Data Warehousing and Decision Support

Chapter 25

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
  - Supports 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—Non-Volatile

- 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:
  - Build wrappers/mediators on top of heterogeneous databases
  - Query driven approach
    - 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
- 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 detailed, flat relational isolated	historical, summarized, multidimensional integrated, consolidated
usage	repetitive	ad-hoc
access	read/write 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

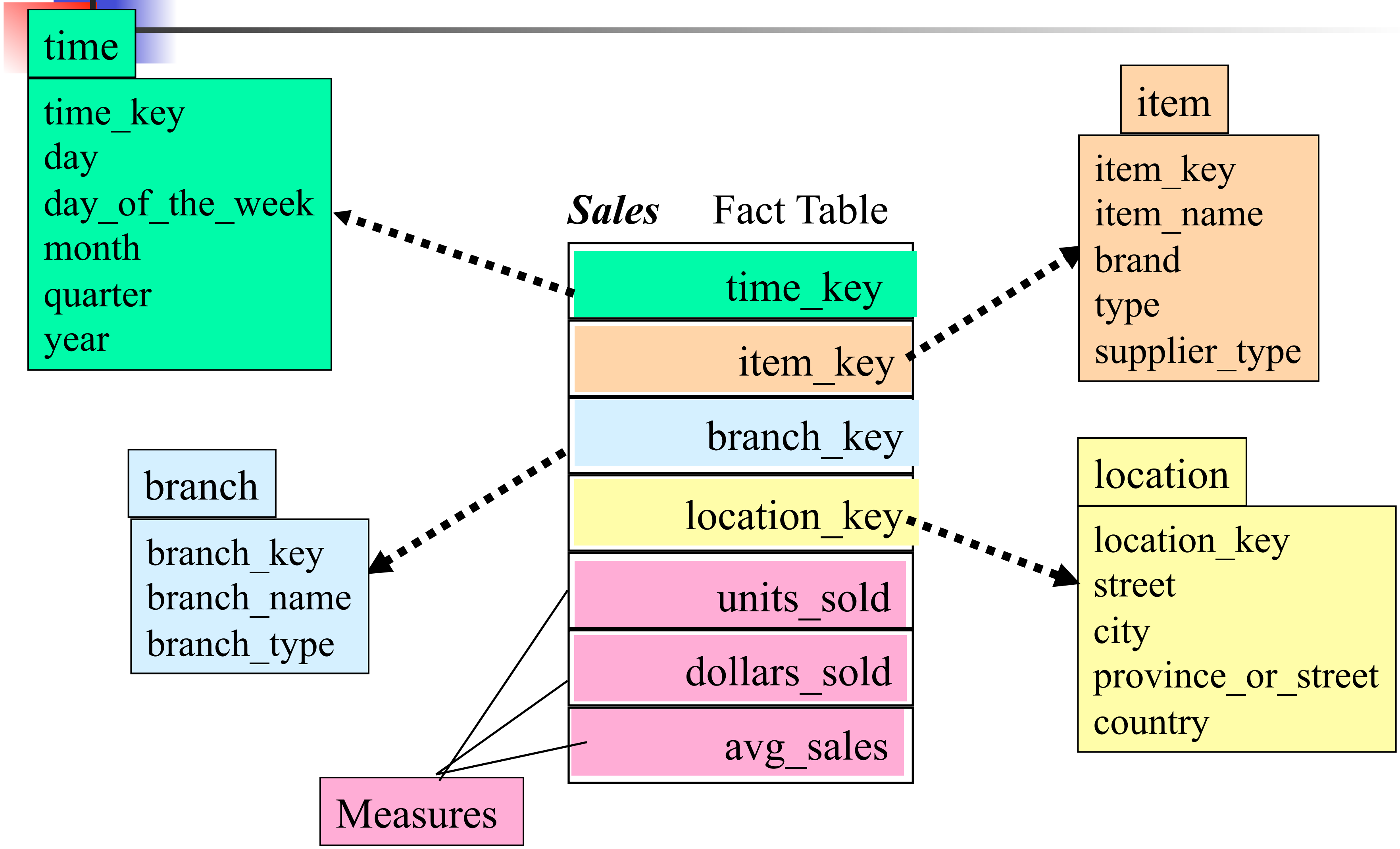
Why Separate Data Warehouse?

