# Data Mining:
# Concepts and Techniques

— Chapter 2 —

**Source Slides from Data Mining: Concepts and Techniques**

**-Jiawei Han and Micheleline Kamber**

# Chapter 2: Data Preprocessing

- Descriptive data summarization
- Data cleaning
- Data reduction
- Data integration and transformation
- Discrimination and concept hierarchy generation
- Summary

# Mining Data Descriptive Characteristics

- Motivation
  - To better understand the data: central tendency, variation and spread
- Data dispersion characteristics
  - median, max, min, quantiles, outliers, variance, etc.
- Numerical dimensions correspond to sorted intervals
  - Data dispersion: analyzed with multiple granularities of precision
  - Boxplot or quantile analysis on sorted intervals
- Dispersion analysis on computed measures
  - Folding measures into numerical dimensions
  - Boxplot or quantile analysis on the transformed cube

# Measuring the Central Tendency

- Mean (algebraic measure) (sample vs. population):
  - Weighted arithmetic mean:
  - Trimmed mean: chopping extreme values

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\bar{x} = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}$$

- Median: A holistic measure-is measure that must be computed on the entire data set as a hole.
  - Middle value if odd number of values, or average of the middle two values otherwise
  - Estimated by interpolation (for *grouped data*):
- Mode
  - Value that occurs most frequently in the data
  - Data sets with 1,2, or3 modes are respectively called Uni-modal, bimodal, tri-modal
  - Empirical formula:

$$mean - mode = 3 \times (mean - median)$$

# Measuring the Dispersion of Data

- Quartiles, outliers and boxplots
  - Quartiles: $Q_1$ (25th percentile), $Q_3$ (75th percentile)
  - Inter-quartile range: IQR = $Q_3 - Q_1$
  - Five number summary: min, $Q_1$, M, $Q_3$, max
  - Boxplot: ends of the box are the quartiles, median is marked, whiskers, and plot outlier individually
  - Outlier: usually, a value higher/lower than 1.5 x IQR
- Variance and standard deviation (*sample: s, population: σ)*
  - Variance: (algebraic, scalable computation)

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 = \frac{1}{n-1}[\sum_{i=1}^{n} x_i^2 - \frac{1}{n}(\sum_{i=1}^{n} x_i)^2] \qquad \sigma^2 = \frac{1}{N}\sum_{i=1}^{n}(x_i - \mu)^2 = \frac{1}{N}\sum_{i=1}^{n} x_i^2 - \mu^2$$

  - Standard deviation *s (or σ)* is the square root of variance $s^2$ *(or* $\sigma^2$*)*

# Data Cleaning as a Process

- Data discrepancy detection
  - Use metadata (e.g., domain, range, dependency, distribution)
  - Check field overloading
  - Check uniqueness rule, consecutive rule and null rule
  - Use commercial tools
    - Data scrubbing: use simple domain knowledge (e.g., postal code, spell-check) to detect errors and make corrections
    - Data auditing: by analyzing data to discover rules and relationship to detect violators (e.g., correlation and clustering to find outliers)
- Data migration and integration
  - Data migration tools: allow transformations to be specified
  - ETL (Extraction/Transformation/Loading) tools: allow users to specify transformations through a graphical user interface
- Integration of the two processes
  - Iterative and interactive (e.g., Potter's Wheels)

# Data Integration

- Data integration:
  - Combines data from multiple sources into a coherent store
- Schema integration: e.g., A.cust-id ≡ B.cust-#
  - Integrate metadata from different sources
- Entity identification problem:
  - Identify real world entities from multiple data sources, e.g., Bill Clinton = William Clinton
- Detecting and resolving data value conflicts
  - For the same real world entity, attribute values from different sources are different
  - Possible reasons: different representations, different scales, e.g., metric vs. British units

# Handling Redundancy in Data Integration

- Redundant data occur often when integration of multiple databases
  - *Object identification*: The same attribute or object may have different names in different databases
  - *Derivable data:* One attribute may be a "derived" attribute in another table, e.g., annual revenue
- Redundant attributes may be able to be detected by *correlation analysis*
- Careful integration of the data from multiple sources may help reduce/avoid redundancies and inconsistencies and improve mining speed and quality

# Correlation Analysis (Numerical Data)

- Some redundancies can be detected by correlation analysis. For numerical attributes we can evaluate the correlation between two attributes by computing the correlation coefficient.
- Correlation coefficient (also called Pearson's product moment coefficient)

$$r_{A,B} = \frac{\sum (A - \overline{A})(B - \overline{B})}{(n-1)\sigma_A \sigma_B} = \frac{\sum (AB) - n\overline{A}\,\overline{B}}{(n-1)\sigma_A \sigma_B}$$

where **n** is the number of tuples, $\overline{A}$ and $\overline{B}$ are the respective means of A and B, $\boldsymbol{\sigma_A}$ and $\boldsymbol{\sigma_B}$ are the respective standard deviation of A and B, and Σ(AB) is the sum of the AB cross-product.

- If $r_{A,B} > 0$, A and B are positively correlated (A's values increase as B's). The higher, the stronger correlation.
- $r_{A,B} = 0$: independent; $r_{A,B} < 0$: negatively correlated

# Correlation Analysis (Categorical Data)

- $X^2$ (chi-square) test

$$\chi^2 = \sum \frac{(Observed - Expected)^2}{Expected}$$

- The larger the $X^2$ value, the more likely the variables are related
- The cells that contribute the most to the $X^2$ value are those whose actual count is very different from the expected count
- Correlation does not imply causality
  - # of hospitals and # of car-theft in a city are correlated
  - Both are causally linked to the third variable: population

# Chi-Square Calculation: An Example

| | Play chess | Not play chess | Sum (row) |
|---|---|---|---|
| Like science fiction | 250(90) | 200(360) | 450 |
| Not like science fiction | 50(210) | 1000(840) | 1050 |
| Sum(col.) | 300 | 1200 | 1500 |

- $X^2$ (chi-square) calculation (numbers in parenthesis are expected counts calculated based on the data distribution in the two categories)

$$\chi^2 = \frac{(250-90)^2}{90} + \frac{(50-210)^2}{210} + \frac{(200-360)^2}{360} + \frac{(1000-840)^2}{840} = 507.93$$

- It shows that like_science_fiction and play_chess are correlated in the group

# Data Transformation

- The data transformed or consolidated into forms appropriate for mining. The following are the data transformations
- Smoothing: remove noise from data
- Aggregation: summarization, data cube construction
- Generalization: concept hierarchy climbing
- Normalization: scaled to fall within a small, specified range
  - min-max normalization
  - z-score normalization
  - normalization by decimal scaling
- Attribute/feature construction
  - New attributes constructed from the given ones

# Data Transformation: Normalization

- Min-max normalization: to [new_min$_A$, new_max$_A$]

$$v' = \frac{v - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A$$

  - Ex. Let income range \$12,000 to \$98,000 normalized to [0.0, 1.0]. Then \$73,000 is mapped to $\frac{73{,}600 - 12{,}000}{98{,}000 - 12{,}000}(1.0 - 0) + 0 = 0.716$

- Z-score normalization (μ: mean, σ: standard deviation):

$$v' = \frac{v - \mu_A}{\sigma_A}$$

  - Ex. Let μ = 54,000, σ = 16,000. Then $\frac{73{,}600 - 54{,}000}{16{,}000} = 1.225$

- Normalization by decimal scaling

$$v' = \frac{v}{10^j}$$ Where $j$ is the smallest integer such that Max(|v'|) < 1

# Chapter 2: Data Preprocessing

- Why preprocess the data?
- Data cleaning
- Data integration and transformation
- Data reduction
- Discretization and concept hierarchy generation
- Summary

# Discretization

- Three types of attributes:
  - Nominal — values from an unordered set, e.g., color, profession
  - Ordinal — values from an ordered set, e.g., military or academic rank
  - Continuous — real numbers, e.g., integer or real numbers
- Discretization:
  - Divide the range of a continuous attribute into intervals
  - Some classification algorithms only accept categorical attributes.
  - Reduce data size by discretization
  - Prepare for further analysis

# Discretization and Concept Hierarchy

- Discretization
  - Reduce the number of values for a given continuous attribute by dividing the range of the attribute into intervals
  - Interval labels can then be used to replace actual data values
  - Supervised: The discretization process uses class information
  - Unsupervised: No Class information
  - Split (top-down) vs. merge (bottom-up)
  - Discretization can be performed recursively on an attribute
- Concept hierarchy formation
  - Recursively reduce the data by collecting and replacing low level concepts (such as numeric values for age) by higher level concepts (such as young, middle-aged, or senior)

# Discretization and Concept Hierarchy Generation for Numeric Data

- Typical methods: All the methods can be applied recursively
  - Binning (covered above)
    - Top-down split, unsupervised,
  - Histogram analysis (covered above)
    - Top-down split, unsupervised
  - Clustering analysis (covered above)
    - Either top-down split or bottom-up merge, unsupervised
  - Entropy-based discretization: supervised, top-down split
  - Interval merging by $\chi^2$ Analysis: unsupervised, bottom-up merge
  - Segmentation by natural partitioning: top-down split, unsupervised

# Discretization and Concept Hierarchy Generation for Numeric Data

- Entropy-based discretization: supervised, top-down split
- Let *D* consist of data tuples defined by a set of attributes and a class-label attribute.
- The class-label attribute provides the class information per tuple.
- The basic method for entropy-based discretization of an attribute *A* within the set is as follows:
  - 1. Each value of *A* can be considered as a potential interval boundary or split-point(denoted *split point*) to partition the range of *A*.
  - Split-point for *A* can partition the tuples in *D* into two subsets satisfying the conditions *A* *split point* and*A* > *split point*, respectively, thereby creating a binary discretization.

# Discretization and Concept Hierarchy Generation for Numeric Data

- To explain the intuition behind entropy-based discretization, we must take a glimpse at classification.
- Suppose we want to classify the tuples in *D* by partitioning on attribute *A* and some split-point.
- Ideally, we would like this partitioning to result in an exact classification of the tuples.

# Discretization and Concept Hierarchy Generation for Numeric Data

- 3. The process of determining a split-point is recursively applied to each partition obtained, until some stopping criterion is met,
  - the minimum information requirement on all candidate split-points is less than a small threshold
  - the number of intervals is greater than a threshold, *max interval*

# Discretization and Concept Hierarchy Generation for Numeric Data

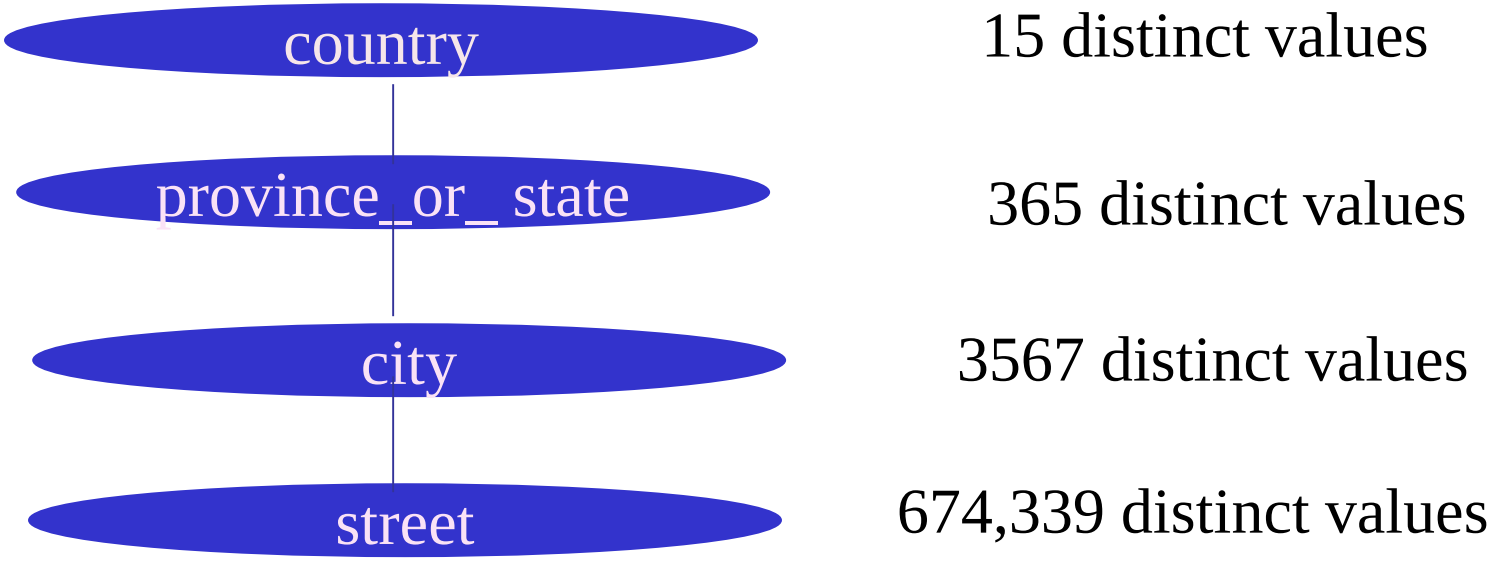
- Interval merging by $\chi^2$ Analysis: unsupervised, bottom-up merge
  - Initially, each distinct value of a numerical attribute A is considered to be one interval.
  - $\chi^2$ tests are performed for every pair of adjacent intervals.
  - Adjacent intervals with the least $\chi^2$ values are merged together, because low $\chi^2$ values for a pair indicate similar class distributions.
  - This merging process proceeds recursively until a predefined stopping criterion is met.

# Concept Hierarchy Generation for Categorical Data

- Specification of a partial/total ordering of attributes explicitly at the schema level by users or experts
  - street < city < state < country
- Specification of a hierarchy for a set of values by explicit data grouping
  - {Urbana, Champaign, Chicago} < Illinois
- Specification of only a partial set of attributes
  - E.g., only street < city, not others
- Automatic generation of hierarchies (or attribute levels) by the analysis of the number of distinct values
  - E.g., for a set of attributes: {street, city, state, country}

# Automatic Concept Hierarchy Generation

- Some hierarchies can be automatically generated based on the analysis of the number of distinct values per attribute in the data set
  - The attribute with the most distinct values is placed at the lowest level of the hierarchy
  - Exceptions, e.g., weekday, month, quarter, year



# Chapter 2: Data Preprocessing

- Why preprocess the data?
- Data cleaning
- Data integration and transformation
- Data reduction
- Discretization and concept hierarchy generation
- Summary

# Summary

- Data  preparation or preprocessing is a big issue for both data warehousing and data mining
- Discriptive data summarization is need for quality data preprocessing
- Data preparation includes
  - Data cleaning and data integration
  - Data reduction and feature selection
  - Discretization
- A lot a methods have been developed but data preprocessing still an active area of research

# References


- D. P. Ballou and G. K. Tayi. Enhancing data quality in data warehouse environments. Communications of ACM, 42:73-78, 1999
- T. Dasu and T. Johnson. Exploratory Data Mining and Data Cleaning. John Wiley & Sons, 2003
- T. Dasu, T. Johnson, S. Muthukrishnan, V. Shkapenyuk. Mining Database Structure; Or, How to Build a Data Quality Browser. SIGMOD'02.
- H.V. Jagadish et al., Special Issue on Data Reduction Techniques. Bulletin of the Technical Committee on Data Engineering, 20(4), December 1997
- D. Pyle. Data Preparation for Data Mining. Morgan Kaufmann, 1999
- E. Rahm and H. H. Do. Data Cleaning: Problems and Current Approaches. *IEEE Bulletin of the Technical Committee on Data Engineering. Vol.23, No.4*
- V. Raman and J. Hellerstein. Potters Wheel: An Interactive Framework for Data Cleaning and Transformation, VLDB'2001
- T. Redman. Data Quality: Management and Technology. Bantam Books, 1992
- Y. Wand and R. Wang. Anchoring data quality dimensions ontological foundations. Communications of ACM, 39:86-95, 1996
- R. Wang, V. Storey, and C. Firth. A framework for analysis of data quality research. IEEE Trans. Knowledge and Data Engineering, 7:623-640, 1995

# Chapter 3: Data Warehousing and OLAP Technology: An Overview

- What is a data warehouse?
- A multi-dimensional data model
- Data warehouse architecture
- Data warehouse implementation
- From data warehousing to data mining

# What is Data Warehouse?

- Defined in many different ways, but not rigorously.
  - A decision support database that is maintained separately from the organization's operational database
  - Support information processing by providing a solid platform of consolidated, historical data for analysis.
- "A data warehouse is a subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management's decision-making process."—W. H. Inmon
- Data warehousing:
  - The process of constructing and using data warehouses

# Data Warehouse—Subject-Oriented

- Organized around major subjects, such as customer, product, sales
- Focusing on the modeling and analysis of data for decision makers, not on daily operations or transaction processing
- Provide a simple and concise view around particular subject issues by excluding data that are not useful in the decision support process

# Data Warehouse—Integrated

- Constructed by integrating multiple, heterogeneous data sources
  - relational databases, flat files, on-line transaction records
- Data cleaning and data integration techniques are applied.
  - Ensure consistency in naming conventions, encoding structures, attribute measures, etc. among different data sources
    - E.g., Hotel price: currency, tax, breakfast covered, etc.
  - When data is moved to the warehouse, it is converted.

