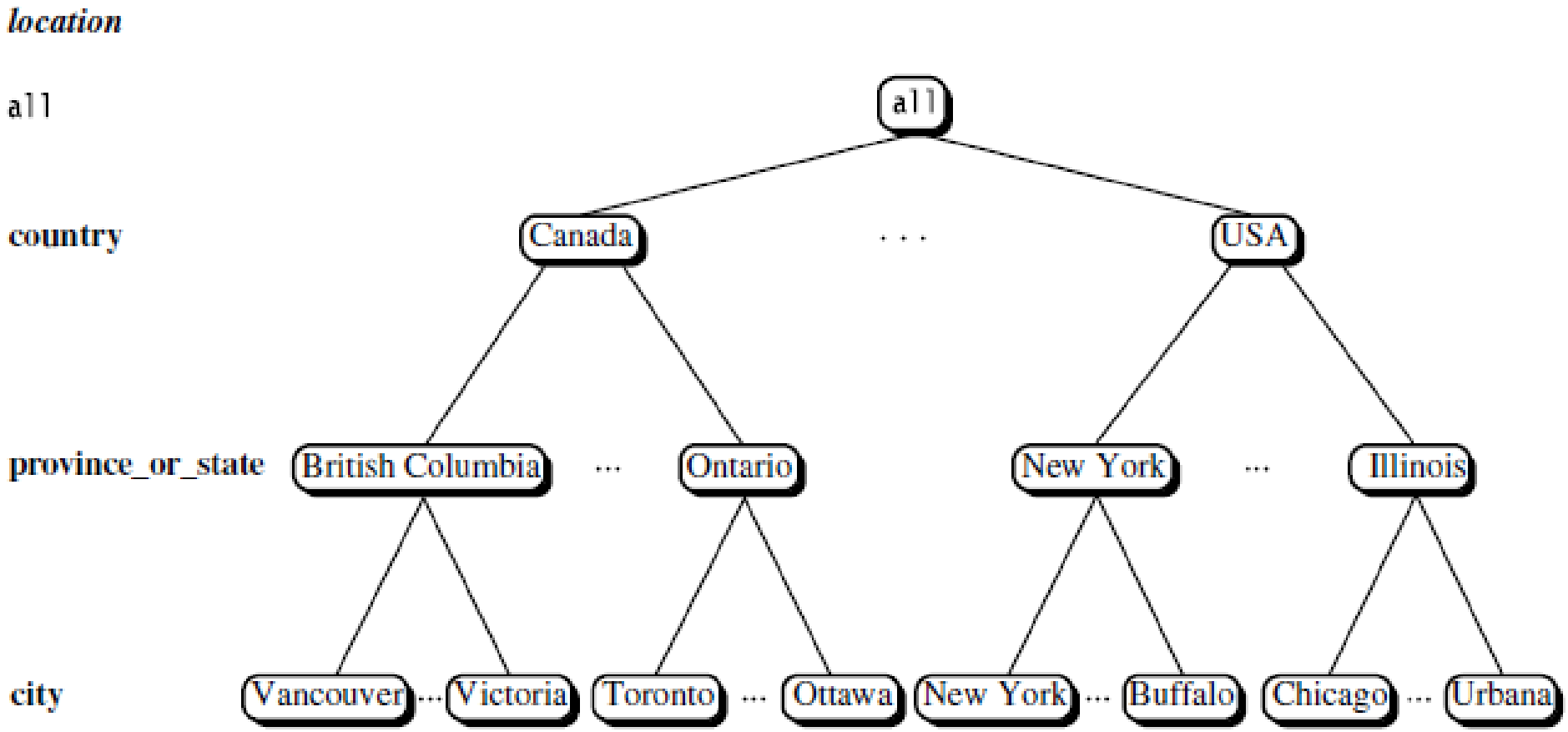
- An aggregate function is *holistic* if there is no constant bound on the storage size needed to describe a sub aggregate.
- That is, there does not exist an algebraic function with *M* arguments (where *M* is a constant) that characterizes the computation.
- Common examples of holistic functions include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

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, NewYork, andChicago.
- 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 to which they belong, such as Canada or the USA.
- These mappings forma 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).
- The concept hierarchy described above is illustrated in the following Figure

Concept heirarachy

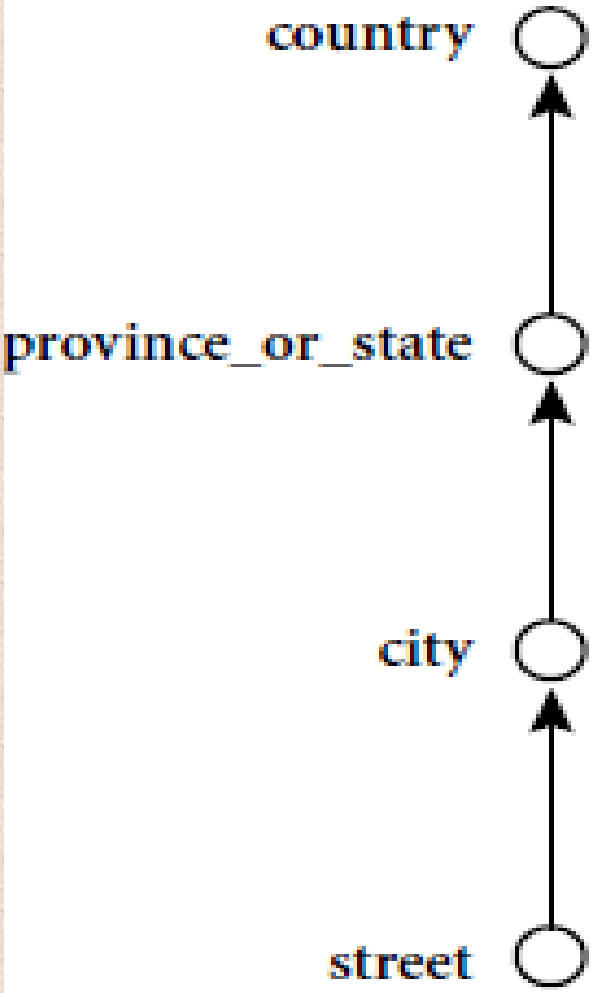


A concept hierarchy for the dimension *location*. Due to space limitations, not all of the nodes of the hierarchy are shown (as indicated by the use of "ellipsis" between nodes).