- 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: DW 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

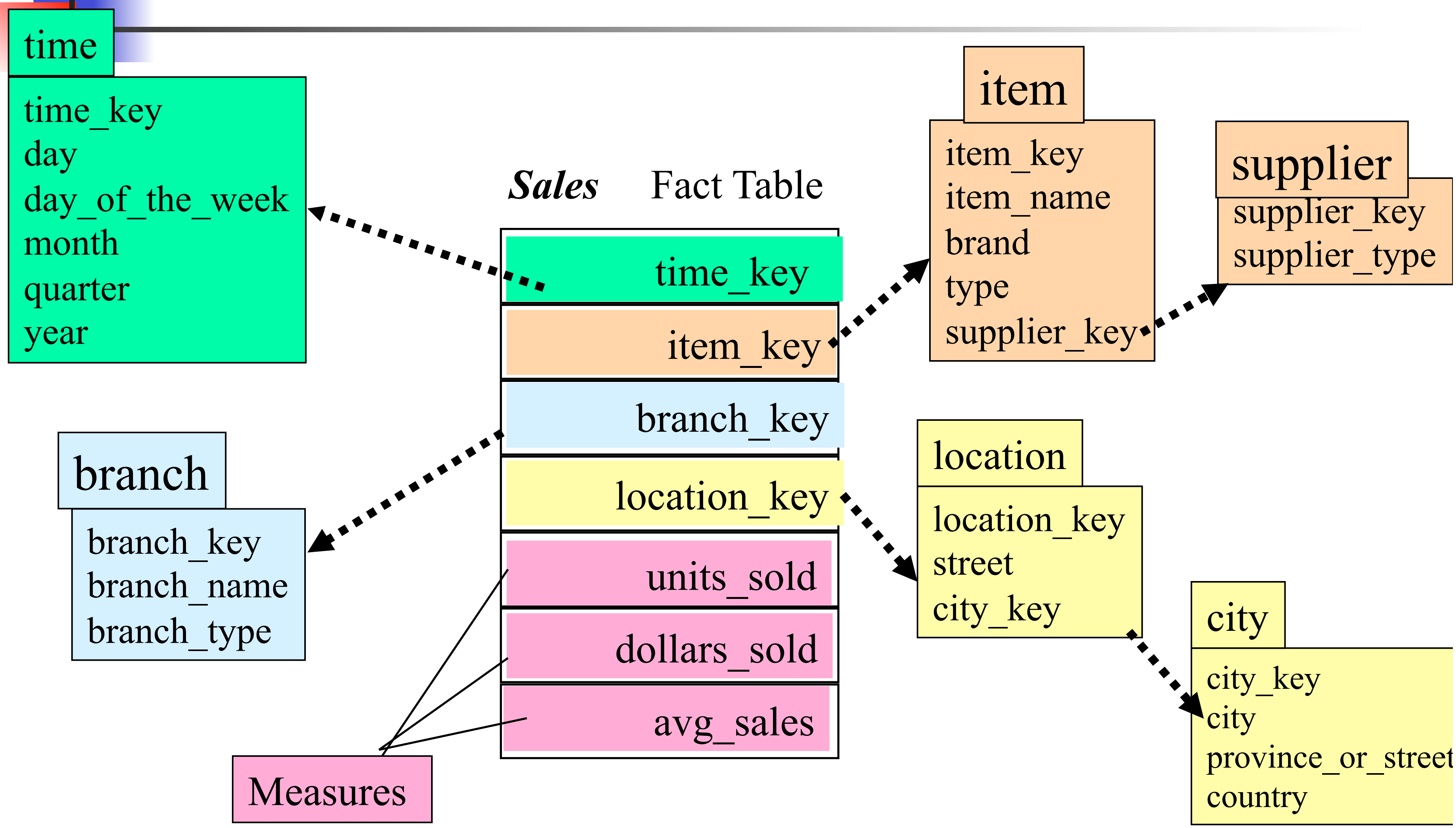
Conceptual Modeling of Data Warehouses

- 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