# Data Warehouse—Time Variant

- The time horizon for the data warehouse is significantly longer than that of operational systems
  - Operational database: current value data
  - Data warehouse data: provide information from a historical perspective (e.g., past 5-10 years)
- Every key structure in the data warehouse
  - Contains an element of time, explicitly or implicitly
  - But the key of operational data may or may not contain "time element"

# Data Warehouse—Nonvolatile

- A physically separate store of data transformed from the operational environment
- Operational update of data does not occur in the data warehouse environment
  - Does not require transaction processing, recovery, and concurrency control mechanisms
  - Requires only two operations in data accessing:
    - *initial loading of data* and *access of data*

# Data Warehouse vs. Heterogeneous DBMS

- Traditional heterogeneous DB integration: A query driven approach
  - Build wrappers/mediators on top of heterogeneous databases
  - When a query is posed to a client site, a meta-dictionary is used to translate the query into queries appropriate for individual heterogeneous sites involved, and the results are integrated into a global answer set
  - Complex information filtering, compete for resources
- Data warehouse: update-driven, high performance
  - Information from heterogeneous sources is integrated in advance and stored in warehouses for direct query and analysis

# Data Warehouse vs. Operational DBMS

- OLTP (on-line transaction processing)
  - Major task of traditional relational DBMS
  - Day-to-day operations: purchasing, inventory, banking, manufacturing, payroll, registration, accounting, etc.
- OLAP (on-line analytical processing)
  - Major task of data warehouse system
  - Data analysis and decision making
- Distinct features (OLTP vs. OLAP):
  - User and system orientation: customer vs. market
  - Data contents: current, detailed vs. historical, consolidated
  - Database design: ER + application vs. star + subject
  - View: current, local vs. evolutionary, integrated
  - Access patterns: update vs. read-only but complex queries

# OLTP vs. OLAP

| | OLTP | OLAP |
|---|---|---|
| users | clerk, IT professional | knowledge worker |
| function | day to day operations | decision support |
| DB design | application-oriented | subject-oriented |
| data | current, up-to-date<br>detailed, flat relational<br>isolated | historical,<br>summarized, multidimensional<br>integrated, consolidated |
| usage | repetitive | ad-hoc |
| access | read/write<br>index/hash on prim. key | lots of scans |
| unit of work | short, simple transaction | complex query |
| # records accessed | tens | millions |
| #users | thousands | hundreds |
| DB size | 100MB-GB | 100GB-TB |
| metric | transaction throughput | query throughput, response |

# Why Separate Data Warehouse?

- High performance for both systems
  - DBMS— tuned for OLTP: access methods, indexing, concurrency control, recovery
  - Warehouse—tuned for OLAP: complex OLAP queries, multidimensional view, consolidation
- Different functions and different data:
  - missing data: Decision support requires historical data which operational DBs do not typically maintain
  - data consolidation: DS requires consolidation (aggregation, summarization) of data from heterogeneous sources
  - data quality: different sources typically use inconsistent data representations, codes and formats which have to be reconciled
- Note: There are more and more systems which perform OLAP analysis directly on relational databases

# Chapter 3: Data Warehousing and OLAP Technology: An Overview

- What is a data warehouse?
- A multi-dimensional data model
- Data warehouse architecture
- Data warehouse implementation
- From data warehousing to data mining

# From Tables and Spreadsheets to Data Cubes

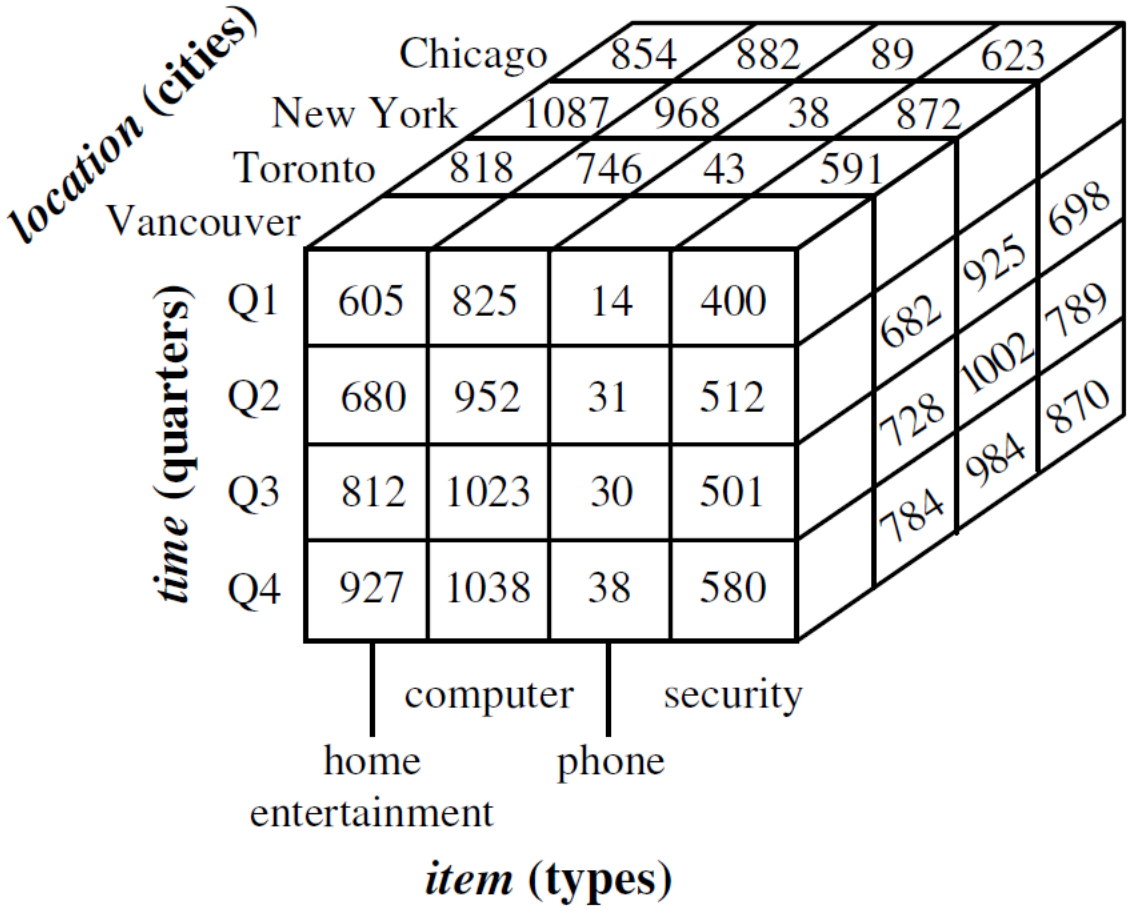
- A data warehouse is based on a multidimensional data model which views data in the form of a data cube
- A data cube, such as sales, allows data to be modeled and viewed in multiple dimensions
  - Dimension tables, such as item (item_name, brand, type), or time(day, week, month, quarter, year)
  - Fact table contains measures (such as dollars_sold) and keys to each of the related dimension tables
- In data warehousing literature, an n-D base cube is called a base cuboid. The top most 0-D cuboid, which holds the highest-level of summarization, is called the apex cuboid. The lattice of cuboids forms a data cube.

# Data Cubes

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 |

# Data Cubes

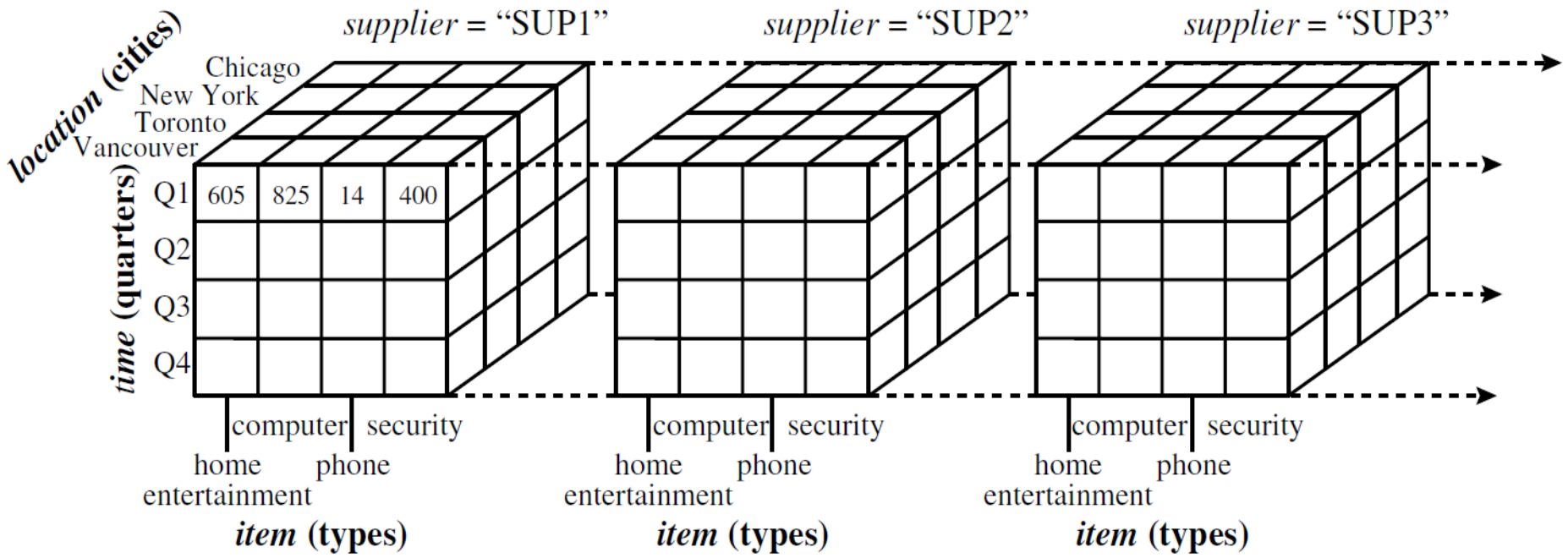

location (cities)
Chicago
New York
Toronto
Vancouver
854
882
89
623
1087
968
38
872
818
746
43
591
time (quarters)
Q1
Q2
Q3
Q4
605
825
14
400
680
952
31
512
812
1023
30
501
927
1038
38
580
682
925
698
728
1002
789
784
984
870
computer
security
home
entertainment
phone
item (types)

# Data Cubes

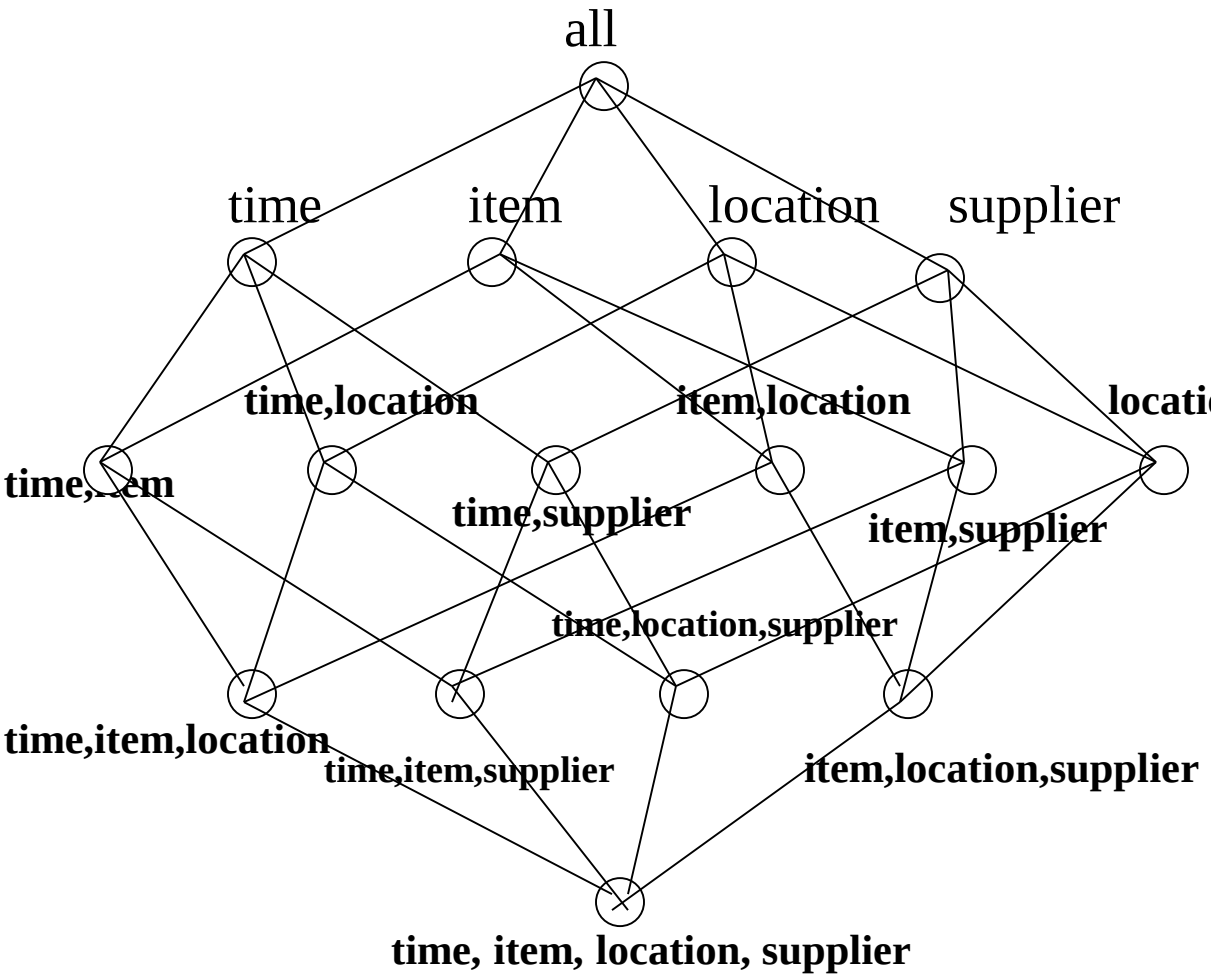
**Table 3.3** A 3-D view of sales data for *AllElectronics*, according to the dimensions *time*, *item*, and *location*. The measure displayed is *dollars_sold* (in thousands).

| | ***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 |

# Data Cubes


supplier = "SUP1"
supplier = "SUP2"
supplier = "SUP3"
location (cities)
Chicago
New York
Toronto
Vancouver
time (quarters)
Q1
Q2
Q3
Q4
605
825
14
400
computer
security
home entertainment
phone
item (types)