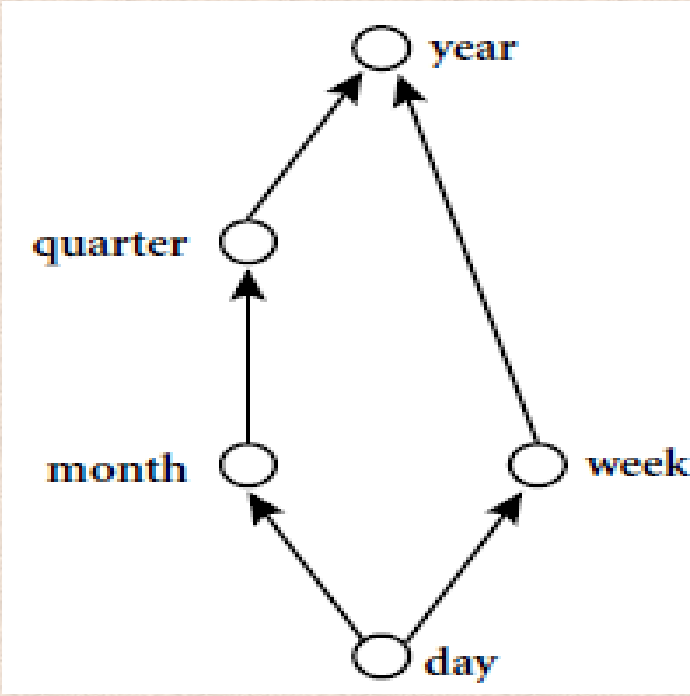
- Many concept hierarchies are implicit within the database schema.
- For example, suppose that the dimension *location* is described by the attributes *number, street, city, province or state, zipcode* and *country*.
- These attributes are related by a totalorder, forming

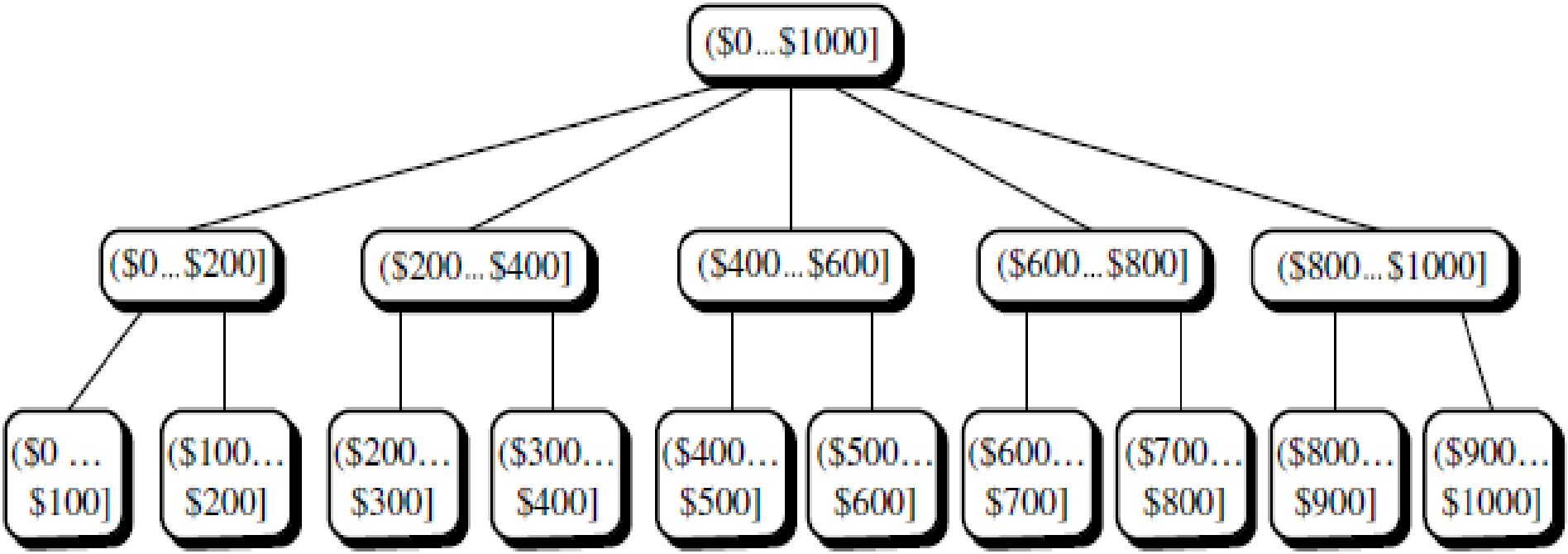
a concept hierarchy such as "*street* < *city* < *province or state* < *country*".

This hierarchy is shown in the Figure



- 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* < [*month* <*quarter; week*} < *year*"
- This lattice structure is shown in Figure
- A concept hierarchy that is a total or partial order among attributes in a database schema is called a schema hierarchy

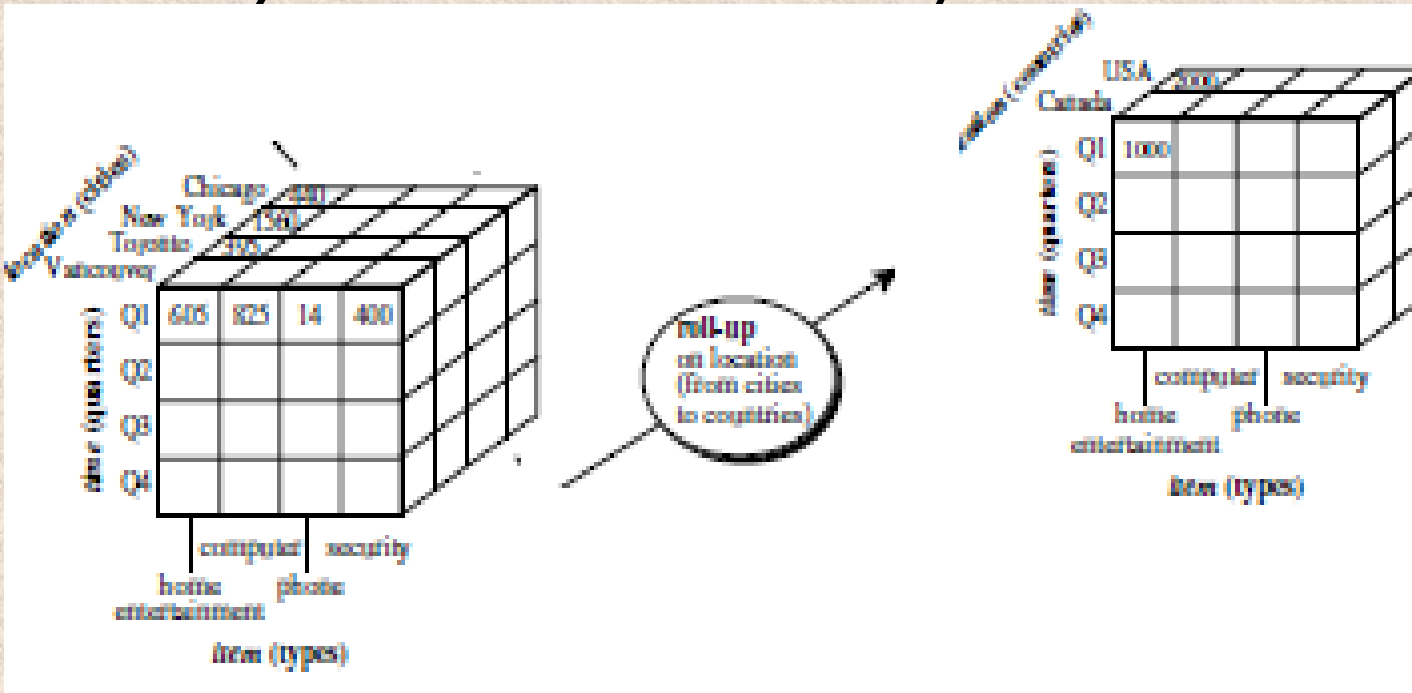




A concept hierarchy for the attribute *price*.