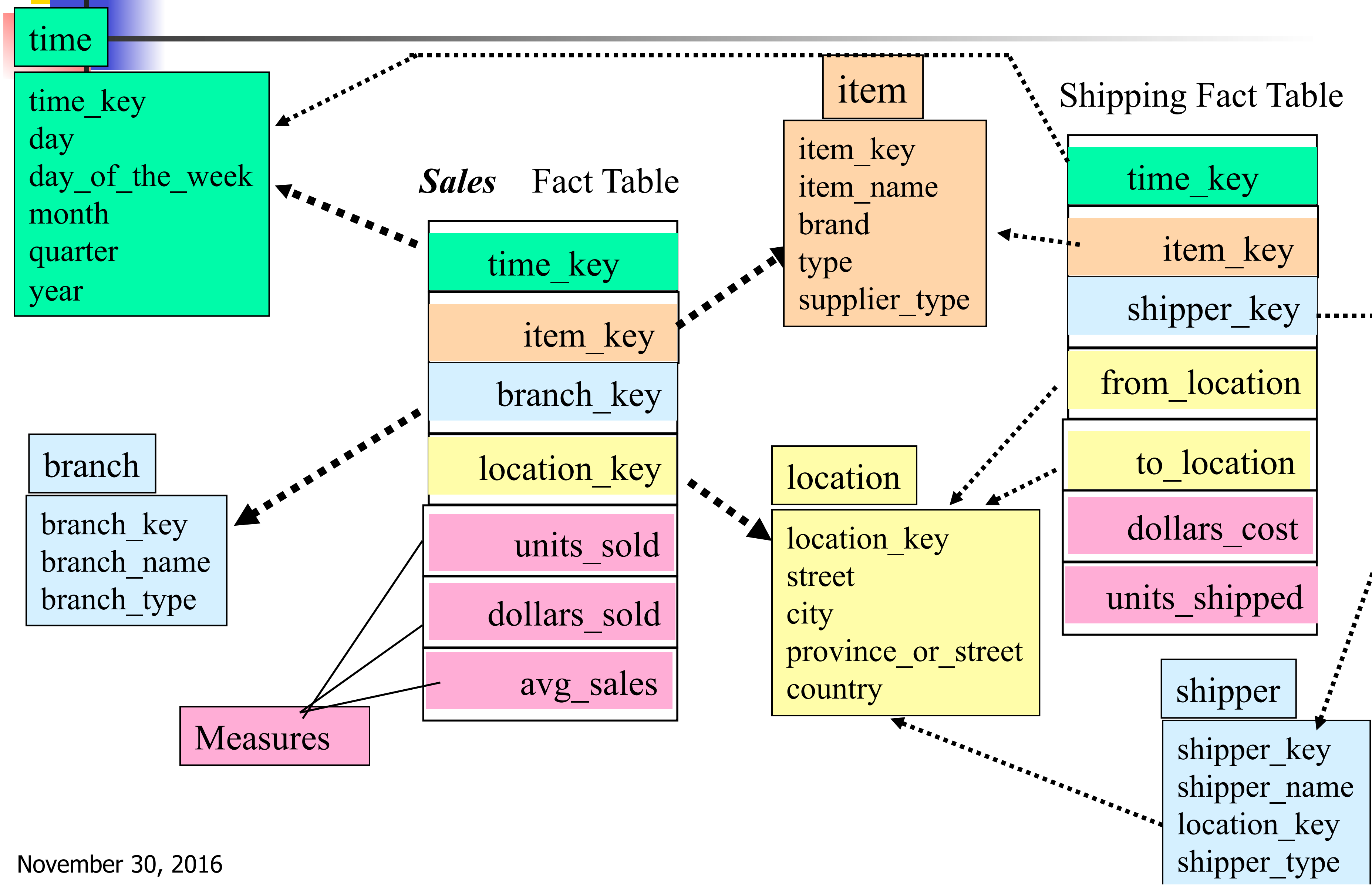
Example of Star Schema



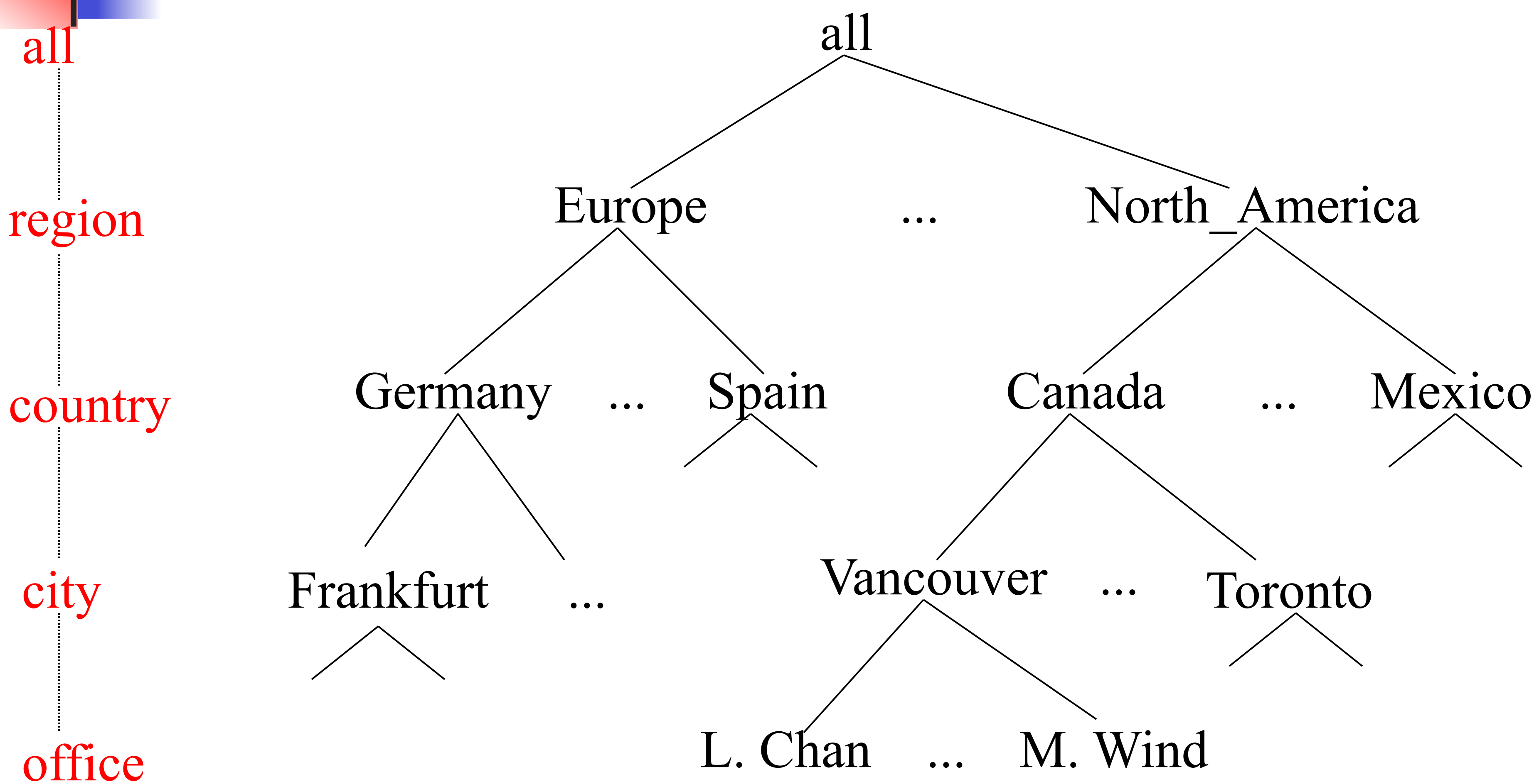
Example of Snowflake Schema



Example of Fact Constellation



A Concept Hierarchy: Dimension (location)