# Cube: A Lattice of Cuboids




0-D(apex) cuboid

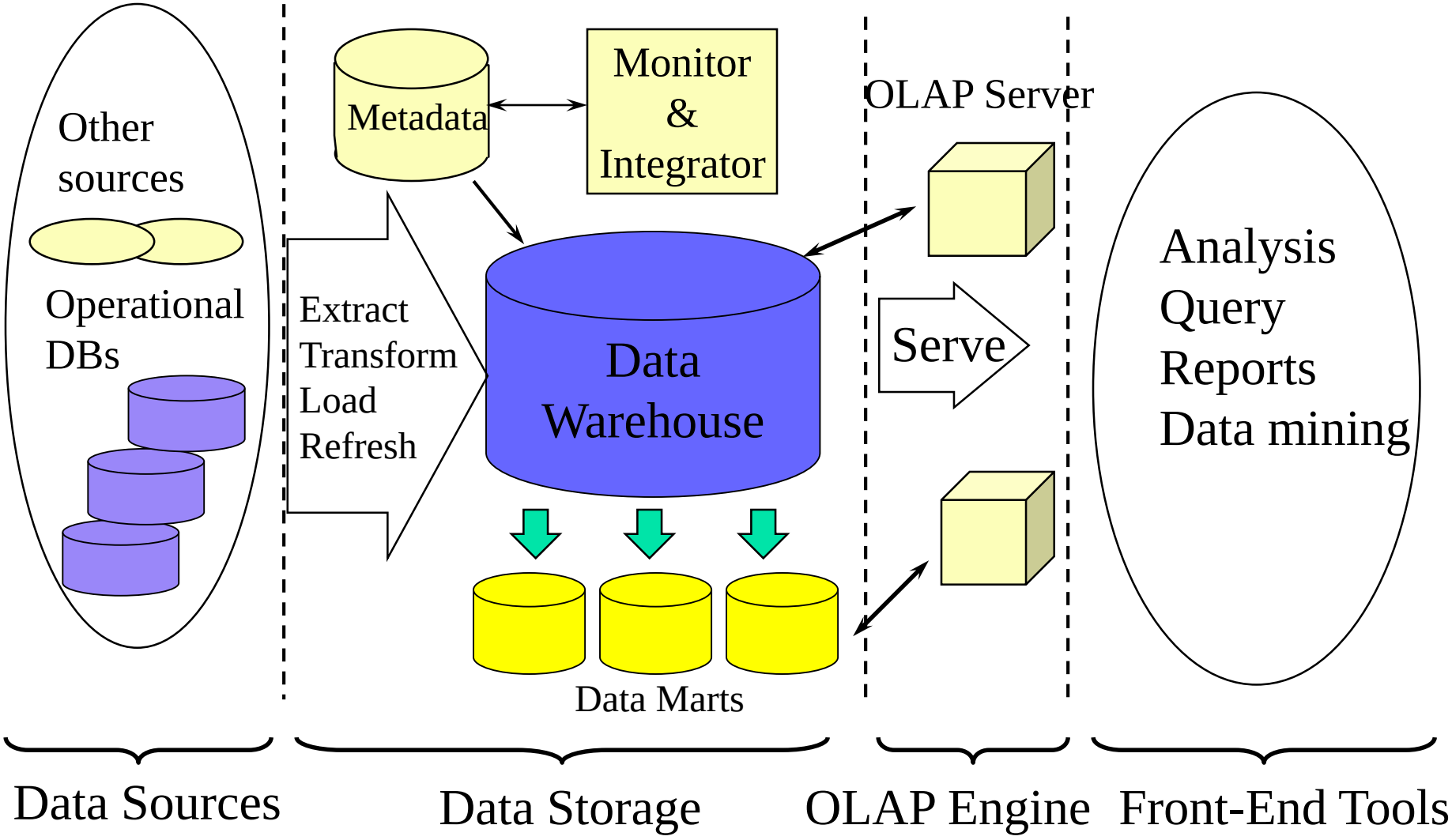
1-D cuboids

2-D cuboids

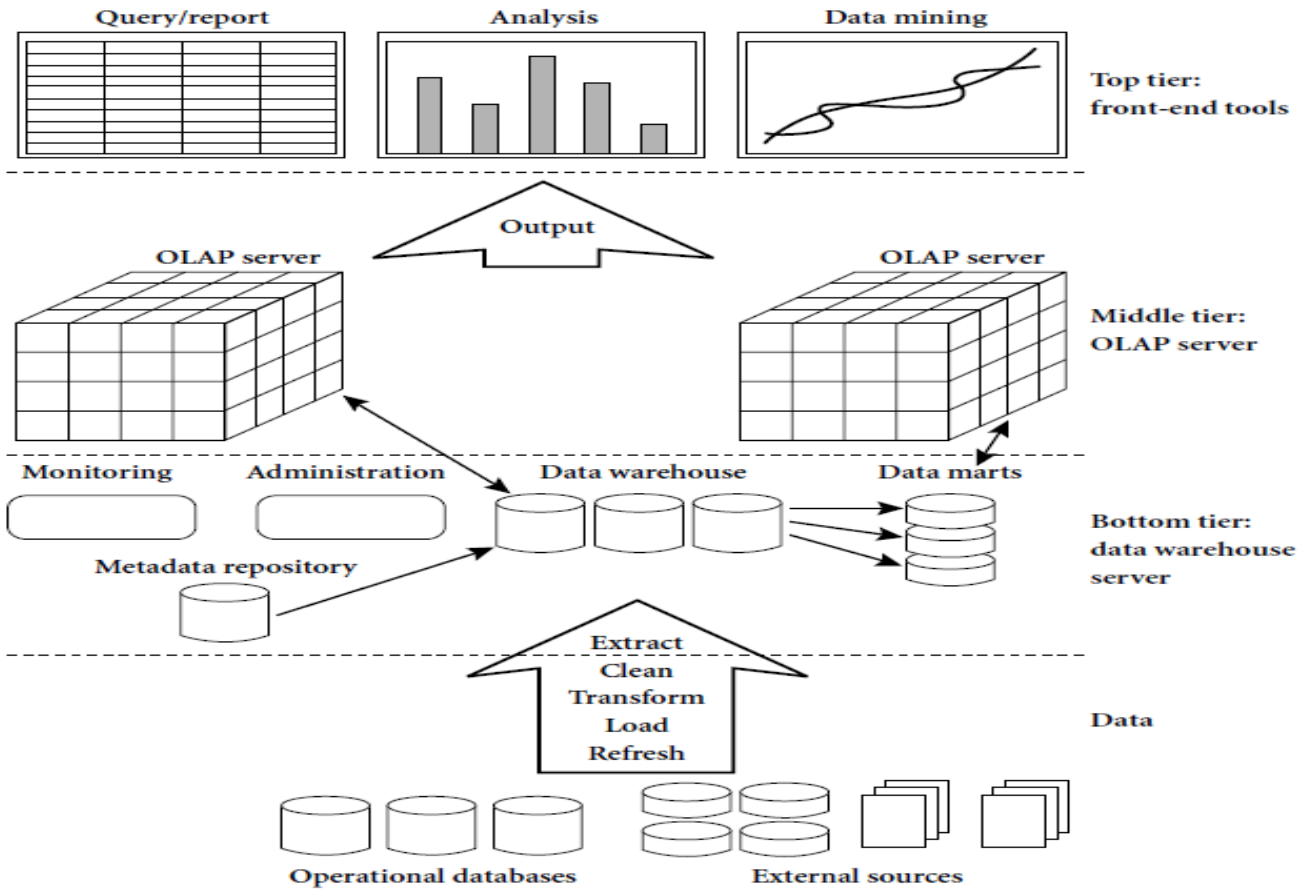
3-D cuboids

4-D(base) cuboid

# Data Warehouse: A Multi-Tiered Architecture



# Data Warehouse: Three-Tier Architecture

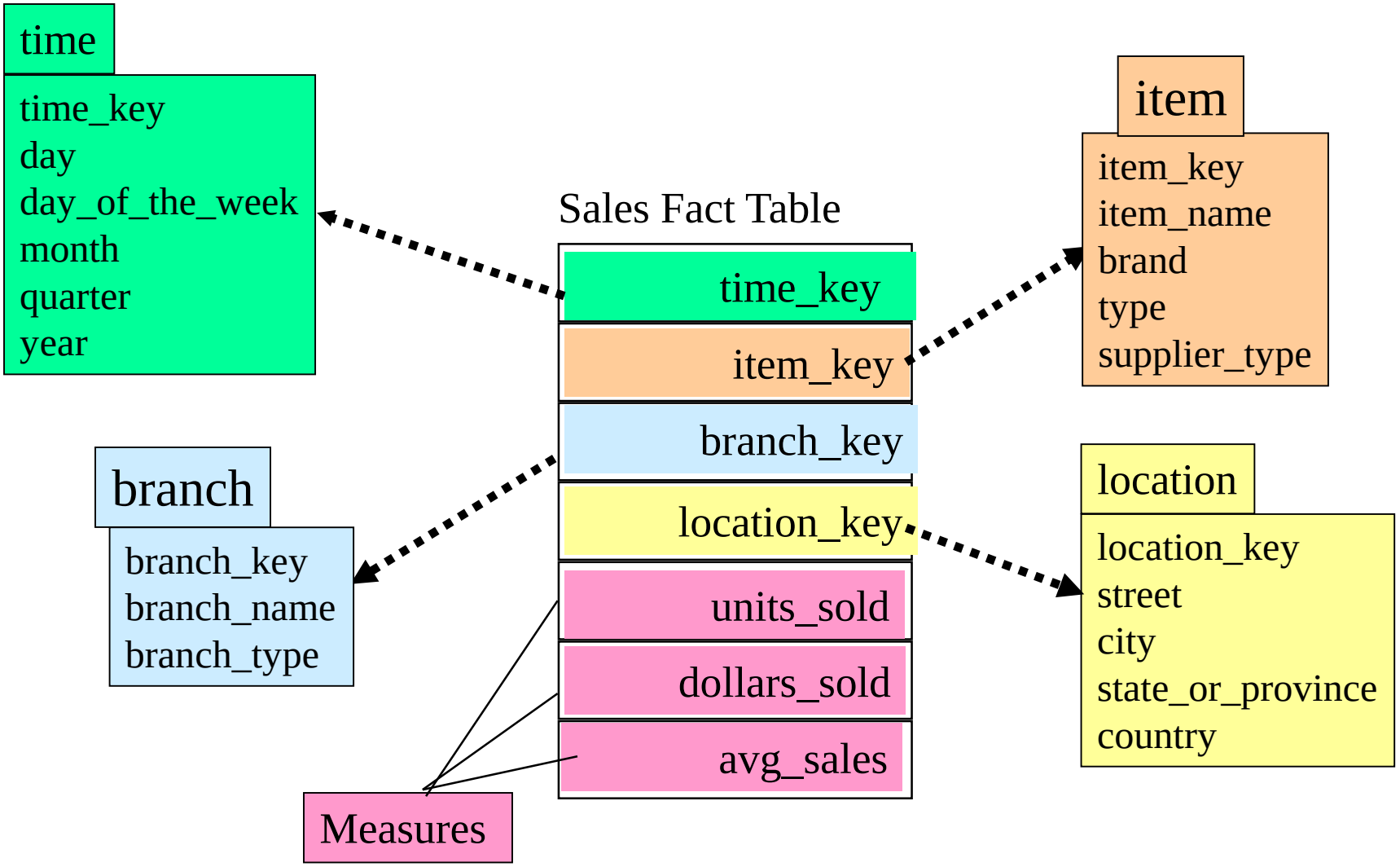



A three-tier data warehousing architecture.

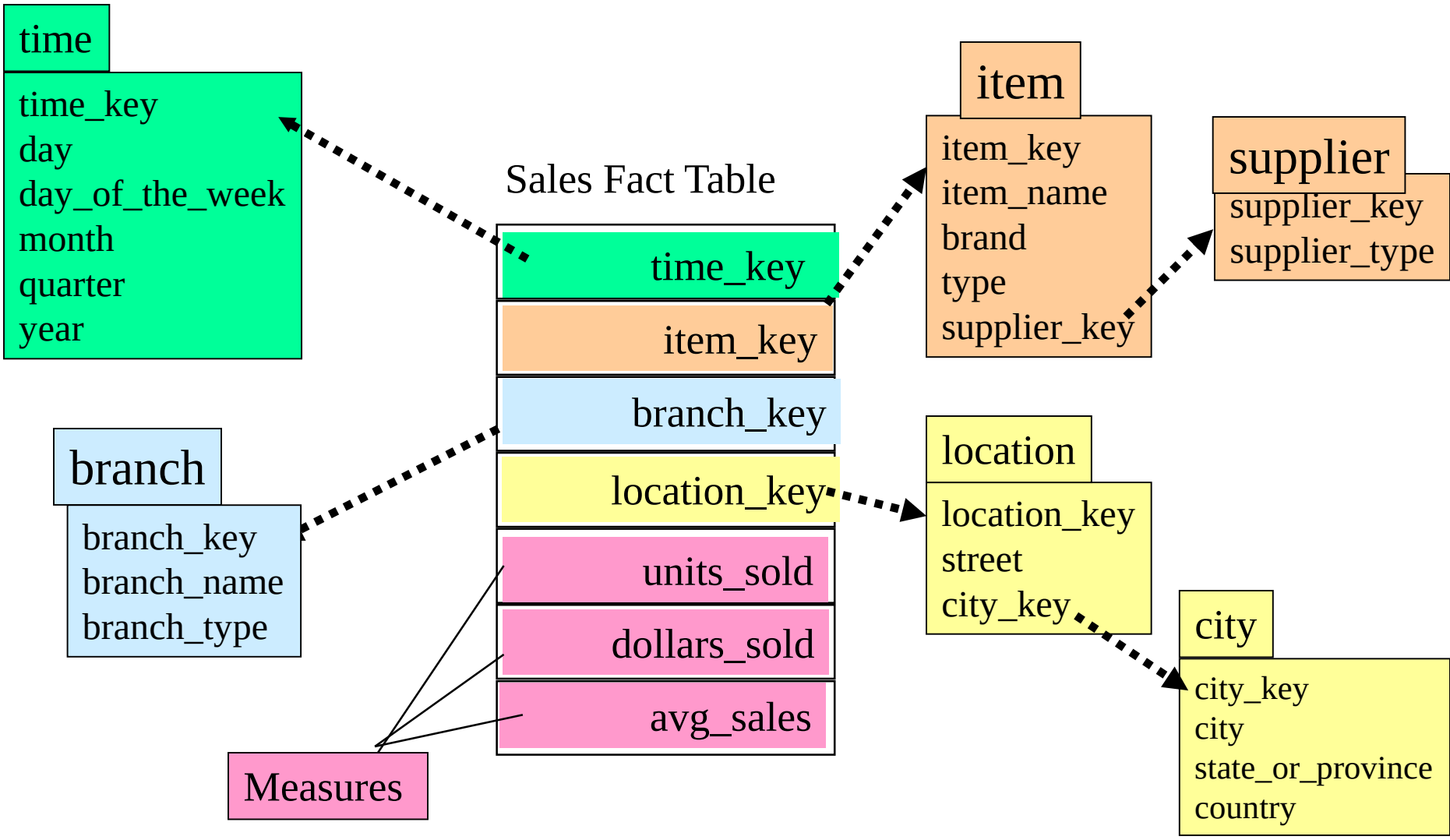
# Conceptual Modeling of Data Warehouses

- Modeling data warehouses: dimensions & measures
  - Star schema: A fact table in the middle connected to a set of dimension tables
  - Snowflake schema: A refinement of star schema where some dimensional hierarchy is normalized into a set of smaller dimension tables, forming a shape similar to snowflake
  - Fact constellations: Multiple fact tables share dimension tables, viewed as a collection of stars, therefore called galaxy schema or fact constellation

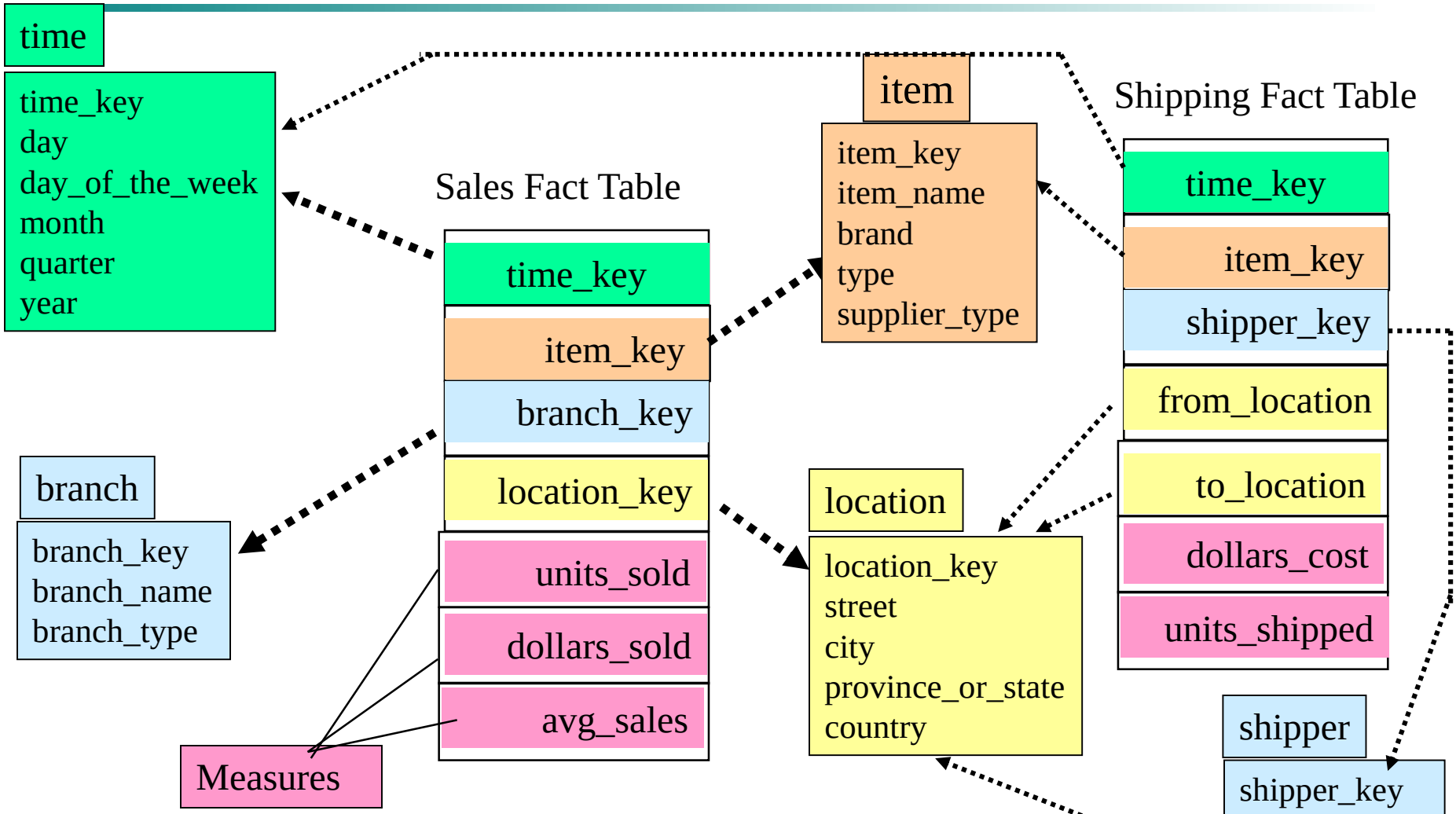
# Example of Star Schema



# Example of Snowflake Schema



# Example of Fact Constellation



# Cube Definition Syntax (BNF) in DMQL

- Cube Definition (Fact Table)

  define cube <cube_name> [<dimension_list>]:
  <measure_list>

- Dimension Definition (Dimension Table)

  define dimension <dimension_name> as
  (<attribute_or_subdimension_list>)

- Special Case (Shared Dimension Tables)
  - First time as "cube definition"
  - define dimension <dimension_name> as
    <dimension_name_first_time> in cube
    <cube_name_first_time>

# Defining Star Schema in DMQL

define cube sales_star [time, item, branch, location]:
dollars_sold = sum(sales_in_dollars), avg_sales = avg(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)