OLAP Operations in the Multidimensional Data Model

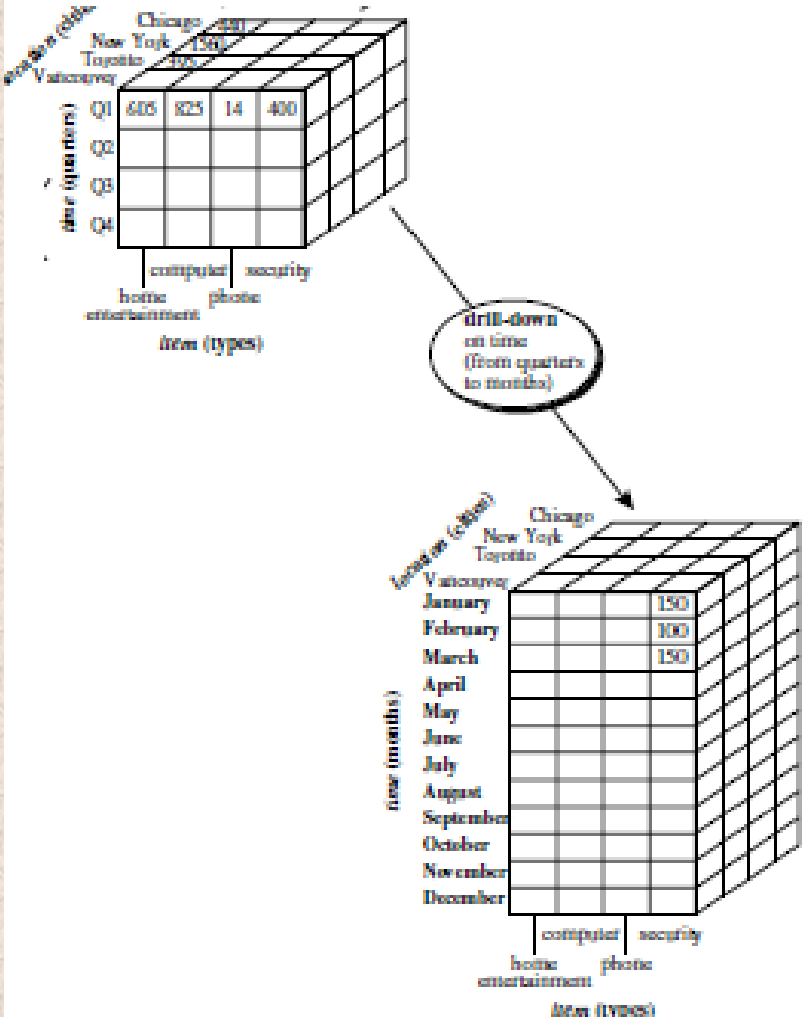
- Roll-up: The roll-up operation (also called the *drill-up* operation by some vendors) performs aggregation on a data cube, either by *climbing up a concept hierarchy* for a dimension or by *dimension reduction.*



- 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

Drill-down

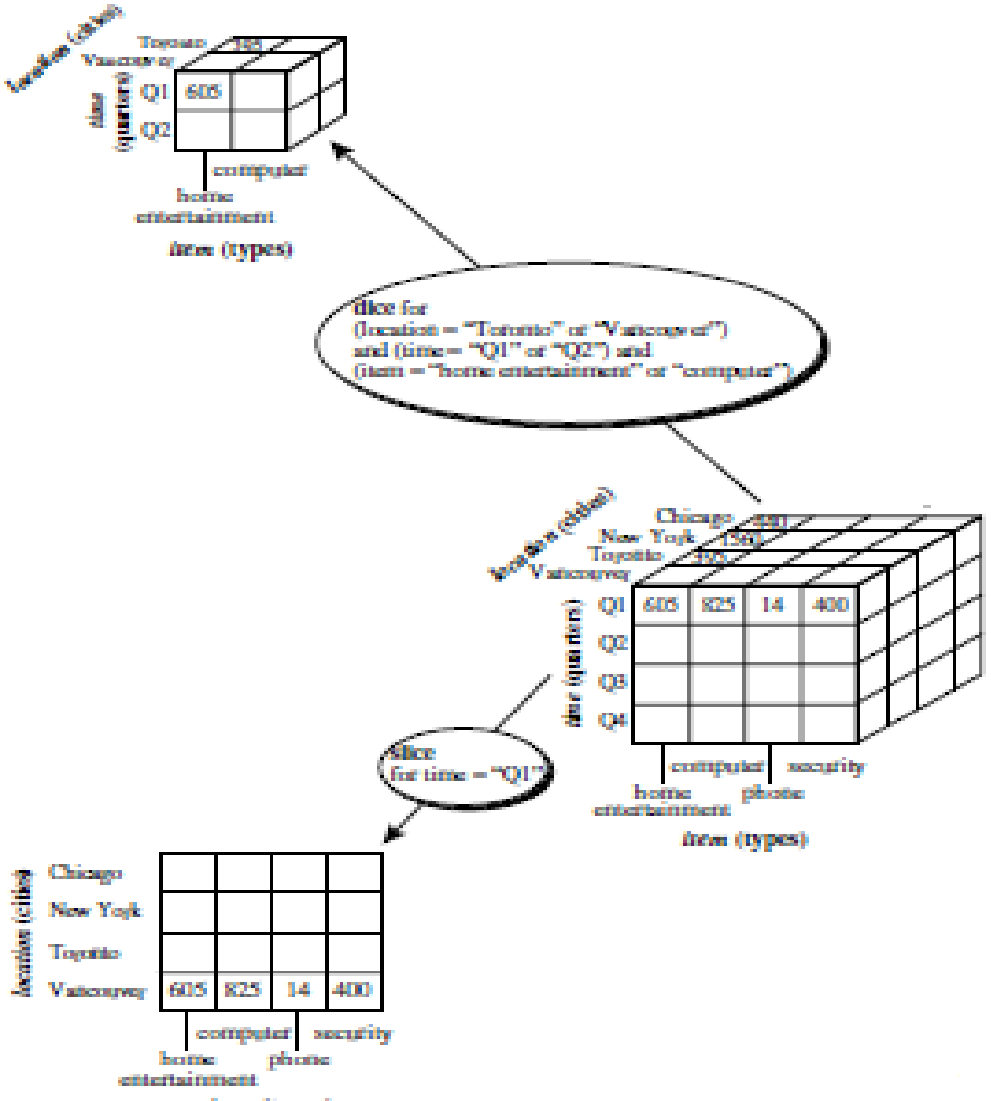
- Drill-down is the reverse of roll-up.
- It navigates from less detailed data to more detailed data.
- Drill-down can be realized by either *stepping down a concept hierarchy* for a dimension or *introducing additional dimensions*.
- Figure shows the result of a drill-down operation performed on the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