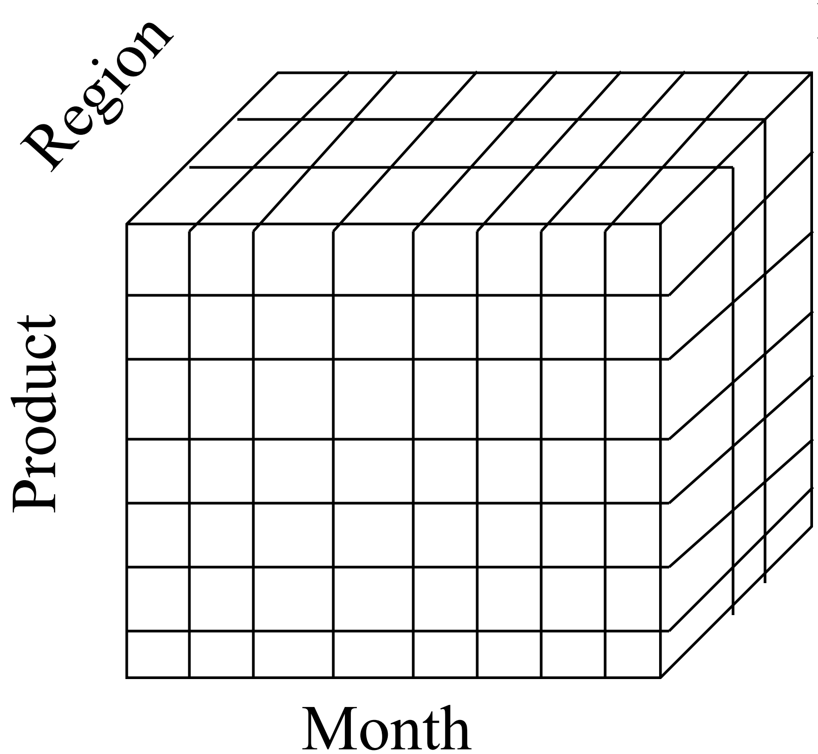


From Tables and Spreadsheets to Data Cubes

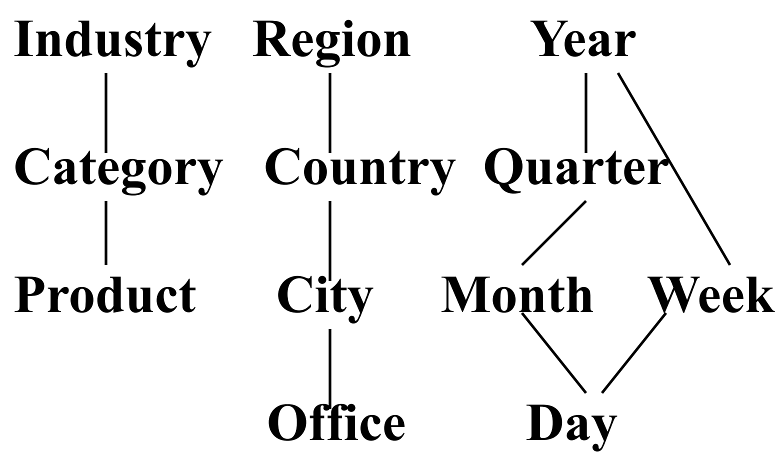
- A data warehouse is based on a multidimensional data model which views data in the form of a data cube
- A data cube, such as sales, allows data to be modeled and viewed in multiple dimensions