# Defining Snowflake Schema in DMQL

define cube sales_snowflake [time, item, branch, location]:
dollars_sold = sum(sales_in_dollars), avg_sales = avg(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, province_or_state, country))

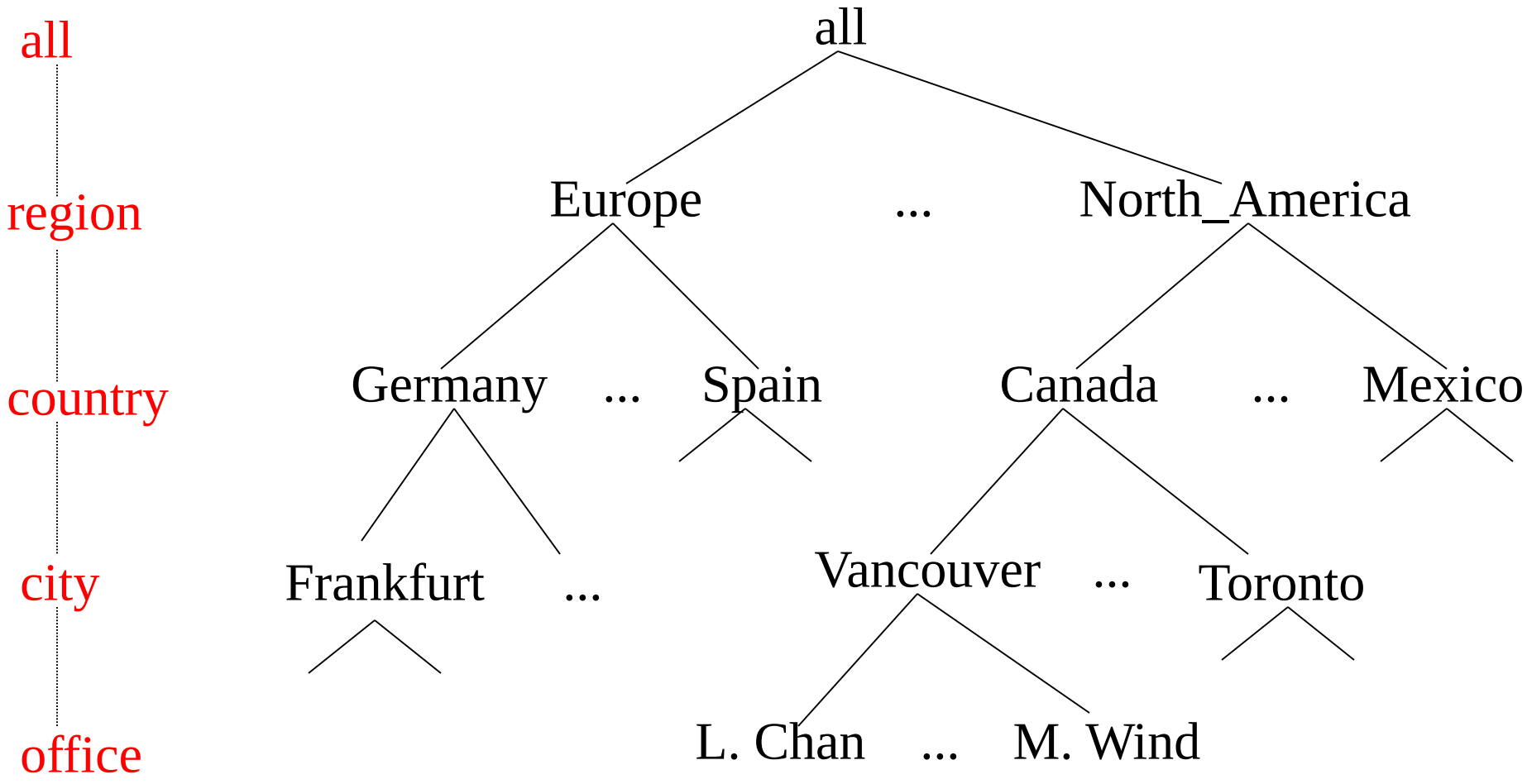
# Defining Fact Constellation in DMQL

```
define cube sales [time, item, branch, location]:
        dollars_sold = sum(sales_in_dollars), avg_sales =
          avg(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]:
        dollar_cost = sum(cost_in_dollars), unit_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
```

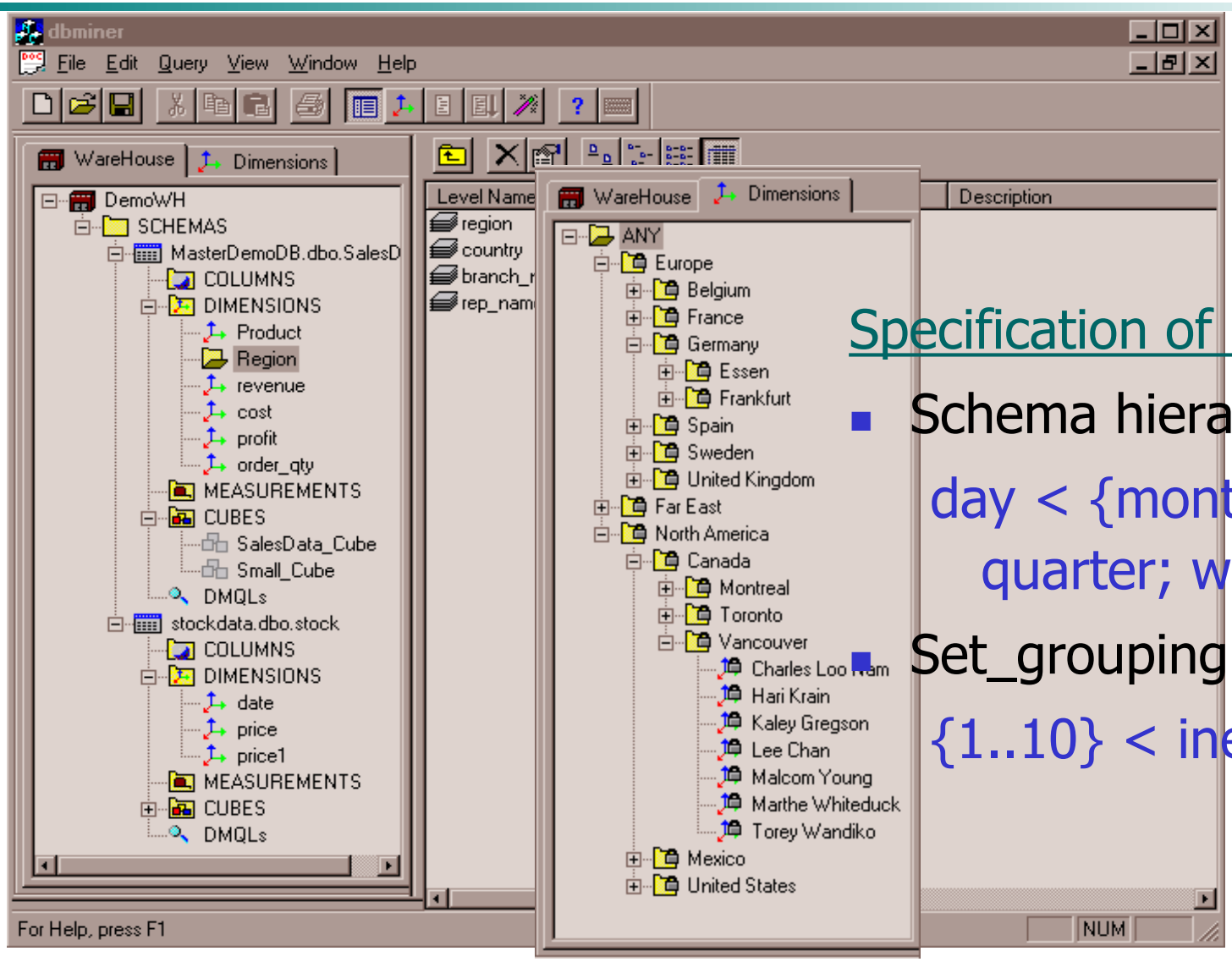
# Measures of Data Cube: Three Categories

- Distributive: if the result derived by applying the function to *n* aggregate values is the same as that derived by applying the function on all the data without partitioning
  - E.g., count(), sum(), min(), max()
- Algebraic: if it can be computed by an algebraic function with *M* arguments (where *M* is a bounded integer), each of which is obtained by applying a distributive aggregate function
  - E.g., avg(), min_N(), standard_deviation()
- Holistic: if there is no constant bound on the storage size needed to describe a subaggregate.
  - E.g., median(), mode(), rank()

# A Concept Hierarchy: Dimension (location)



# View of Warehouses and Hierarchies




## Specification of hierarchies

- Schema hierarchy

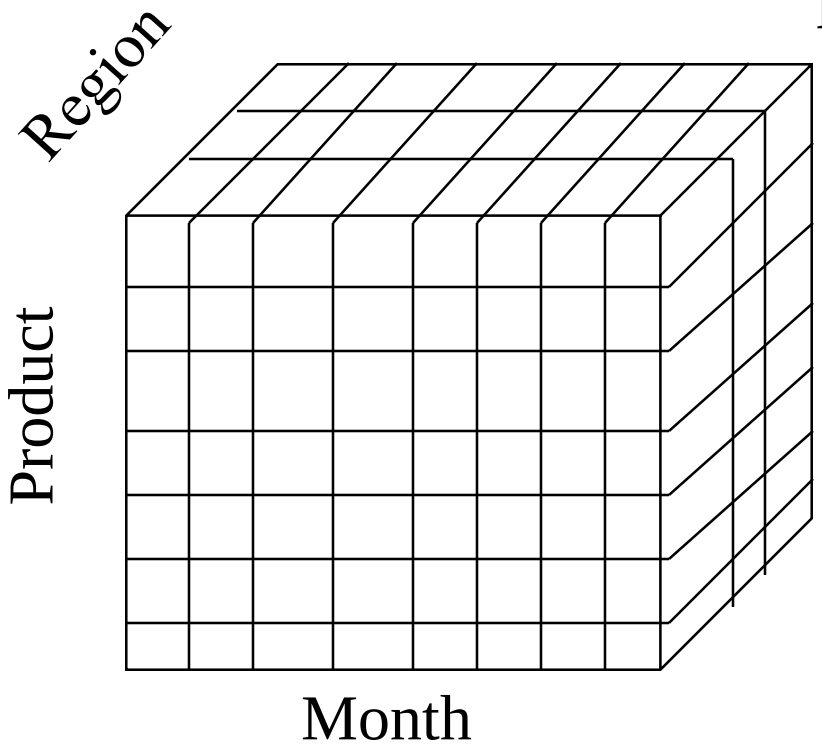
  day < {month < quarter; week} < year

- Set_grouping hierarchy

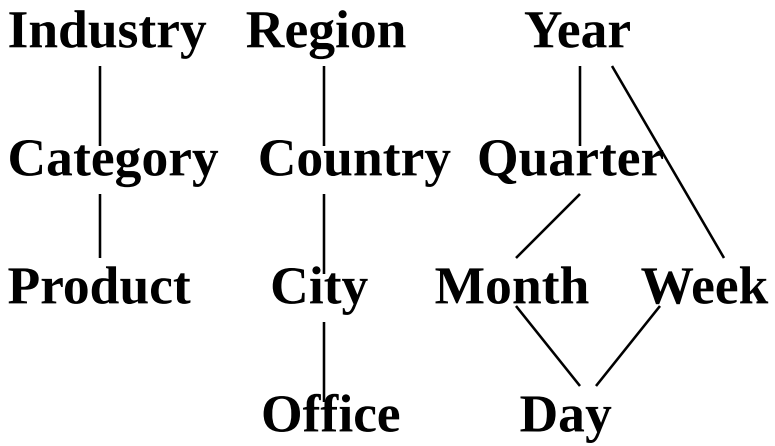
  {1..10} < inexpensive

# Multidimensional Data

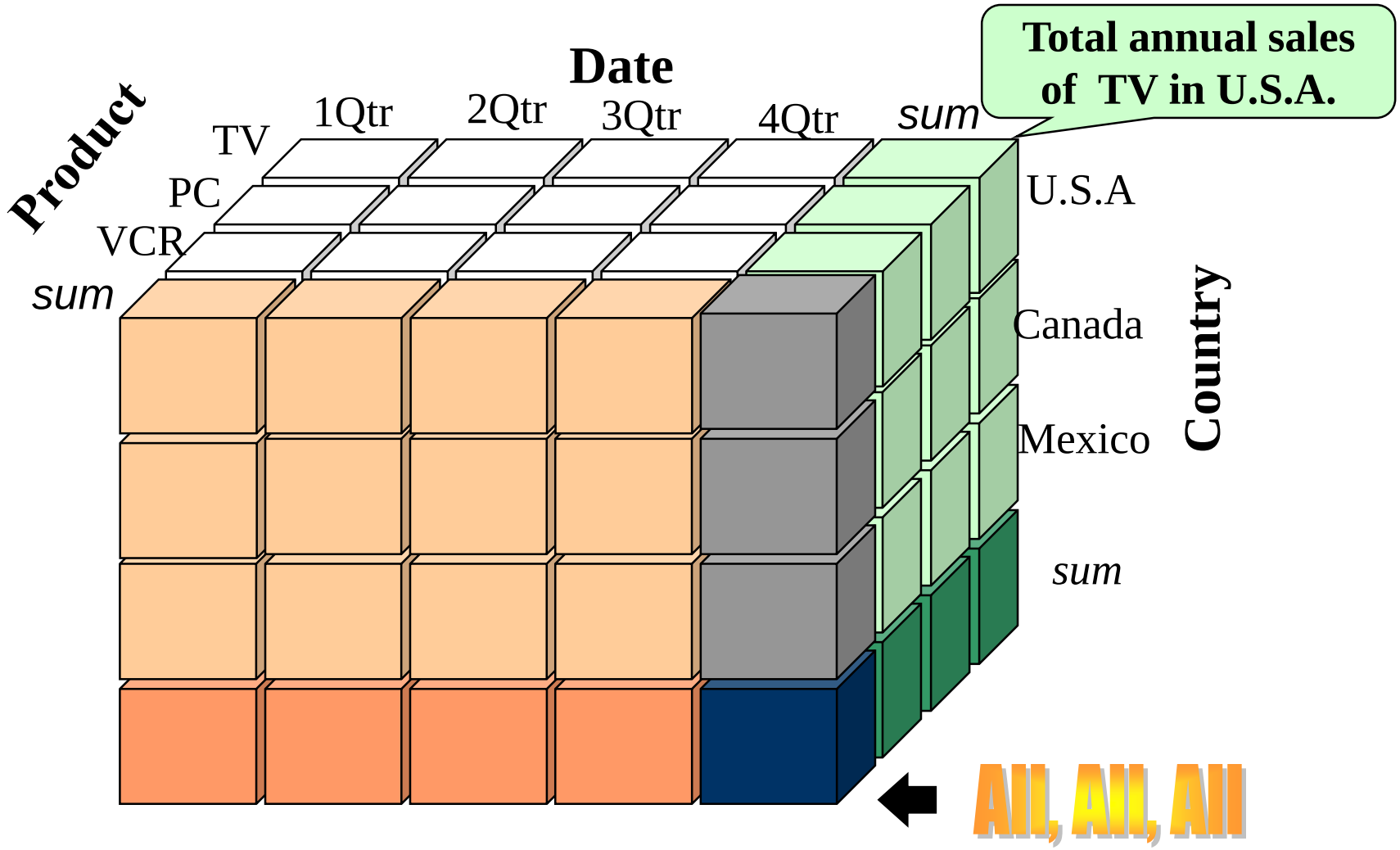
- Sales volume as a function of product, month, and region