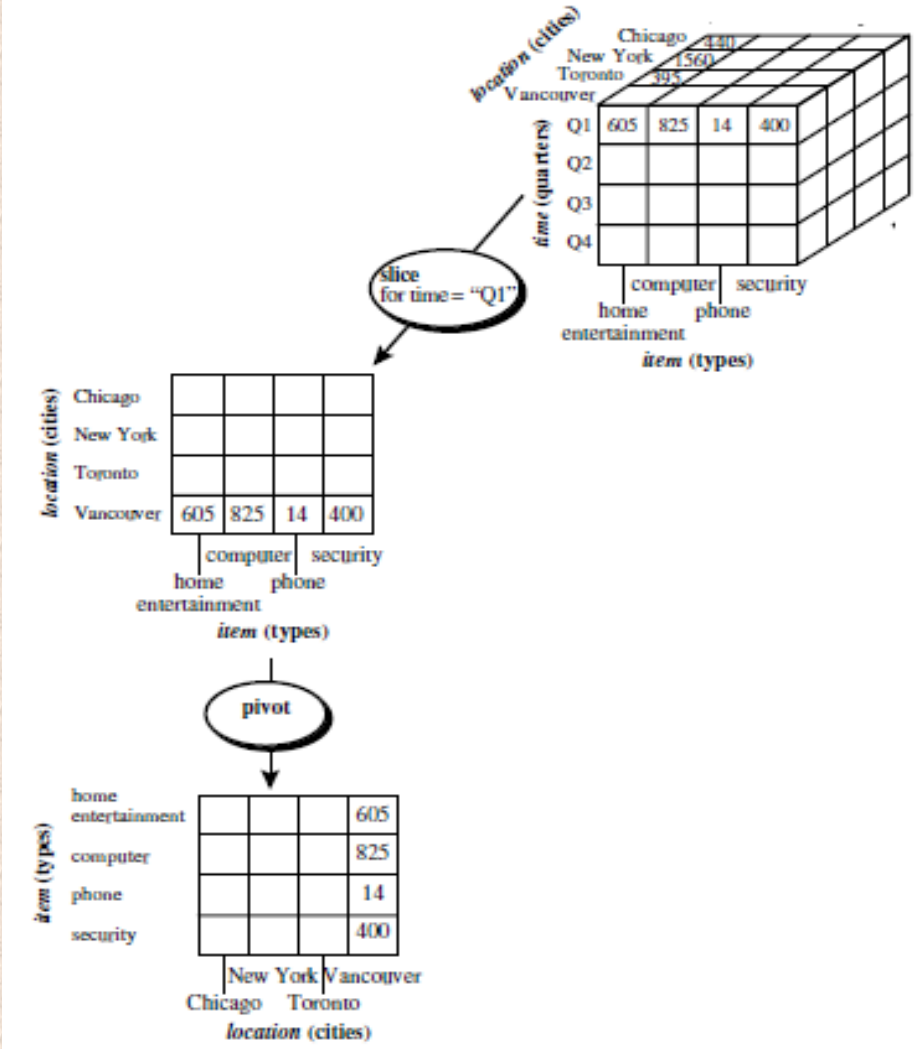
- Drill-down occurs by descending the *time* hierarchy from the level of *quarter* to the more detailed level of *month*.
- The resulting data cube details the total sales per month rather than summarizing them by quarter.
- Because a drill-down adds more detail to the given data, it can also be performed by adding new dimensions to a cube

Slice and dice

- The *slice* operation performs a selection on one dimension of the given cube, resulting in a subcube.
- The *dice* operation defines a subcube by performing a selection on two or more dimensions



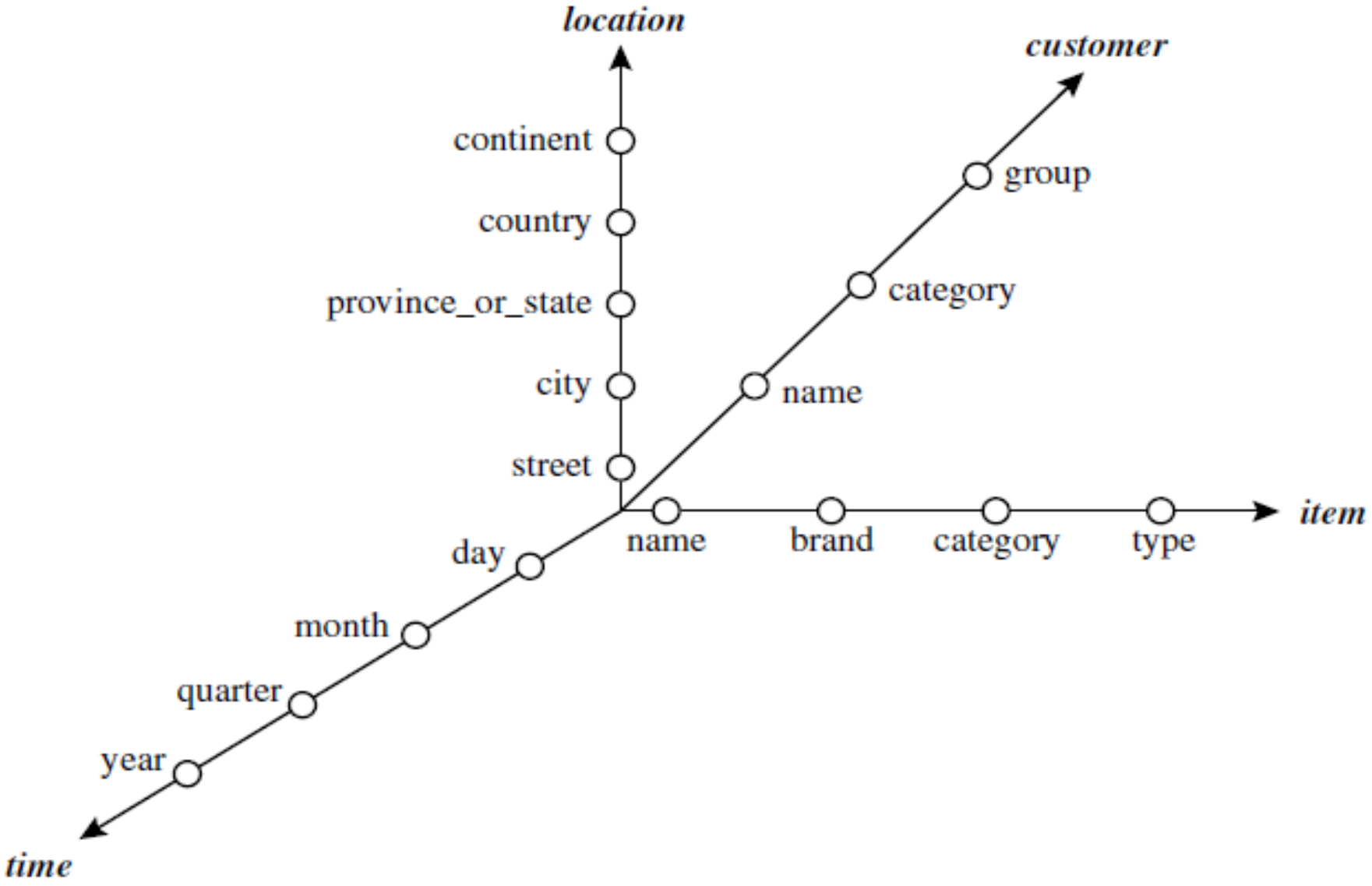
- Pivot (rotate): *Pivot* (also called *rotate*) is a visualization operation that rotates the data axes in view in order to provide an alternative presentation of the data.
- Figure shows a pivot operation where the *item* and *location* axes in a 2-D slice are rotated



- 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.