  - Dimension tables, such as item (item_name, brand, type), or time(day, week, month, quarter, year)
  - 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.

Multidimensional Data

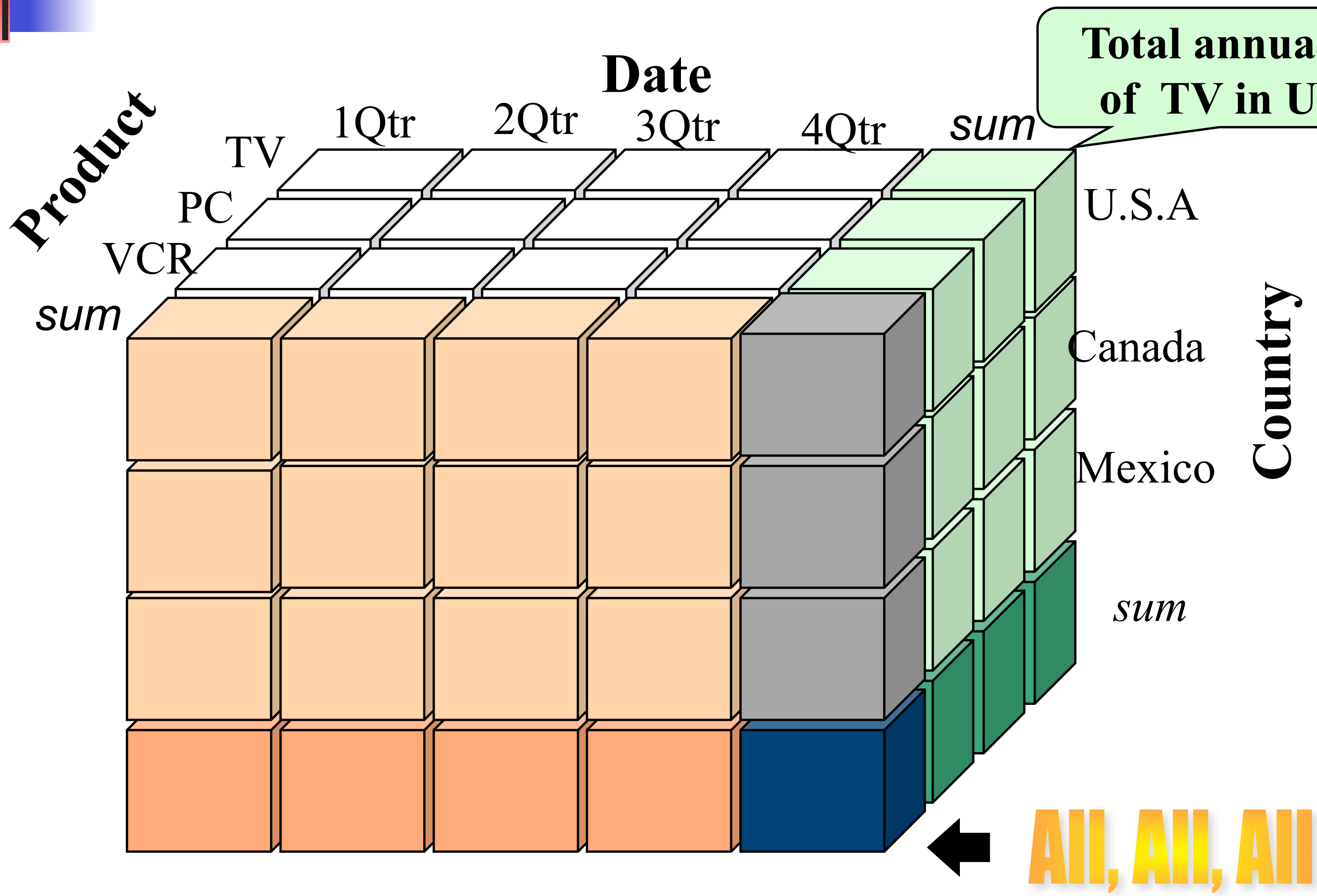
- Sales volume as a function of product, month, and region