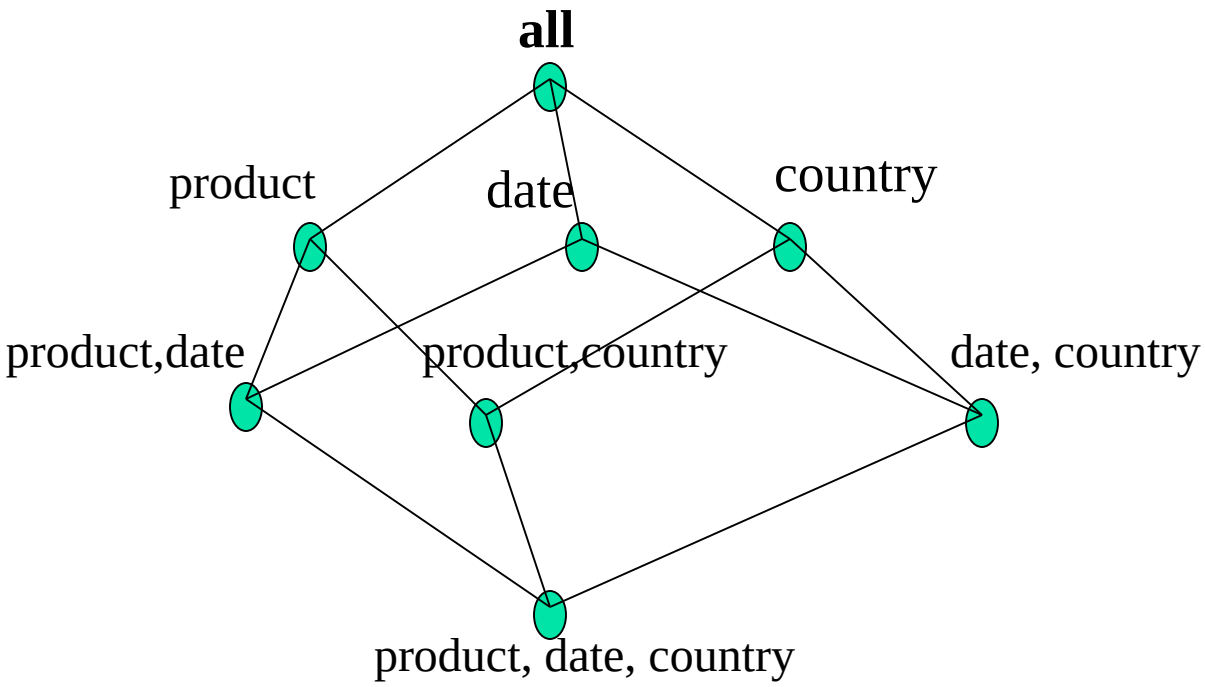

**Dimensions: Product, Location, Time**
**Hierarchical summarization paths**



# A Sample Data Cube



# Cuboids Corresponding to the Cube



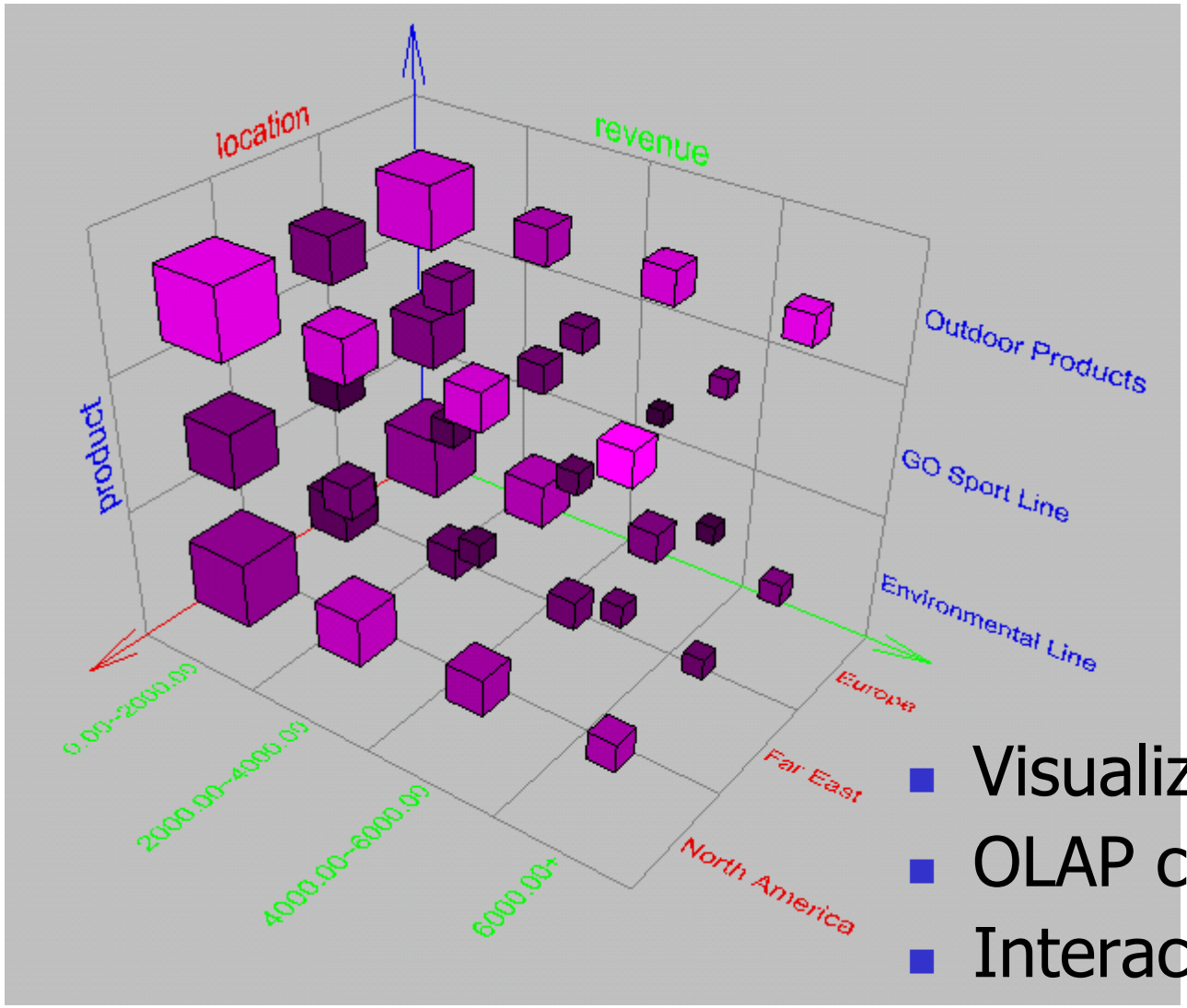

0-D(apex) cuboid

1-D cuboids

2-D cuboids

3-D(base) cuboid

# Browsing a Data Cube




- Visualization
- OLAP capabilities
- Interactive manipulation

# Typical OLAP Operations

- Roll up (drill-up): summarize data
  - *by climbing up hierarchy or by dimension reduction*
- Drill down (roll down): reverse of roll-up
  - *from higher level summary to lower level summary or detailed data, or introducing new dimensions*
- Slice and dice: *project and select*
- Pivot (rotate):
  - *reorient the cube, visualization, 3D to series of 2D planes*
- Other operations
  - *drill across: involving (across) more than one fact table*
  - *drill through: through the bottom level of the cube to its back-end relational tables (using SQL)*

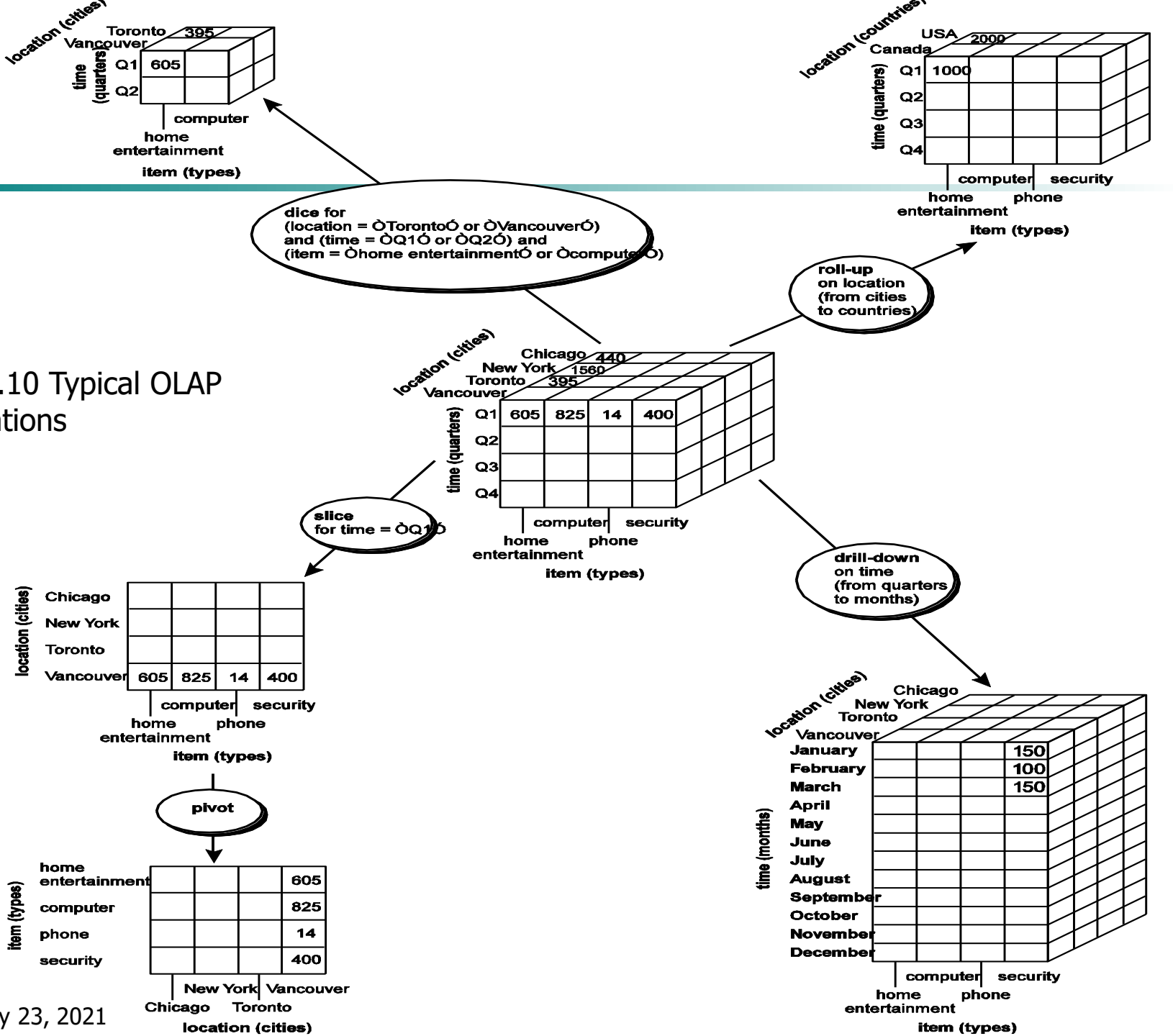
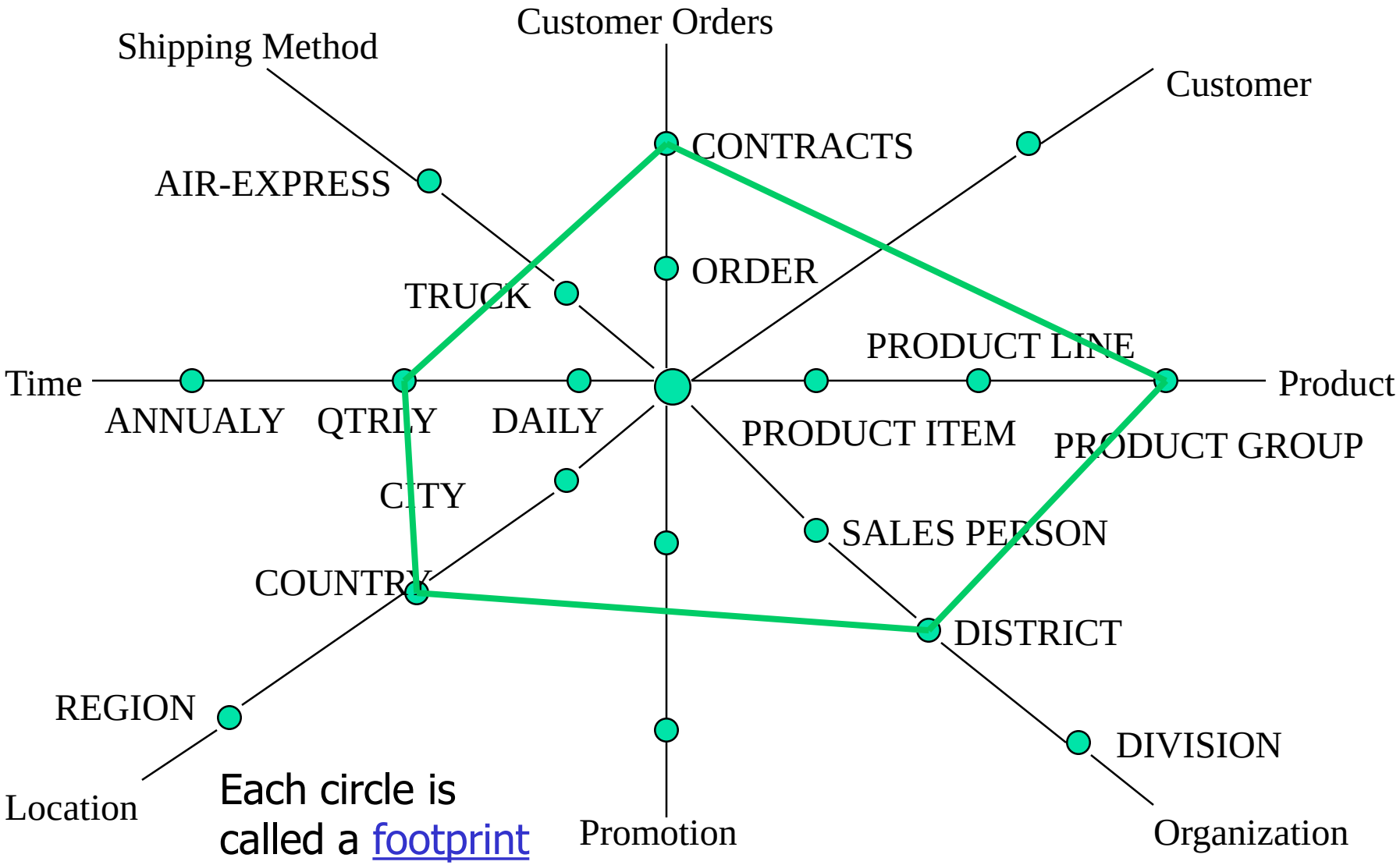



Fig. 3.10 Typical OLAP Operations

# A Star-Net Query Model




Each circle is called a footprint

# Chapter 3: Data Warehousing and OLAP Technology: An Overview

- What is a data warehouse?
- A multi-dimensional data model
- Data warehouse architecture
- Data warehouse implementation
- From data warehousing to data mining

# Design of Data Warehouse: A Business Analysis Framework

- Four views regarding the design of a data warehouse
  - Top-down view
    - allows selection of the relevant information necessary for the data warehouse
  - Data source view
    - exposes the information being captured, stored, and managed by operational systems
  - Data warehouse view
    - consists of fact tables and dimension tables
  - Business query view
    - sees the perspectives of data in the warehouse from the view of end-user

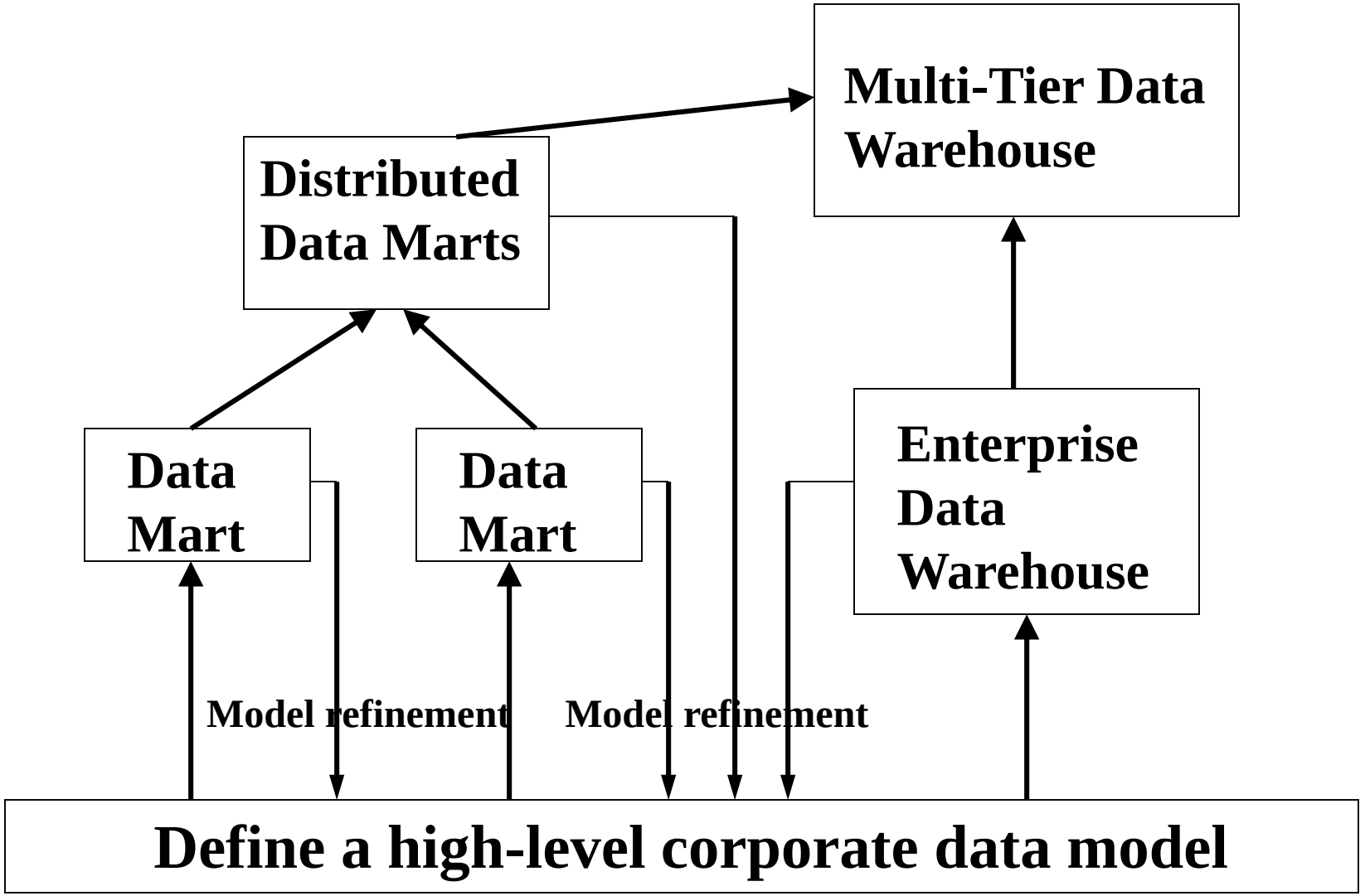
# Data Warehouse Design Process

- Top-down, bottom-up approaches or a combination of both
  - Top-down: Starts with overall design and planning (mature)
  - Bottom-up: Starts with experiments and prototypes (rapid)
- From software engineering point of view
  - Waterfall: structured and systematic analysis at each step before proceeding to the next
  - Spiral: rapid generation of increasingly functional systems, short turn around time, quick turn around
- Typical data warehouse design process
  - Choose a business process to model, e.g., orders, invoices, etc.
  - Choose the *grain* (*atomic level of data*) of the business process
  - Choose the dimensions that will apply to each fact table record
  - Choose the measure that will populate each fact table record