A Starnet Query Model for Querying Multidimensional Databases

- The querying of multidimensional databases can be based on a starnet model.
- A starnet model consists of radial lines emanating froma 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.



Modeling business queries: a starnet model.

DataWarehouse Architecture

- Four different views regarding the design of a data warehousemust be considered: the
  - *top-down view*,
  - the *data source view*,
  - the *data warehouse view*, and
  - the *business query view*.

- The top-down view allows the selection of the relevant information necessary for the data warehouse.
- This information matches the current and future business needs.
- The data source view exposes the information being captured, stored, and managed by operational systems. This information may be documented at various levels of detail and accuracy, from individual data source tables to integrated data source tables.

- Data sources are often modeled by traditional data modeling techniques, such as the entity-relationship model or CASE (computer-aided software engineering) tools.
- The data warehouse view includes fact tables and dimension tables.
- It represents the information that is stored inside the data warehouse, including pre calculated totals and counts, as well as information regarding the source, date, and time of origin, added to provide historical context.
- Finally, the business query view is the perspective of data in the data warehouse from the viewpoint of the end user.

The Process of Data Warehouse Design

- 1. Choose a *business process* to model, for example, orders, invoices, shipments, inventory, account administration, sales, or the general ledger.
- If the business process is organizational and involves multiple complex object collections, a data warehouse model should be followed.
- However, if the process is departmental and focuses on the analysis of one kind of business process, a data mart model should be chosen.

2. Choose the *grain* of the business process.

- The grain is the fundamental, atomic level of data to be represented in the fact table for this process, for example, individual transactions, individual daily snapshots, and so on.

3. Choose the *dimensions* that will apply to each fact table record.

- Typical dimensions are time, item, customer, supplier, warehouse, transaction type, and status.

- Choose the *measures* that will populate each fact table record.
- Typical measures are numeric additive quantities like *dollars sold* and *units sold*

A Three-Tier Data Warehouse Architecture

- 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 (such as customer profile information provided by external consultants).
- These tools and utilities perform data extraction, cleaning, and transformation (e.g., to merge similar data from different sources into a unified format), as well as load and refresh functions to update the data warehouse.

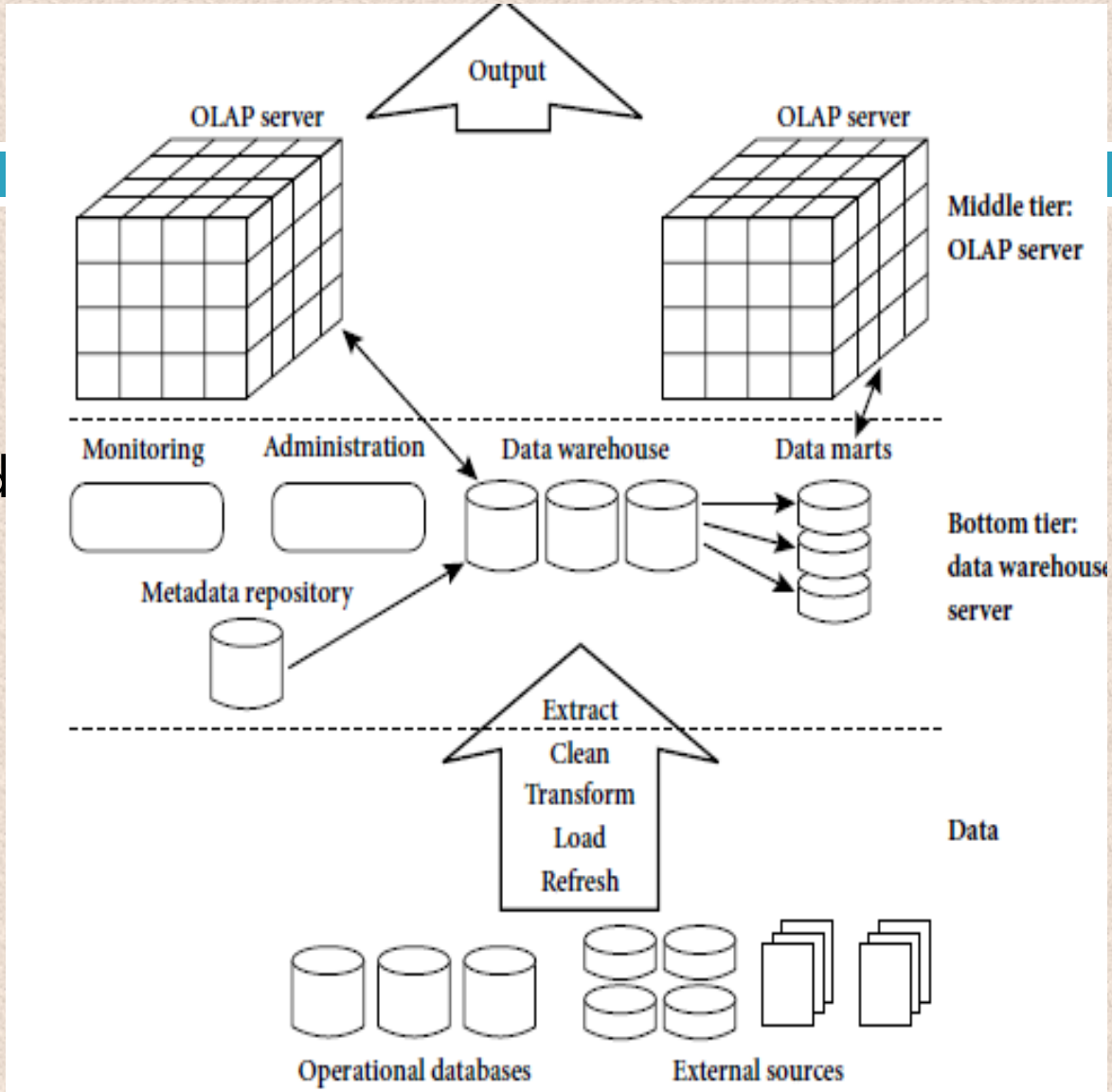
- The data are extracted using application program interfaces known as gateways.
- A gateway is supported by the underlying DBMS and allows client programs to generate SQL code to be executed at a server.
- Examples of gateways include ODBC (Open Database Connection) and OLEDB (Open Linking and Embedding for Databases) by Microsoft and JDBC (Java Database Connection).
- This tier also contains a metadata repository, which stores information about the data warehouse and its contents.