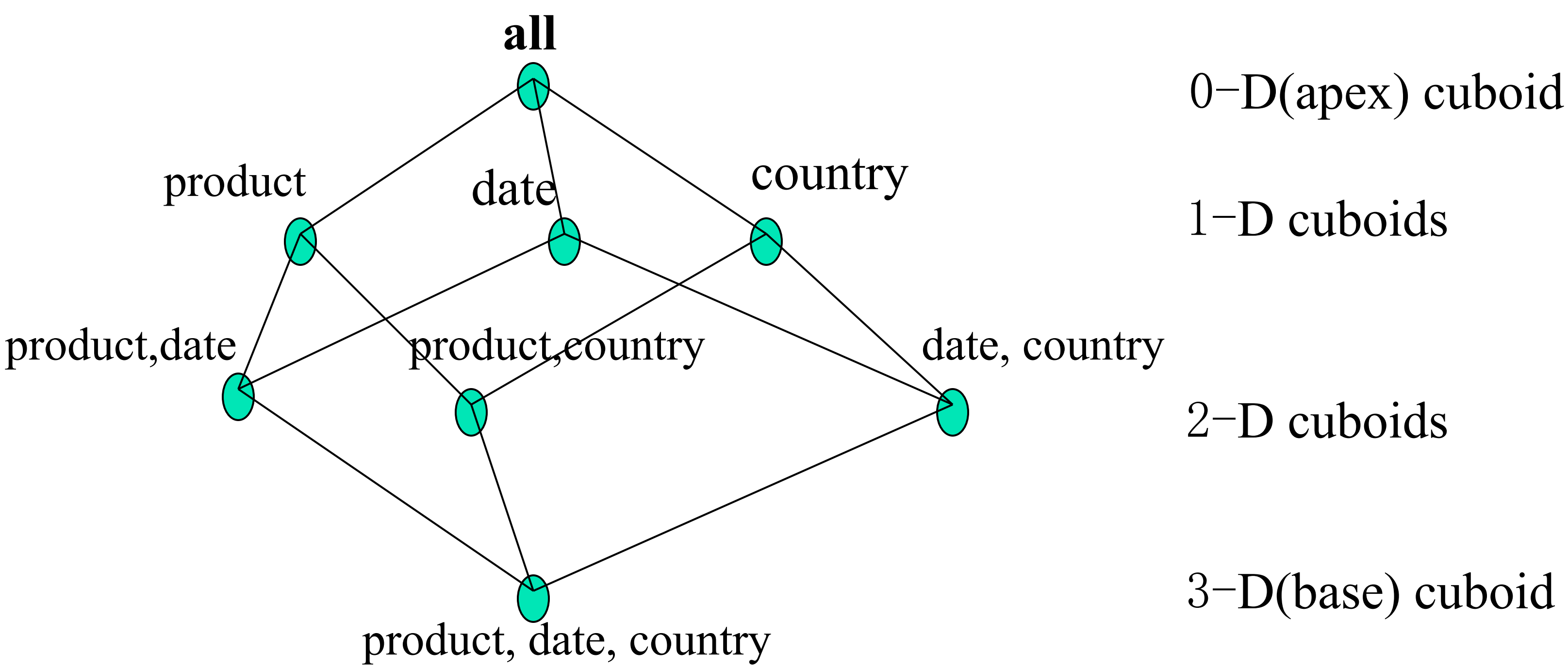
Dimensions: Product, Location, Time
Hierarchical summarization paths