# Three Data Warehouse Models

- Enterprise warehouse
  - collects all of the information about subjects spanning the entire organization
- Data Mart
  - a subset of corporate-wide data that is of value to a specific groups of users. Its scope is confined to specific, selected groups, such as marketing data mart
    - Independent vs. dependent (directly from warehouse) data mart
- Virtual warehouse
  - A set of views over operational databases
  - Only some of the possible summary views may be materialized

# Data Warehouse Development: A Recommended Approach



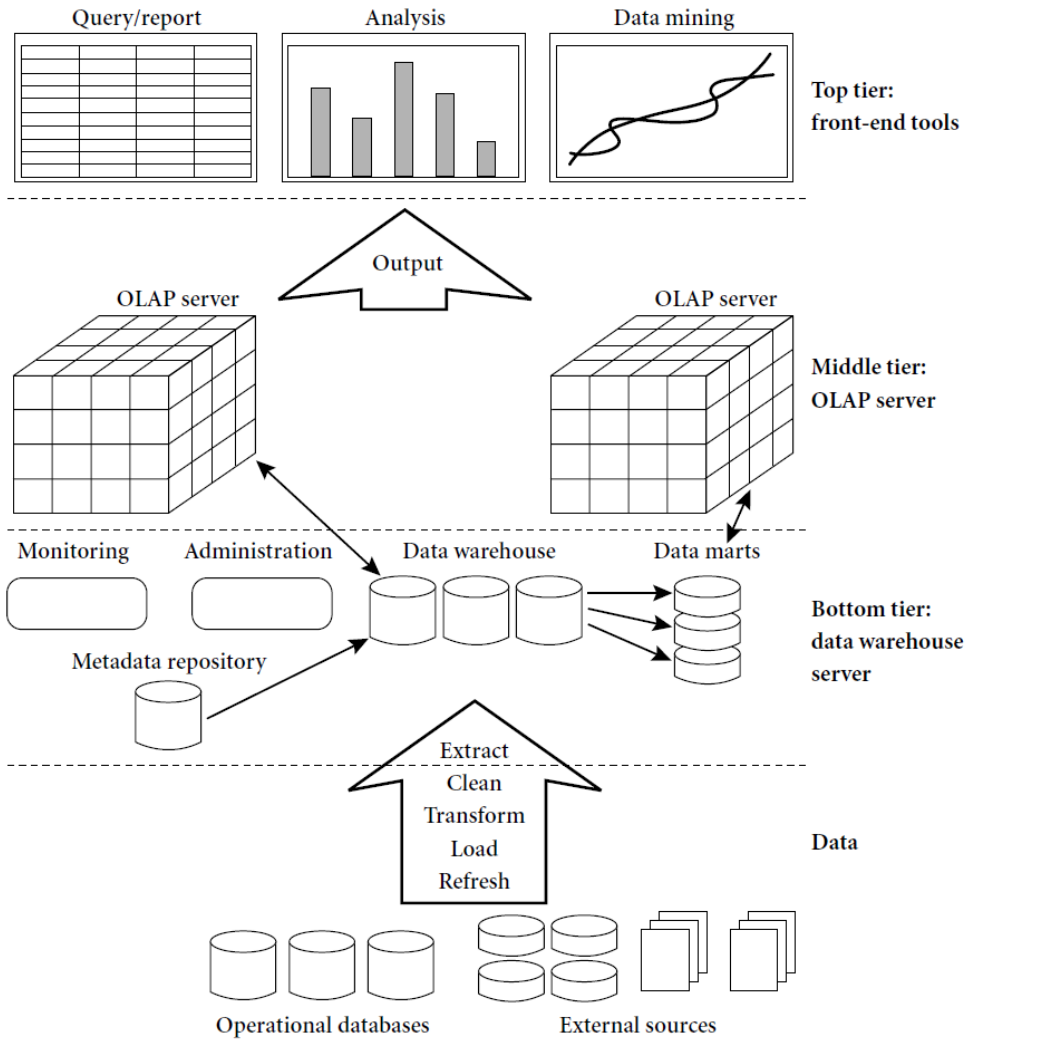
# Three-tier data warehouse architecture

- 1. Bottom tier is a warehouse database server
- 2. Middle tier is an OLAP server
- 3. Top tier is a front-end client layer

# Data Warehouse Back-End Tools and Utilities

- Data extraction
  - get data from multiple, heterogeneous, and external sources
- Data cleaning
  - detect errors in the data and rectify them when possible
- Data transformation
  - convert data from legacy or host format to warehouse format
- Load
  - sort, summarize, consolidate, compute views, check integrity, and build indicies and partitions
- Refresh
  - propagate the updates from the data sources to the warehouse

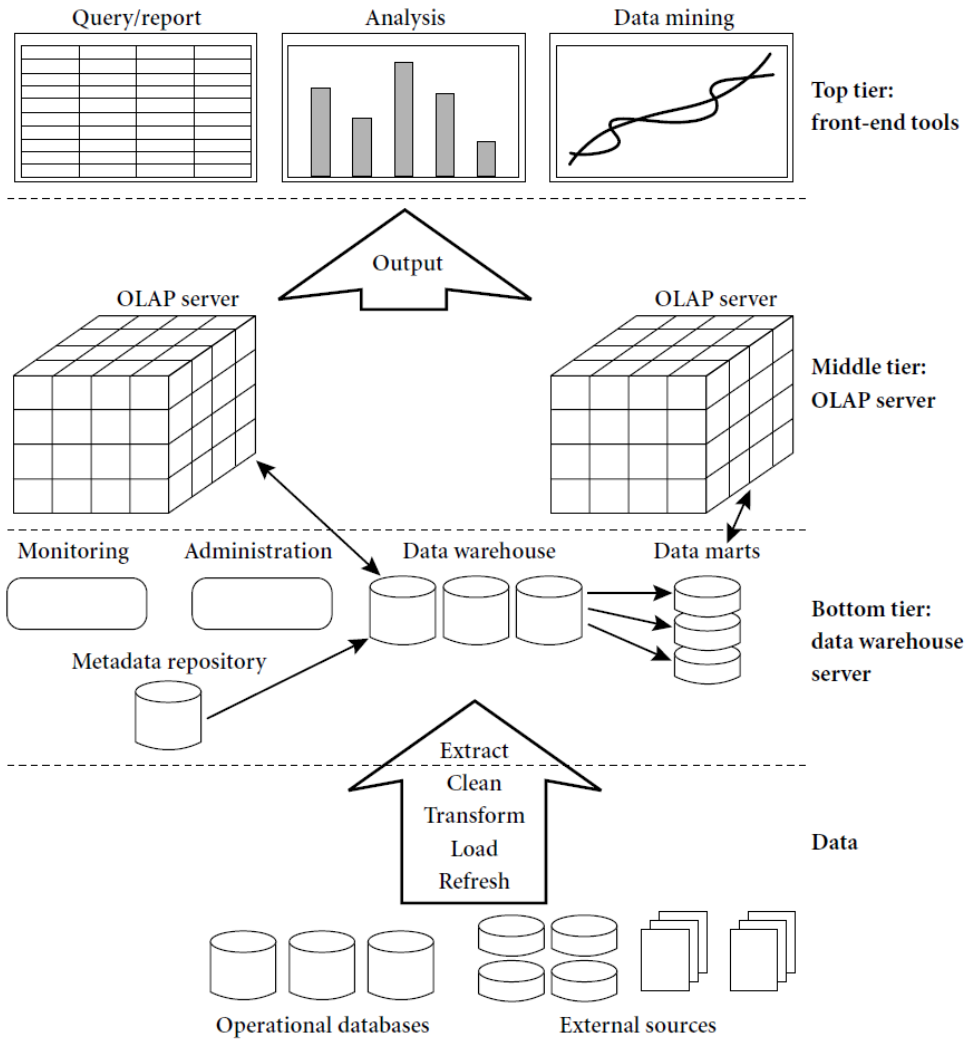
# Three-tier data warehouse architecture



# Metadata Repository

- Meta data is the data defining warehouse objects.  It stores:
- Description of the structure of the data warehouse
  - schema, view, dimensions, hierarchies, derived data defn, data mart locations and contents
- Operational meta-data
  - data lineage (history of migrated data and transformation path), currency of data (active, archived, or purged), monitoring information (warehouse usage statistics, error reports, audit trails)
- The algorithms used for summarization
- The mapping from operational environment to the data warehouse
- Data related to system performance
  - warehouse schema, view and derived data definitions
- Business data
  - business terms and definitions, ownership of data, charging policies

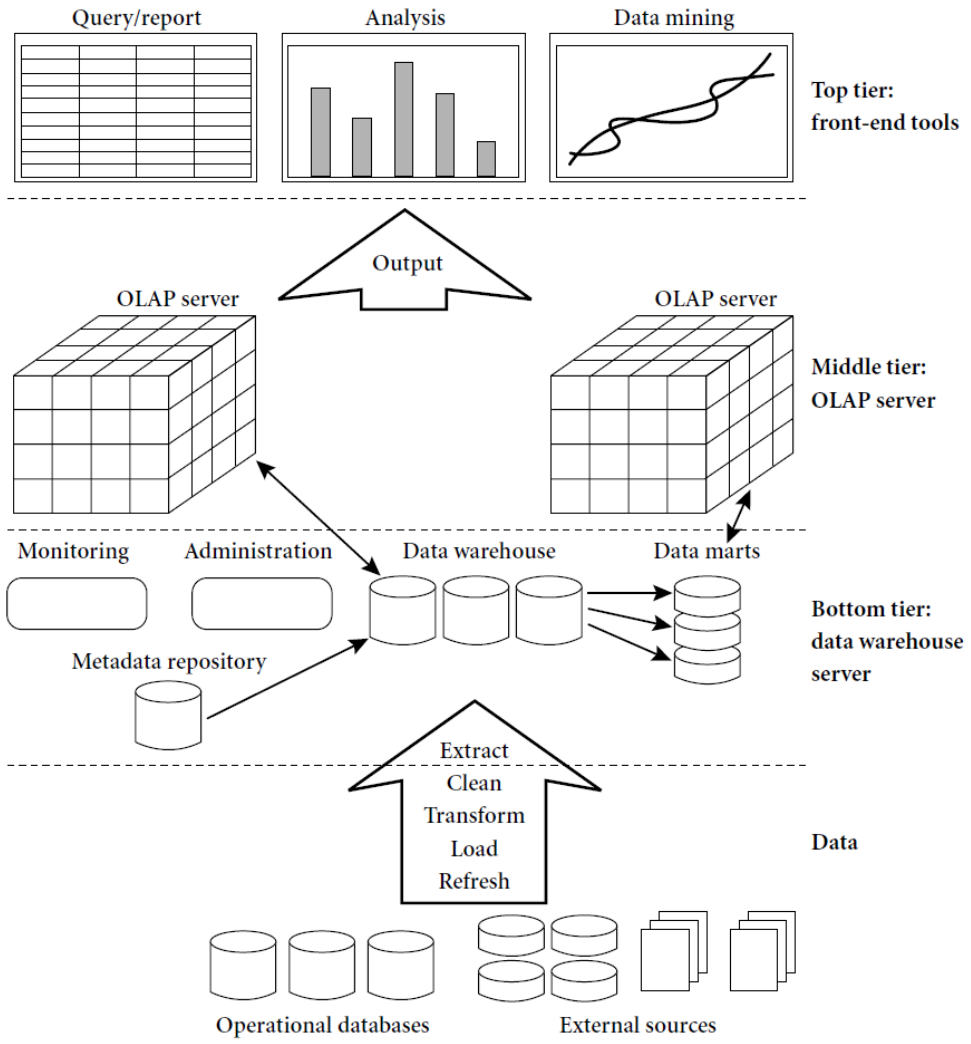
# Three-tier data warehouse architecture



# OLAP Server Architectures

- Relational OLAP (ROLAP)
  - Use relational or extended-relational DBMS to store and manage warehouse data and OLAP middle ware
  - Include optimization of DBMS backend, implementation of aggregation navigation logic, and additional tools and services
  - Greater scalability
- Multidimensional OLAP (MOLAP)
  - Sparse array-based multidimensional storage engine
  - Fast indexing to pre-computed summarized data
- Hybrid OLAP (HOLAP) (e.g., Microsoft SQLServer)
  - Flexibility, e.g., low level: relational, high-level: array
- Specialized SQL servers (e.g., Redbricks)
  - Specialized support for SQL queries over star/snowflake schemas

# Three-tier data warehouse architecture



# Chapter 3: Data Warehousing and OLAP Technology: An Overview

- What is a data warehouse?
- A multi-dimensional data model
- Data warehouse architecture
- Data warehouse implementation
- From data warehousing to data mining

# Data Warehouse Usage

- Three kinds of data warehouse applications
  - Information processing
    - supports querying, basic statistical analysis, and reporting using crosstabs, tables, charts and graphs
  - Analytical processing
    - multidimensional analysis of data warehouse data
    - supports basic OLAP operations, slice-dice, drilling, pivoting
  - Data mining
    - knowledge discovery from hidden patterns
    - supports associations, constructing analytical models, performing classification and prediction, and presenting the mining results using visualization tools

# From On-Line Analytical Processing (OLAP) to On Line Analytical Mining (OLAM)

- Why online analytical mining?
  - High quality of data in data warehouses
    - DW contains integrated, consistent, cleaned data
  - Available information processing structure surrounding data warehouses
    - ODBC, OLEDB, Web accessing, service facilities, reporting and OLAP tools
  - OLAP-based exploratory data analysis
    - Mining with drilling, dicing, pivoting, etc.
  - On-line selection of data mining functions
    - Integration and swapping of multiple mining functions, algorithms, and tasks

# From On-Line Analytical Processing (OLAP) to On Line Analytical Mining (OLAM)

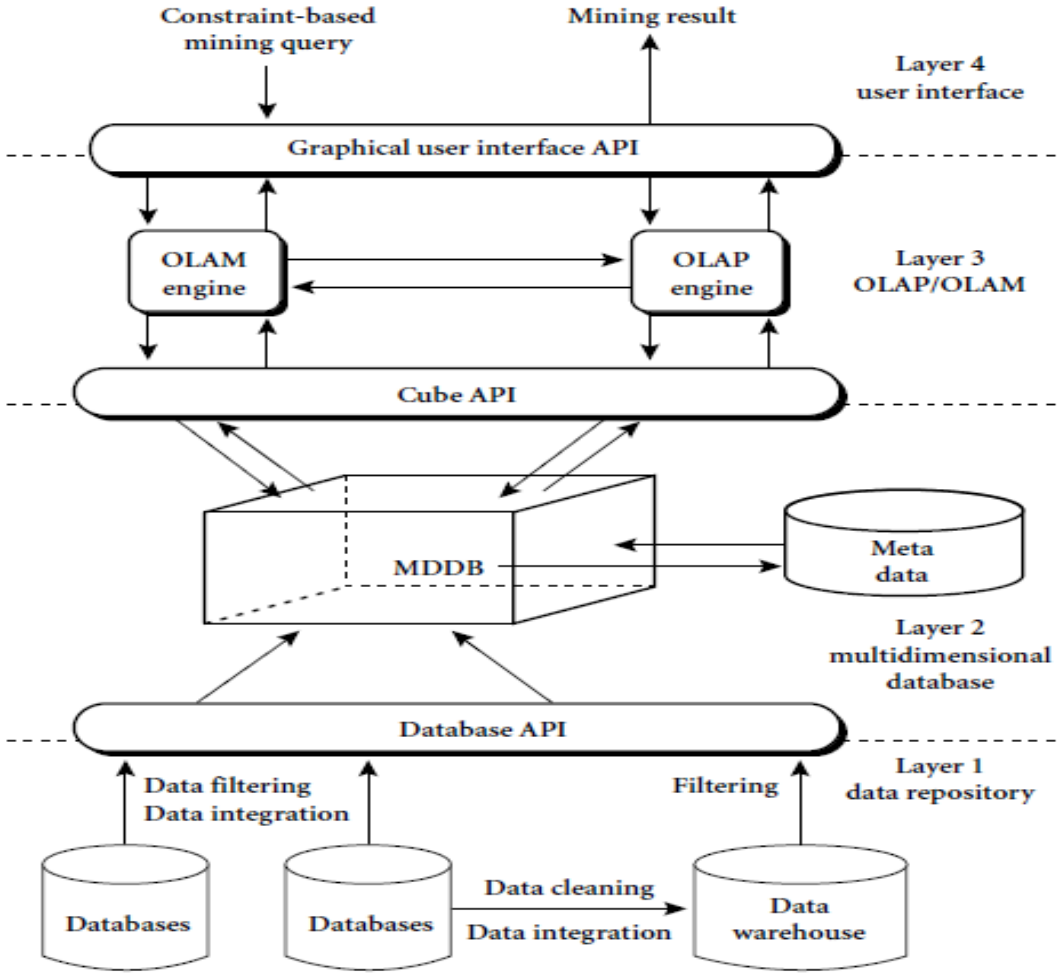



**Figure 3.18** An integrated OLAM and OLAP architecture.

# Efficient Data Cube Computation

- Data cube computation is an essential task in data warehouse implementation.
- Data cubes facilitate the on-line analytical processing of multidimensional data.
- The precomputation of all or part of a data cube can greatly reduce the response time and enhance the performance of on-line analytical processing.
- Data cube computation is challenging task
  - computational time and
  - storage

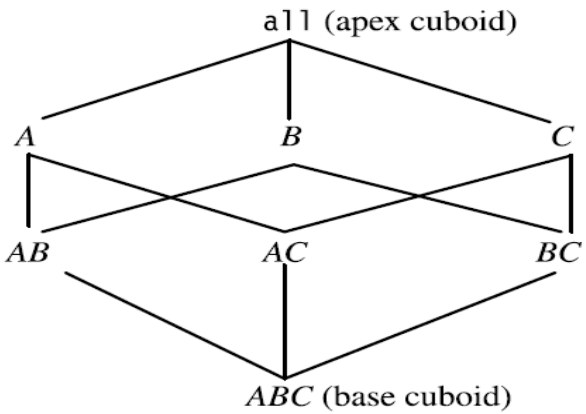
# Efficient Data Cube Computation

- Types of Data cubes
  - Full Cube,
  - Iceberg Cube,
  - Closed Cube, and
  - Shell Cube

- Data cube can be viewed as a lattice of cuboids
  - The bottom-most cuboid is the base cuboid
  - The top-most cuboid (apex) contains only one cell

# Efficient Data Cube Computation

- 3-D data cube for the dimensions *A*, *B*, and *C*, and an aggregate measure, *M*.