- The middle tier is an OLAP server that is typically implemented using either

(1) a relational OLAP (ROLAP) model, that is, an extended relational DBMS that maps operations on multidimensional data to standard relational operations; or

(2) a multidimensional OLAP (MOLAP) model, that is, a special-purpose server that directly implements multidimensional data and operations

- 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).



.12 A three-tier data warehousing architecture.

- 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

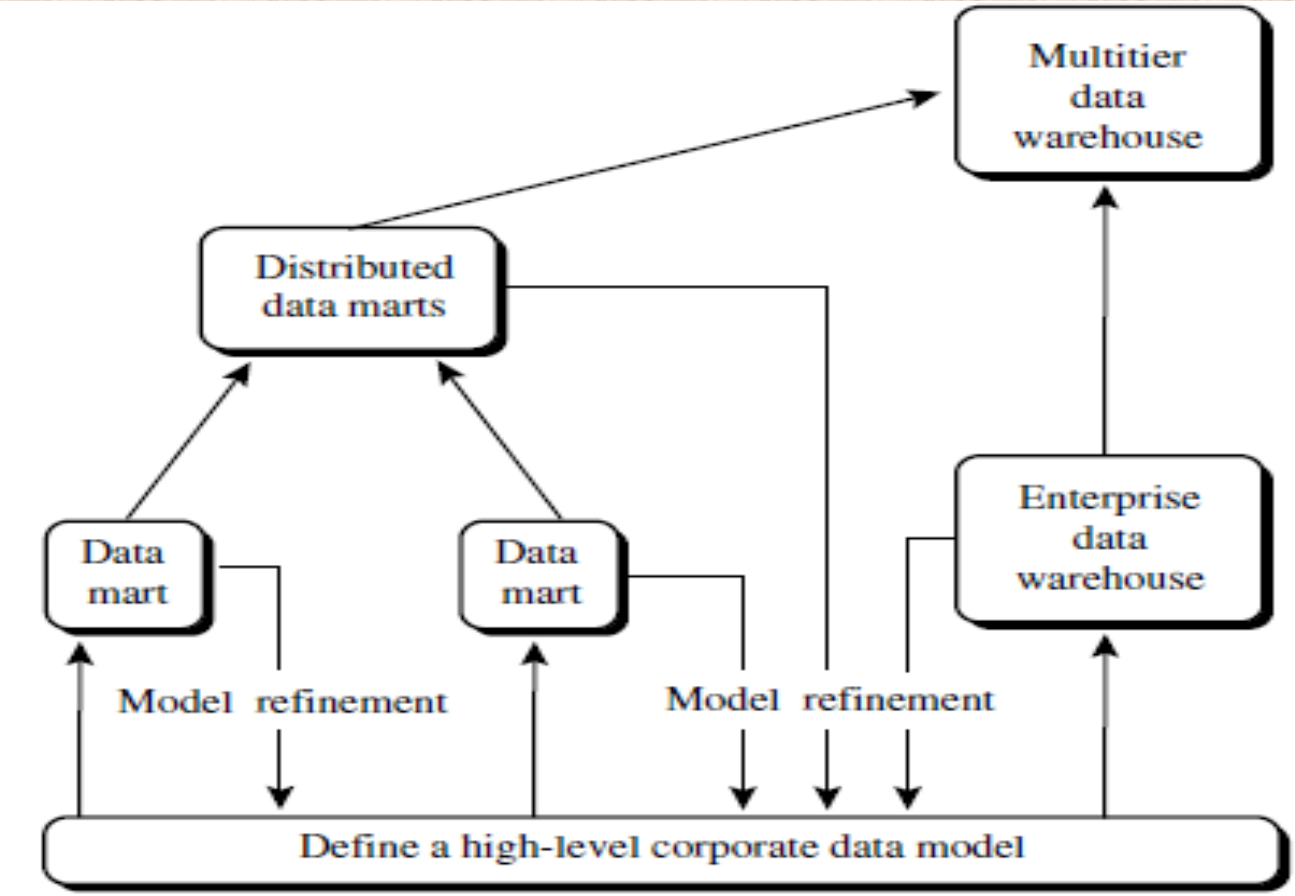
- A virtual warehouse is a set of views over operational databases.
- For efficient query processing, only some of the possible summary views may be materialized.
- A virtual warehouse is easy to build but requires excess capacity on operational database servers.

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

- The top-down development of an enterprise warehouse serves as a systematic solution and minimizes integration problems.
- However, it is expensive, takes a long time to develop, and lacks flexibility due to the difficulty in achieving consistency and consensus for a common data model for the entire organization.

- The bottom-up approach to the design, development, and deployment of independent data marts provides flexibility, low cost, and rapid return of investment.
- It, however, can 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, as shown in Figure



A recommended approach for data warehouse development.

- 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.
- This high-level model, although it will need to be refined in the further development of enterprise data warehouses and departmental data marts, will greatly reduce future integration problems.
- Second, independent data marts can be implemented in parallel with the enterprise warehouse based on the same corporate data model set as above.

- 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.

Data Warehouse Back-End Tools and Utilities

- 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
- Time stamping 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 structure of the data warehouse*, 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

- *The 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

Types of OLAP Servers: 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-relationalDBMS* 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
- views of data through *array-based multidimensional storage engines*. They mapmultidimensional
- 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 utilizationmay be lowif the data set is sparse.
- In such cases, sparse matrix compression techniques should be explored (Chapter 4).
- 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 scalability of ROLAP and the faster
- computation of MOLAP. For example, a HOLAP server may allow large volumes
- of detail 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 systemvendors 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

"*How are data actually stored in ROLAP and MOLAP architectures?*"

- ROLAP uses relational tables to store data for on-line analytical processing. Recall that the fact table associated with a base cuboid is referred
- to as a *base fact table*. The base fact table stores data at the abstraction level indicated by the join keys in the schema for the given data cube.
- Aggregated data can also be stored in fact tables, referred to as summary fact tables.

- Some summary fact tables store both
- base fact table data and aggregated data, as in Example.
- Alternatively, separate summary fact tables can be used for each level of abstraction, to store only aggregated data.

Single table for base and summary facts.

RID	item	...	day	month	quarter	year	dollars_sold
1001	TV	...	15	10	Q4	2003	250.60
1002	TV	...	23	10	Q4	2003	175.00
...	...	...	...	...	...	...	...
5001	TV	...	all	10	Q4	2003	45,786.08
...	...	...	...	...	...	...	...

- The above table shows a summary fact table that contains both base fact data and aggregated data. The schema of the table is "(*record identifier (RID), item, . . . , day, month, quarter, year, dollars sold*)", where *day, month, quarter*, and *year* define the date of sales, 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 date of sales is October 15, 2003, and October 23, 2003, respectively.

Single table for base and summary facts.