A Sample Data Cube



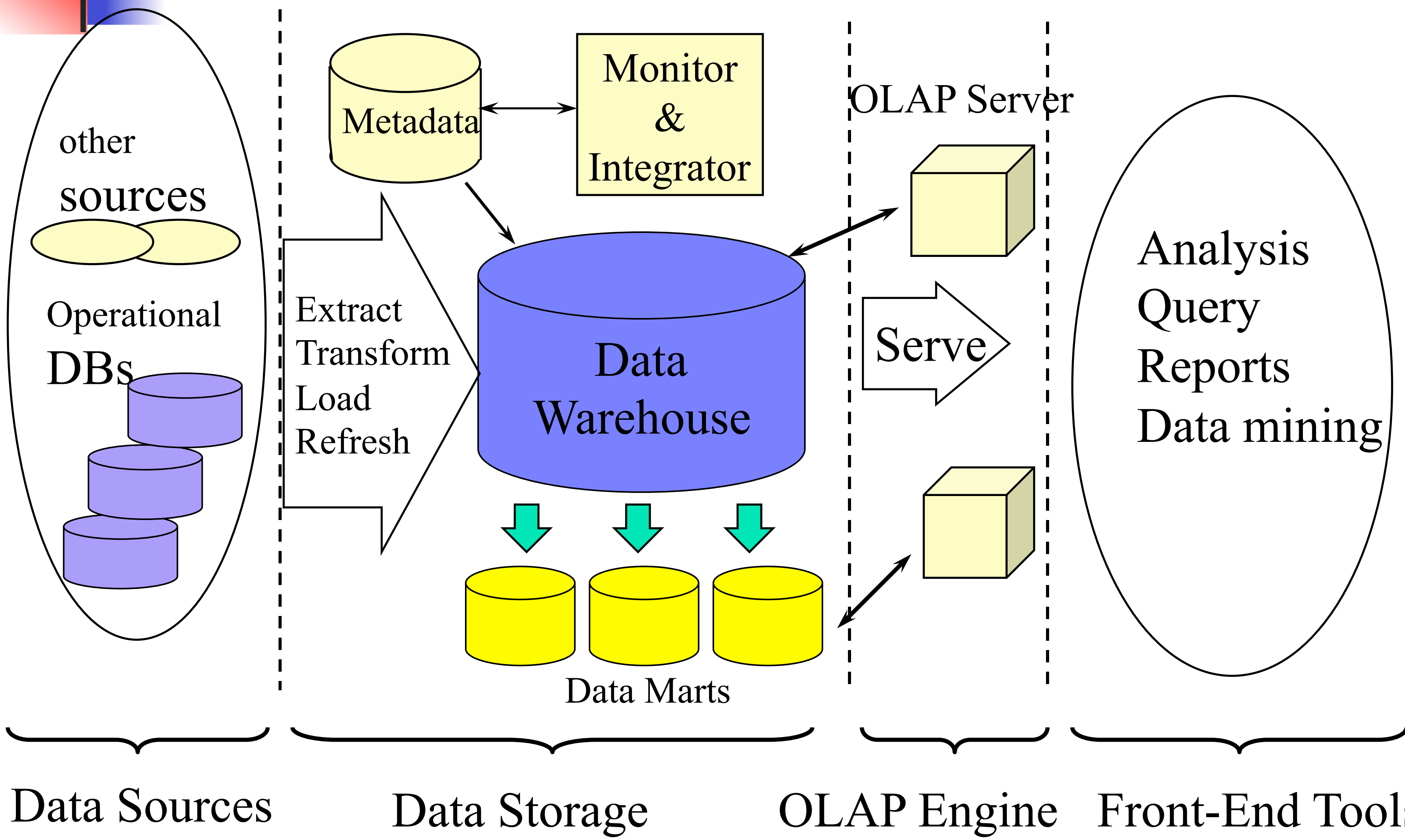
Cuboids Corresponding to the Cube



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):
  - *aggregation on selected dimensions.*
- 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)*

Multi-Tiered Architecture



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

OLAP Server Architectures

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

Efficient Data Cube Computation

- Data cube can be viewed as a lattice of cuboids
  - The bottom-most cuboid is the base cuboid
  - The top-most cuboid (apex) contains only one cell
  - How many cuboids in an n-dimensional cube?

$$2^n$$

Problem: How to Implement Data Cube Efficiently?