- A cell in the base cuboid is a base cell.

- A cell from a nonbase cuboid is an aggregate cell.

# Efficient Data Cube Computation

- An aggregate cell aggregates over one or more dimensions
- Each aggregated dimension is indicated by a " * " in the cell notation.
- An *n*-dimensional data cube. Let *a* = (*a*1, *a*2,…. , *an*, *measures*) be a cell from one of the cuboids making up the data cube.

# Efficient Data Cube Computation

- A data cube with the dimensions *month, city*, and *customer group*, and the measure *price*.
- (*Jan*, * , * , 2800) and (*, *Toronto*, * , 1200) are 1-D cells,
- (*Jan*, * , *Business*, 150) is a 2-D cell, and (*Jan*, *Toronto*, *Business*, 45) is a 3-D cell.
- All base cells are 3-D, whereas 1-D and 2-D cells are aggregate cells.

# Efficient Data Cube Computation

- An ancestor-descendant relationship may exist between cells.
- In an *n*-dimensional data cube, an *i*-D cell *a* = (*a*1, *a*2, … , *an*, *measuresa*) is an ancestor of a *j*-D cell *b* = (*b*1, *b*2, … , *bn*, *measuresb*),
- And *b* is a descendant of *a*, if and only if
  - (1) *i* < *j*, and
  - (2) for 1 *m* *n*, *am* = *bm* whenever *am* 6= "".
- Cell *a* is called a parent of cell *b*, and *b* is a child of *a*, if and only if *j* = *i*+1 and *b* is a descendant of *a*.

# Efficient Data Cube Computation

- 1-D cell *a*=(*Jan*, * , * , 2800), and
- 2-D cell *b* = (*Jan*, * , *Business*, 150), are *ancestors* of 3-D cell *c* = (*Jan*, *Toronto*, *Business*, 45);
- *c* is a *descendant* of both *a* and *b*; *b* is a *parent* of *c*, and *c* is a *child* of *b*.

# Efficient Data Cube Computation

- **Full Cube:**
- To ensure fast on-line analytical processing, it is sometimes desirable to precompute the full cube (i.e., all the cells of all of the cuboids for a given data cube).

- This computation is exponential to the number of dimensions. That is, a data cube of $n$ dimensions contains $2^n$ cuboids.

- There are even more cuboids if we consider concept hierarchies for each dimension.

- The size of each cuboid depends on the cardinality of its dimensions.

# Efficient Data Cube Computation

- Precomputation of the full cube can require huge and often excessive amounts of memory.
- Individual cuboids may be stored on secondary storage and accessed when necessary.
- Also, compute smaller cubes, consisting of a subset of the given set of dimensions, or a smaller range of possible values for some of the dimensions.
- The smaller cube is a full cube for the given subset of dimensions and/or dimension values.

# Efficient Data Cube Computation

- It is important to explore scalable methods for computing all of the cuboids making up a data cube, that is, for **full materialization.**
  - limited amount of main memory available for cuboid computation.
  - The total size of the computed data cube,
  - The time required for such computation.
- Partial materialization of data cubes offers an interesting trade-off between storage space and response time for OLAP.

- Instead, we can compute only a subset of the data cube's cuboids, or subcubes consisting of subsets of cells from the various cuboids.

# Efficient Data Cube Computation

- Many cells in a cuboid may actually be of little or no interest to the data analyst.
- Measures such as *count*, *sum*, or *sales in dollars* are commonly used.
- For many cells in a cuboid, the measure value will be zero.
- The product of the cardinalities for the dimensions in a cuboid is large relative to the number of nonzero-valued tuples that are stored in the cuboid, then we say that the cuboid is sparse.
- If a cube contains many sparse cuboids, we say that the cube is sparse.

# Efficient Data Cube Computation

- The cube cells are often quite sparsely distributed within a multiple dimensional space.

- For example, a customer may only buy a few items in a store at a time.
  - generate only a few nonempty cells,
  - leaving most other cube cells empty
  - it is useful to materialize only those cells in a cuboid (group-by) whose measure value is above some minimum threshold.
- In a data cube for sales wish materialize only those cells for which *count >10* or only those cells representing *sales* $>*100*

# Efficient Data Cube Computation

- Saves processing time and disk space, but also leads to a more focused analysis.
- Partially materialized cubes are known as iceberg cubes.
- Materializing only a fraction of the cells in a data cube, the result is seen as the "tip of the iceberg"
- The "iceberg" is the potential full cube including all cells.
- An iceberg cube can be specified with an SQL query, as shown in the next slide.

# Iceberg Cube

*compute cube sales iceberg as select month, city, customer group, count(*) from salesInfo*

*cube by month, city, customer group*

*having count(*) >= min sup*

- The constraint specified in the having clause is known as the iceberg condition.
- Motivation
  - Only a small portion of cube cells may be "above the water" in a sparse cube
  - Only calculate "interesting" cells—data above certain threshold

# Iceberg Cube

- The *iceberg* cubes will lessen the burden of computing trivial aggregate cells in a data cube.

- However, we could still end up with a large number of uninteresting cells to compute.
  - Avoid explosive growth of the cube
    - Suppose 100 dimensions, only 1 base cell. How many aggregate cells if count >= 1? What about count >= 2?

# closed cube

- Need to compress a data cube
- Closed cube:
  - A cell, *c*, is a *closed cell* if there exists no cell, *d*, such that *d* is a specialization (descendant) of cell *c* (that is, where *d* is obtained by replacing a  in *c* with a non- value),and *d* has the same measure value as *c*.
- A closed cube is a data cube consisting of only closed cells.

# Multi-Way Array Aggregation

- The Multiway Array Aggregation (or simply MultiWay) method computes a full data cube by using a multidimensional array as its basic data structure.

- A different approach is developed for the array-based cube construction.
  - (1) Partition the array into chunks.
  - (2) Compute aggregates by visiting cube cells.

# Multi-Way Array Aggregation

- 1. Partition the array into chunks.
  - A chunk is a subcube that is small enough to fit into the memory available for cube computation.
  - Chunking is a method for dividing an *n*-dimensional array into small *n*-dimensional chunks.
  - The chunks are compressed so as to remove wasted space resulting from empty array cells (i.e., cells that do not contain any valid data, whose cell count is zero).
  - For instance, "*chunkID* + *offset*" can be used as a cell addressing mechanism to compress a sparse array structure and when searching for cells within a chunk.
  - Such a compression technique is powerful enough to handle sparse cubes, both on disk and in memory.

# Multi-Way Array Aggregation

- (2) Compute aggregates by visiting (i.e., accessing the values at) cube cells.
  - The order in which cells are visited can be optimized.
  - To *minimize the number of times that each cell must be revisited*, thereby reducing memory access and storage costs.
  - The trick is to exploit this ordering
  - The partial aggregates can be computed simultaneously, and any unnecessary revisiting of cells is avoided.

# Example

- **Multiway array cube computation.**
- Consider a 3-D data array containing the three dimensions *A*, *B*, and *C*.
- The 3-D array is partitioned into small, memory-based chunks.
- Dimension *A* is organized into four equal-sized partitions, *a*0, *a*1, *a*2, and *a*3.
- Dimensions *B* and *C* are similarly organized into four partitions each.
- Chunks 1, 2, . . . , 64 correspond to the subcubes *a*0*b*0*c*0, *a*1*b*0*c*0, . . . , *a*3*b*3*c*3, respectively.

# Example

- Suppose that the cardinality of the dimensions *A*, *B*, and *C* is 40, 400, and 4000, respectively.
- The size of the array for each dimension, *A*, *B*, and *C*, is also 40, 400, and 4000, respectively.
- The size of each partition in *A*, *B*, and *C* is therefore 10, 100, and 1000, respectively.
- Full materialization of the corresponding data cube involves the computation of all of the cuboids defining this cube.

# Example

- The resulting full cube consists of the following cuboids:
- The base cuboid, denoted by *ABC* (from which all of the other cuboids are directly or indirectly computed).
- This cube is already computed and corresponds to the given 3-D array.
- The 2-D cuboids, *AB*, *AC*, and *BC*, which respectively correspond to the group-by's *AB*, *AC*, and *BC*. These cuboids must be computed.
- The 1-D cuboids, *A*, *B*, and *C*, which respectively correspond to the group-by's *A*, *B*, and *C*.

# Example

- These cuboids must be computed.
- The 0-D (apex) cuboid, denoted by all, which corresponds to the group-by (); that is, there is no group-by here.
- This cuboid must be computed.
- It consists of one value.
- If, say, the data cube measure is count, then the value to be computed is simply the total count of all of the tuples in *ABC*.

# Example

- For example, when chunk 1 (i.e., *a*0*b*0*c*0) is being scanned (say, for the computation of the 2-D chunk *b*0*c*0 of *BC*, as described above), all of the other 2-D chunks relating to *a*0*b*0*c*0 can be simultaneously computed.

- That is, when *a*0*b*0*c*0 is being scanned, each of the three chunks, *b*0*c*0, *a*0*c*0, and *a*0*b*0, on the three 2-D aggregation planes, *BC*, *AC*, and *AB*, should be computed then as well.

- In other words, multiway computation simultaneously aggregates to each of the 2-D planes while a 3-D chunk is in memory.

# Example

- Therefore, the largest 2-D plane is *BC* (of size 400x4000 = 1, 600, 000).
- The second largest 2-D plane is *AC* (of size 40x4000 = 160, 000).
- *AB* is the smallest 2-D plane (with a size of 40x400 = 16, 000).

# Multi-way Array Aggregation for Cube Computation (MOLAP)

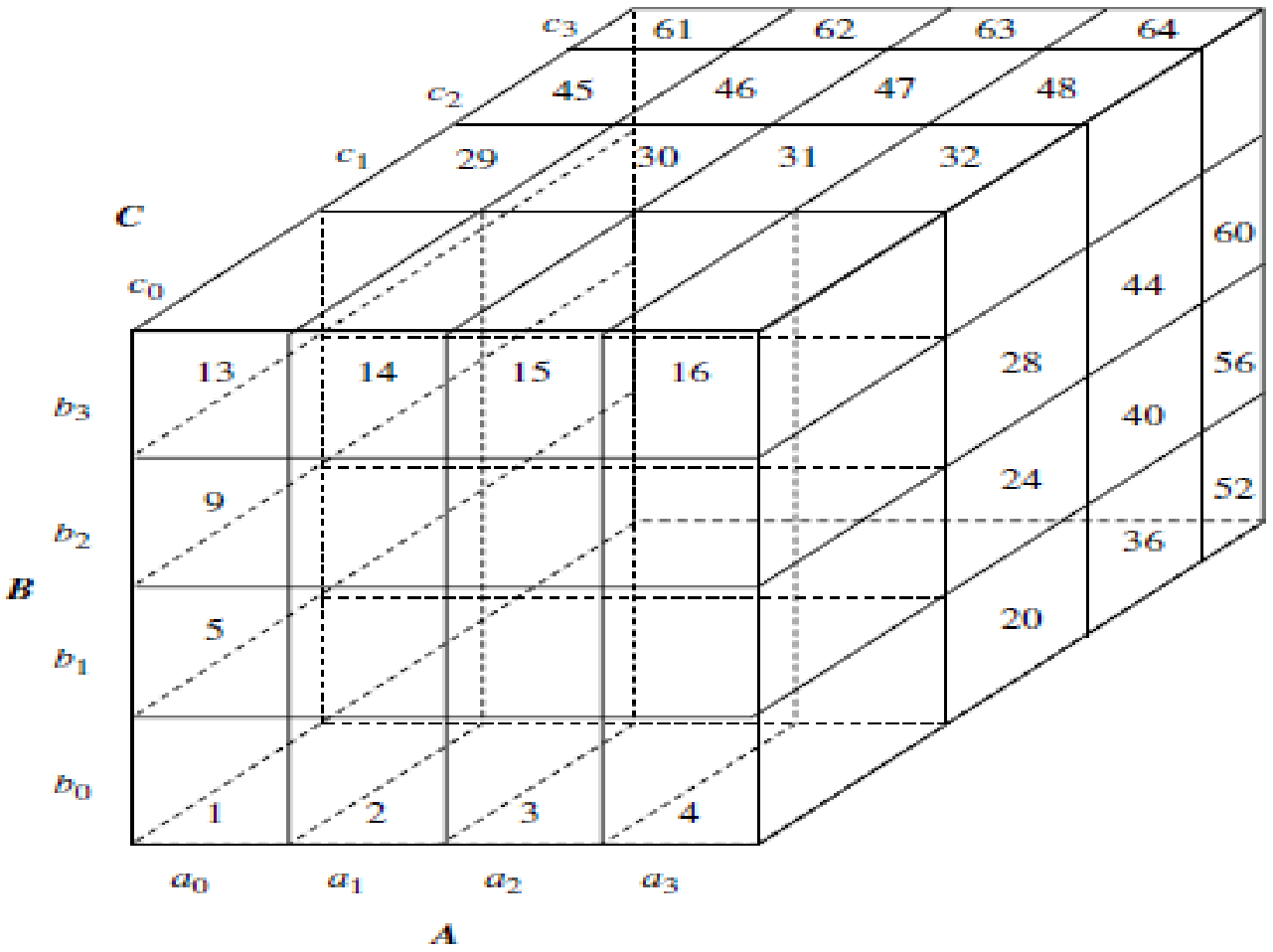
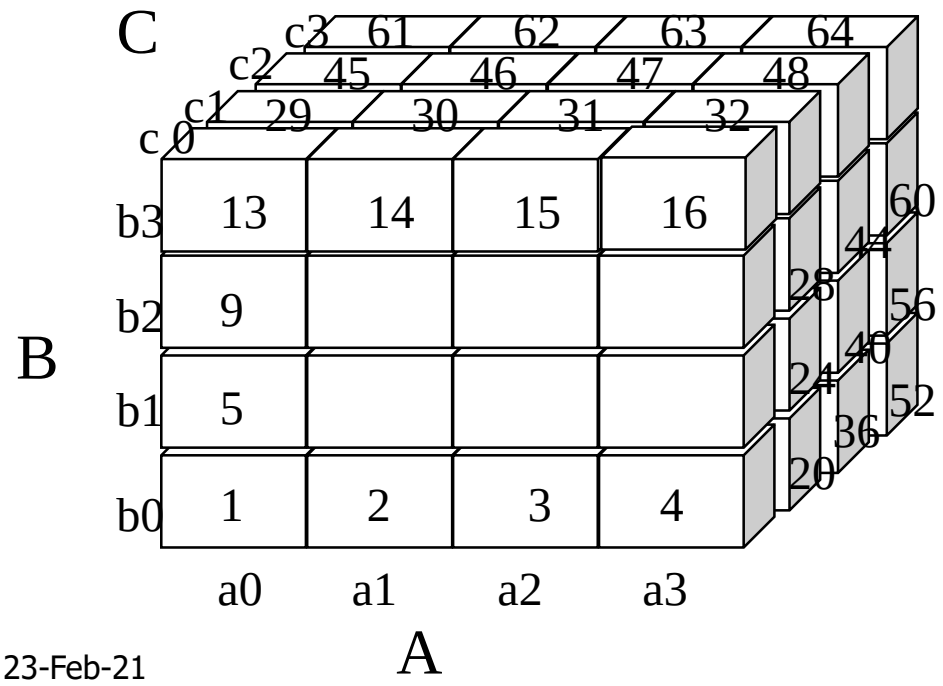



**Figure 4.3** A 3-D array for the dimensions *A*, *B*, and *C*, organized into 64 *chunks*. Each chunk is small enough to fit into the memory available for cube computation.