RID	item	...	day	month	quarter	year	dollars_sold
1001	TV	...	15	10	Q4	2003	250.60
1002	TV	...	23	10	Q4	2003	175.00
...	...	...	...	...	...	...	...
5001	TV	...	all	10	Q4	2003	45,786.08
...	...	...	...	...	...	...	...

- 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 2003.
- That is, the *dollars sold* amount shown is an aggregation representing the entire month of October 2003, rather than just October 15 or 23, 2003.
- The special value all is used to represent subtotals in summarized data.

- MOLAP uses multidimensional array structures to store data for on-line analytical processing.

DataWarehouse Implementation

- Data warehouses contain huge volumes of data.
- OLAP servers demand that decision support queries be answered in the order of seconds.
- Therefore, it is crucial for data warehouse systems to support highly efficient cube computation techniques, access methods, and query processing techniques. In this section, we present an overview of methods for the efficient implementation of data warehouse systems.

Efficient Computation of Data Cubes

- At the core of multidimensional data analysis is the efficient computation of aggregations across many sets of dimensions.
- In SQL terms, these aggregations are referred to as group-by's.
- Each group-by can be represented by a *cuboid*, where the set of group-by's forms a lattice of cuboids defining a data cube.
- this section, we explore issues relating to the efficient computation of data cubes.

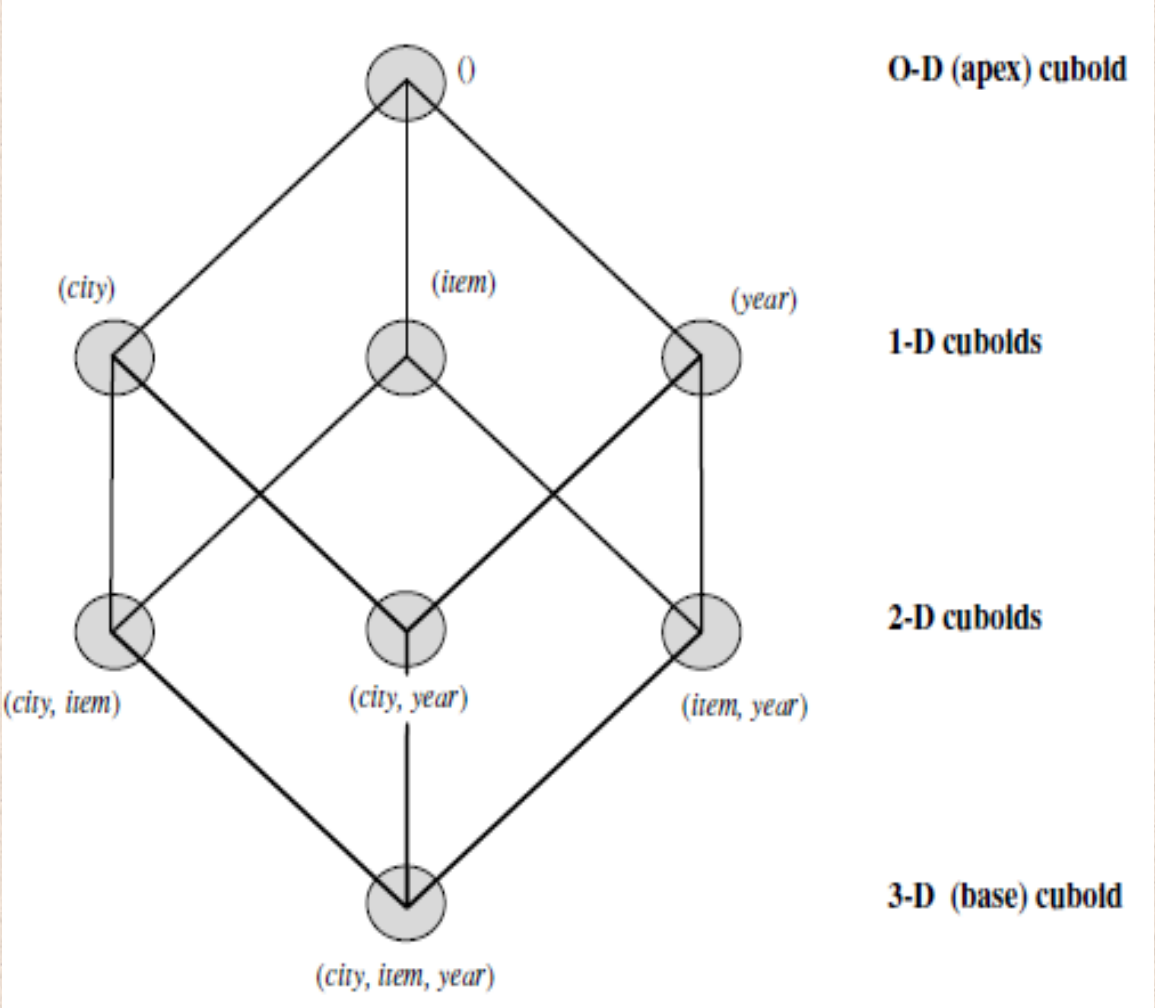
The compute cube Operator and the Curse of Dimensionality

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

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

- **What is the total number of cuboids, or group-by's, that can be computed for this data cube?**
- Taking the three attributes, *city, item*, and *year*, as the dimensions for the data cube, and *sales in dollars* as the measure, the total number of cuboids, or groupby's, that can be computed for this data cube is $2^3 = 8$.
- The possible group-by's are the following: {(*city, item, year*), (*city, item*), (*city, year*), (*item, year*), (*city*), (*item*), (*year*), ()} , where () means that the group-by is empty (i.e., the dimensions are not grouped).

- These group-by's form a lattice of cuboids for the data cube, as shown in Figure.
- The base cuboid contains all three dimensions, *city, item*, and *year*



- The apex cuboid, or 0-D cuboid, refers to the case where the group-by is empty.
- It contains the total sum of all sales.
- The base cuboid is the least generalized (most specific) of the cuboids.
- The apex cuboid is the most generalized (least specific) of the cuboids, and is often denoted as all.
- If we start at the apex cuboid and explore downward in the lattice, this is equivalent to drilling down within the data cube.
- If we start at the base cuboid and explore upward, this is akin to rolling up.

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

- The data cube could be defined as

define cube sales cube [city, item, year]: sum(sales in dollars)

- For a cube with *n* dimensions, there are a total of 2^n cuboids, including the base cuboid.
- A statement such as compute cube sales cube would explicitly instruct the system to compute the sales aggregate cuboids for all of the eight subsets of the set {*city, item, year*}, including the empty subset.

- On-line analytical processing may need to access different cuboids for different queries.
- Therefore, it may seem like a good idea to compute all or at least some of the cuboids in a data cube in advance.
- Precomputation leads to fast response time and avoids some redundant computation.
- Most, if not all, OLAP products resort to some degree of precomputation of multidimensional aggregates.

- A major challenge related to this pre-computation, however, is that the required storage space may explode if all of the cuboids in a data cube are pre-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.

"*How many cuboids are there in an* n-*dimensional data cube?*"

- If there were no hierarchies associated with each dimension, then the total number of cuboids for an *n*-dimensional data cube, as we have seen above, is 2^n.
- However, in practice, many dimensions do have hierarchies.
- For example, the dimension *time* is usually not explored at only one conceptual level, such as *year*, but rather at multiple conceptual levels, such as in the hierarchy "*day* < *month* < *quarter* < *year*".

$$Total\ number\ of\ cuboids = \prod_{i=1}^{n}(L_i + 1),$$

- For an *n*-dimensional data cube, the total number of cuboids that can be generated (including the cuboids generated by climbing up the hierarchies along each dimension) is given above.
- where *Li* is the number of levels associated with dimension *i*.
- One is added to *Li* in Equation to include the *virtual* top level, all. (Note that generalizing to all is equivalent to the removal of the dimension.)
- This formula is based on the fact that, at most, one abstraction level in each dimension will appear in a cuboid

- For example, the time dimension as specified above has 4 conceptual levels, or 5 if we include the virtual level all.
- If the cube has 10 dimensions and each dimension has 5 levels (including all), the total number of cuboids that can be generated is $5^{10} = 9.8 \times 10^6$.

- The size of each cuboid also depends on the *cardinality* (i.e., number of distinct values) of each dimension
- For example, if the *AllElectronics* branch in each city sold every item, there would be $|city| \times |item|$ tuples in the *city-item* group-by alone.
- As the number of dimensions, number of conceptual hierarchies, or cardinality increases, the storage space required for many of the group-by's will grossly exceed the (fixed) size of the input relation.

Partial Materialization: Selected Computation of Cuboids

- It is unrealistic to precompute and materialize all of the cuboids that can possibly be generated for a data cube (or from a base cuboid).
- If there are many cuboids, and these cuboids are large in size, a more reasonable option is *partial materialization*, that is, to materialize only *some* of the possible cuboids that can be generated.
- 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.
- 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.

- The selection of the subset of cuboids or subcubes to materialize should take into account the queries in the workload, their frequencies, and their accessing costs.
- In addition, it should consider workload characteristics, the cost for incremental updates, and the total storage requirements.
- The selection must also consider the broad context of physical database design, such as the generation and selection of indices.

- Several OLAP products have adopted heuristic approaches for cuboid and subcube selection.
- A popular approach is to materialize the set of cuboids on which other frequently referenced cuboids are based.
- Alternatively, we can compute an *iceberg cube*, which is a data cube that stores only those cube cells whose aggregate value (e.g., count) is above some minimum support threshold.
- Another common strategy is to materialize a *shell cube*.
- This involves precomputing the cuboids for only a small number of dimensions (such as 3 to 5) of a data cube.
- Queries on additional combinations of the dimensions can be computed on-the-fly.

Indexing OLAP Data

- To facilitate efficient data accessing, most data warehouse systems support index structures and materialized views (using cuboids).
- 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 domain of the attribute.

- If the domain of a given attribute 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

- 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 bitmap index table for *item*.
- 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.

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."

Indexing OLAP data using bitmap indices.

- Figure shows a base (data) table containing the dimensions *item* and *city*, and its mapping to bitmap index tables for each of the dimensions

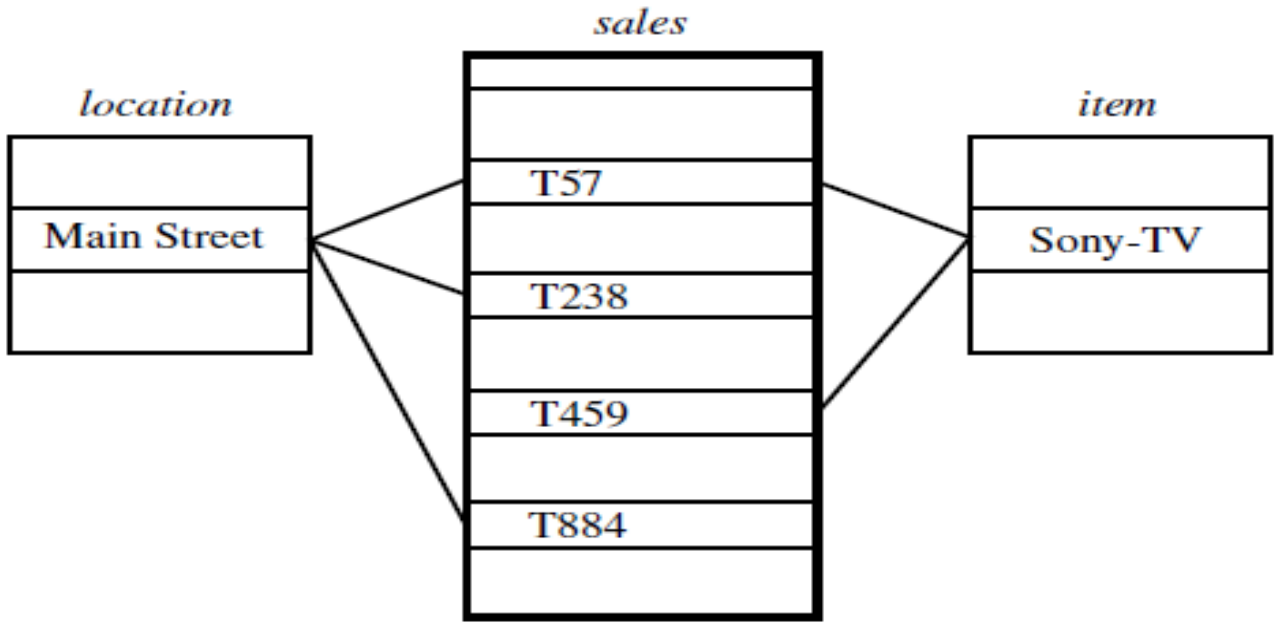
- Bitmap indexing is advantageous compared to hash and tree indices.
- It is especially useful for low-cardinality domains because comparison, join, and aggregation operations are then reduced to bit arithmetic, which substantially reduces the processing time.
- Bitmap indexing leads to significant reductions in space and 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.

join indexing

- The join indexing method gained popularity from its 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 key and its matching primary keys, from the joinable relation.
- The star schema model of data warehouses makes join indexing attractive for cross table search, because the linkage between a fact table and its corresponding dimension tables comprises the foreign key of the fact table and the primary key of the dimension table.

- 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.



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

- We have already defined a star schema for *AllElectronics* of the form

"*sales star* [*time, item, branch, location*]: *dollars sold* = sum *(sales in dollars)*". An example of a join index relationship between the *sales* fact table and the dimension tables for *location* and *item* is shown in Figure

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 two dimensions *location/item/sales*

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

- For example, the "*Main Street*" value in the *location* dimension table joins with tuples T57, T238, and T884 of the *sales* fact table. Similarly, the "*Sony-TV*" value in the *item* dimension table joins with tuples T57 and T459 of the *sales* fact table. The corresponding join index tables are shown in Figure

- 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 dimension tables together.

Best Wishes