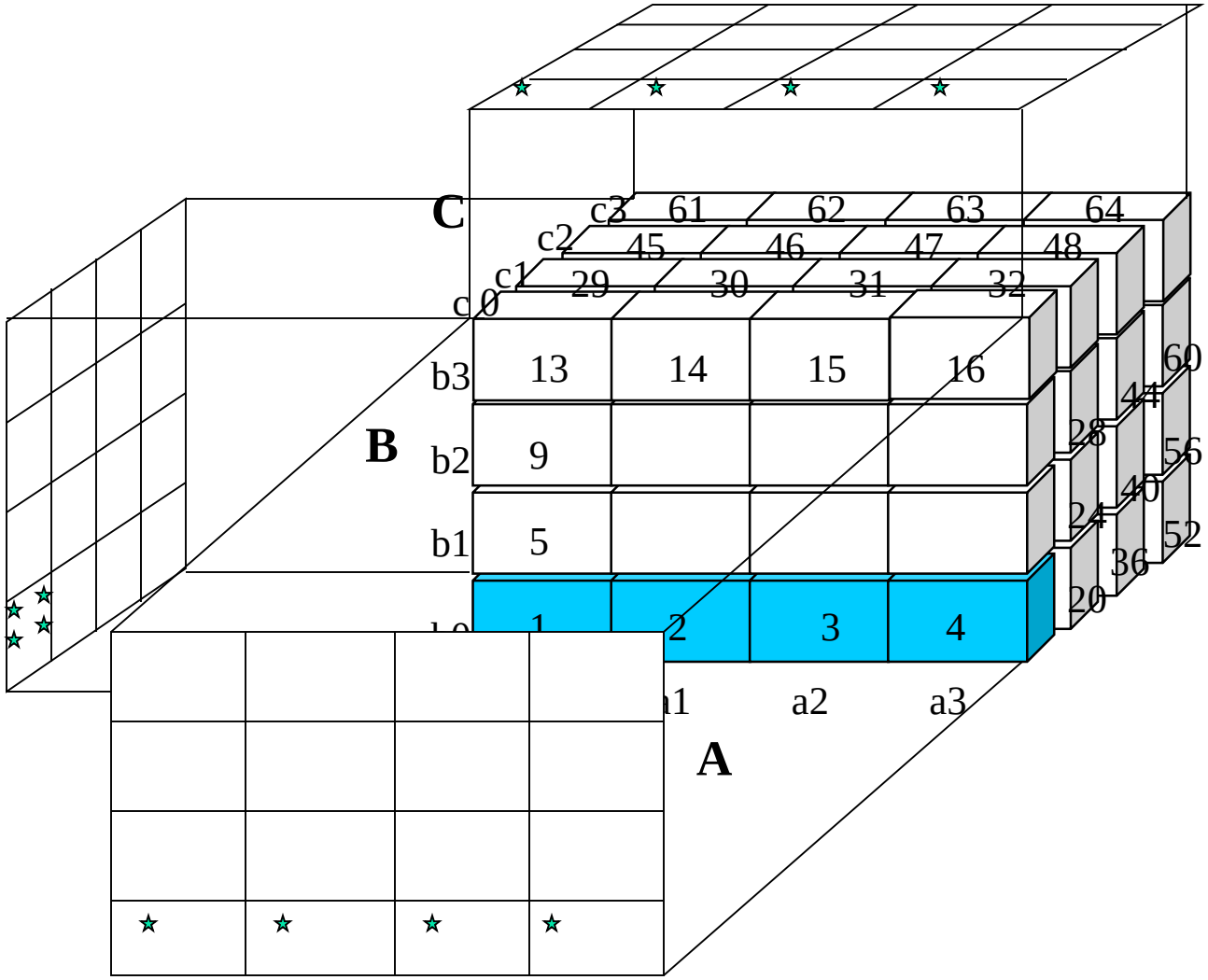
# Multi-way Array Aggregation for Cube Computation (MOLAP)

- Partition arrays into chunks (a small subcube which fits in memory).
- Compressed sparse array addressing: (chunk_id, offset)
- Compute aggregates in "multiway" by visiting cube cells in the order which minimizes the # of times to visit each cell, and reduces memory access and storage cost.

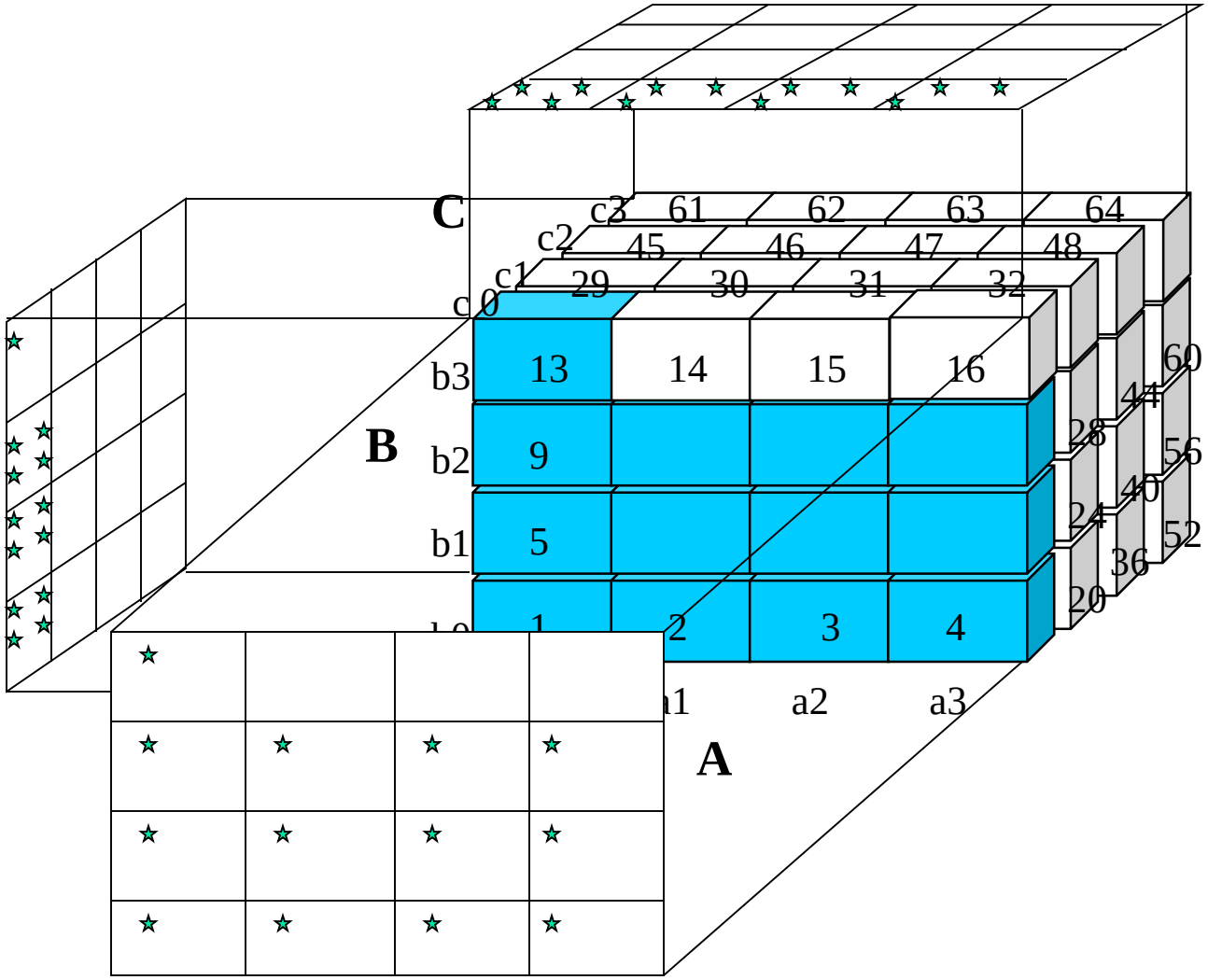



**What is the best traversing order to do multi-way aggregation?**

# Multi-way Array Aggregation for Cube Computation



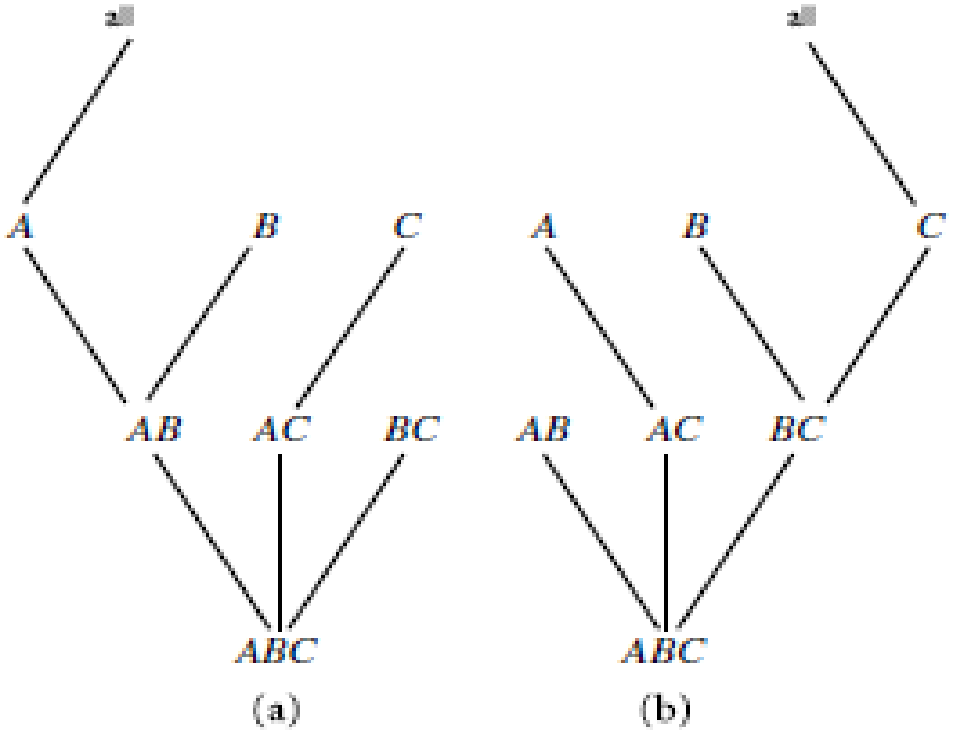
# Multi-way Array Aggregation for Cube Computation



# Multi-Way Array Aggregation for Cube Computation (Cont.)

- Method: the planes should be sorted and computed according to their size in ascending order
  - Idea: keep the smallest plane in the main memory, fetch and compute only one chunk at a time for the largest plane
- Limitation of the method: computing well only for a small number of dimensions
  - If there are a large number of dimensions, "top-down" computation and iceberg cube computation methods can be explored

# Multi-Way Array Aggregation for Cube Computation (Cont.)




- Two orderings of multiway array aggregation for computation of the 3-D cube of Example 4.4: (a) most efficient ordering of array aggregation (minimum memory requirements = 156,000 memory units); (b) least efficient ordering of array aggregation (minimum memory requirements = 1,641,000 memory units).

# Indexing OLAP Data: Bitmap Index

- To facilitate efficient data accessing, most data warehouse systems support index structures and materialized views
- The bitmap indexing allows quick searching in data cubes.
- Index on a particular column
- Each value in the column has a bit vector: bit-op is fast
- The length of the bit vector: # of records in the base table
- The *i-th* bit is set if the *i*-th row of the base table has the value for the indexed column
- not suitable for high cardinality domains

# Indexing OLAP Data: Bitmap Index

| Cust | Region | Type |
|---|---|---|
| C1 | Asia | Retail |
| C2 | Europe | Dealer |
| C3 | Asia | Dealer |
| C4 | America | Retail |
| C5 | Europe | Dealer |

**Base table**

| RecID | Asia | Europe | America |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 |
| 3 | 1 | 0 | 0 |
| 4 | 0 | 0 | 1 |
| 5 | 0 | 1 | 0 |

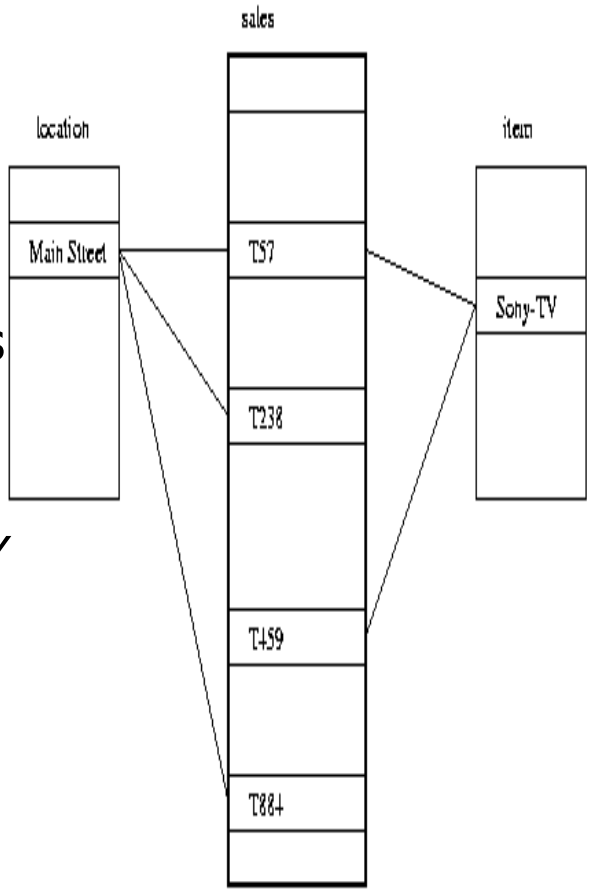
**Index on Region**

| RecID | Retail | Dealer |
|---|---|---|
| 1 | 1 | 0 |
| 2 | 0 | 1 |
| 3 | 0 | 1 |
| 4 | 1 | 0 |
| 5 | 0 | 1 |

**Index on Type**

# Indexing OLAP Data: Join Indices

- Join index: JI(R-id, S-id) where R (R-id, …) ⋈ S (S-id, …)
- Traditional indices map the values to a list of record ids
  - It materializes relational join in JI file and speeds up relational join
- In data warehouses, join index relates the values of the dimensions of a start schema to rows in the fact table.
  - E.g. fact table: *Sales* and two dimensions *city* and *product*
    - A join index on *city* maintains for each distinct city a list of R-IDs of the tuples recording the Sales in the city
  - Join indices can span multiple dimensions



# Efficient Processing OLAP Queries

- Determine which operations should be performed on the available cuboids
  - Transform drill, roll, etc. into corresponding SQL and/or OLAP operations, e.g., dice = selection + projection
- Determine which materialized cuboid(s) should be selected for OLAP op.
  - Let the query to be processed be on {brand, province_or_state} with the condition "year = 2004", and there are 4 materialized cuboids available:

    1) {year, item_name, city}

    2) {year, brand, country}

    3) {year, brand, province_or_state}

    4) {item_name, province_or_state} where year = 2004

    Which should be selected to process the query?
- Explore indexing structures and compressed vs. dense array structs in MOLAP

# References (I)


- S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. VLDB'96
- D. Agrawal, A. E. Abbadi, A. Singh, and T. Yurek. Efficient view maintenance in data warehouses. SIGMOD'97
- R. Agrawal, A. Gupta, and S. Sarawagi. Modeling multidimensional databases. ICDE'97
- S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. ACM SIGMOD Record, 26:65-74, 1997
- E. F. Codd, S. B. Codd, and C. T. Salley. Beyond decision support. Computer World, 27, July 1993.
- J. Gray, et al. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals. Data Mining and Knowledge Discovery, 1:29-54, 1997.
- A. Gupta and I. S. Mumick. Materialized Views: Techniques, Implementations, and Applications. MIT Press, 1999.
- J. Han. Towards on-line analytical mining in large databases. *ACM SIGMOD Record*, 27:97-107, 1998.
- V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. SIGMOD'96

# References (II)


- C. Imhoff, N. Galemmo, and J. G. Geiger. Mastering Data Warehouse Design: Relational and Dimensional Techniques. John Wiley, 2003
- W. H. Inmon. Building the Data Warehouse. John Wiley, 1996
- R. Kimball and M. Ross. The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling. 2ed. John Wiley, 2002
- P. O'Neil and D. Quass. Improved query performance with variant indexes. SIGMOD'97
- Microsoft. OLEDB for OLAP programmer's reference version 1.0. In http://www.microsoft.com/data/oledb/olap, 1998
- A. Shoshani. OLAP and statistical databases: Similarities and differences. PODS'00.
- S. Sarawagi and M. Stonebraker. Efficient organization of large multidimensional arrays. ICDE'94
- OLAP council. MDAPI specification version 2.0. In http://www.olapcouncil.org/research/apily.htm, 1998
- E. Thomsen. OLAP Solutions: Building Multidimensional Information Systems. John Wiley, 1997
- P. Valduriez. Join indices. ACM Trans. Database Systems, 12:218-246, 1987.
- J. Widom. Research problems in data warehousing. CIKM'95.