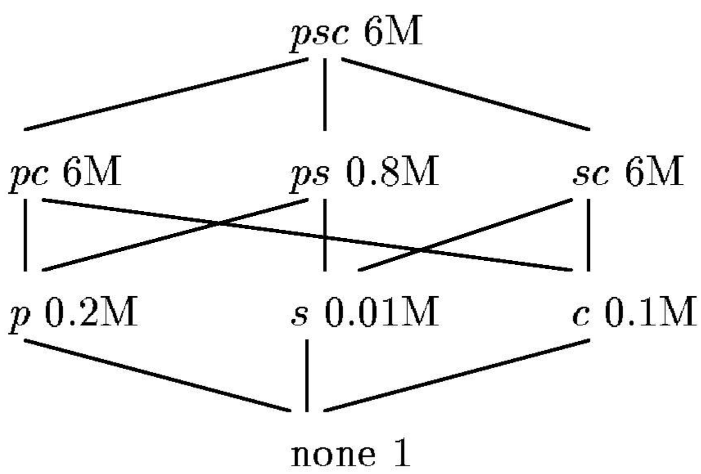
- Physically materialize the whole data cube
  - Space consuming in storage and time consuming in construction
  - Indexing overhead
- Materialize nothing
  - No extra space needed but unacceptable response time
- Materialize only part of the data cube
  - Intuition: precompute frequently-asked queries?
  - However: each cell of data cube is an aggregation, the value of many cells are dependent on the values of other cells in the data cube
  - A better approach: materialize queries which can help answer many other queries quickly

Motivating example

- Assume the data cube:
  - Stored in a relational DB (MDDB is not very scalable)
  - Different cuboids are assigned to different tables
  - The cost of answering a query is proportional to the number of rows examined
- Use TPC-D decision-support benchmark
  - Attributes: *part*, *supplier*, and *customer*
  - Measure: total *sales*
  - 3-D data cube: cell (*p*, *s* ,*c*)

Motivating example (cont.)

- Hypercube lattice: the eight views (cuboids) constructed by grouping on some of *part*, *supplier*, and *customer*



Finding total *sales* grouped by *part*

- Processing 6 million rows if cuboid *pc* is materialized
- Processing 0.2 million rows if cuboid *p* is materialized
- Processing 0.8 million rows if cuboid *ps* is materialized

Motivating example (cont.)

How to find a good set of queries?

- How many views must be materialized to get reasonable performance?
- Given space S, what views should be materialized to get the minimal average query cost?
- If we are willing to tolerate an X% degradation in average query cost from a fully materialized data cube, how much space can we save over the fully materialized data cube?

Dependence relation

The dependence relation on queries:

- Q1 $\preceq$ Q2 iff Q1 can be answered using only the results of query Q2 (Q1 is dependent on Q2).

  In which

  - $\preceq$ is a partial order, and
  - There is a top element, a view upon which is dependent (base cuboid)
- Example:
  - (*part*) $\preceq$ (*part, customer*)
  - (*part*) $\npreceq$ (*customer*) and (*customer*) $\npreceq$ (*part*)

The linear cost model

- For $\langle L, \preceq \rangle$, $Q \preceq Q_A$, $C(Q)$ is the number of rows in the table for that query Q_A used to compute Q
  - This linear relationship can be expressed as:

$$T = m * S + c$$

(m: time/size ratio; c: query overhead; S: size of the view)

  - Validation of the model using TPC-D data:

Source	Size	Time (sec.)	Ratio
From cell itself	1	2.07	not applicable
From view (supplier)	10,000	2.38	.000031
From view (part, supplier)	800,000	20.77	.000023
From view (part, supplier, customer)	6,000,000	226.23	.000037

Growth of query response time with size of view

The benefit of a materialized view

- Denote the benefit of a materialized view *v*, relative to some set of views *S*, as *B(v, S)*
- For each $w \preceq v$, define B_W by:
  - Let *C(v)* be the cost of view *v*
  - Let *u* be the view of least cost in *S* such that $w \preceq u$ (such u must exist)
  - $B_W = C(u) - C(v)$ if $C(v) < C(u)$
    $= 0$ if $C(v) \geq C(u)$
  - B_W is the benefit that it can obtain from *v*
- Define $B(v, S) = \Sigma_{w \leq v} B_W$ which means how *v* can improve the cost of evaluating views, including itself

The greedy algorithm

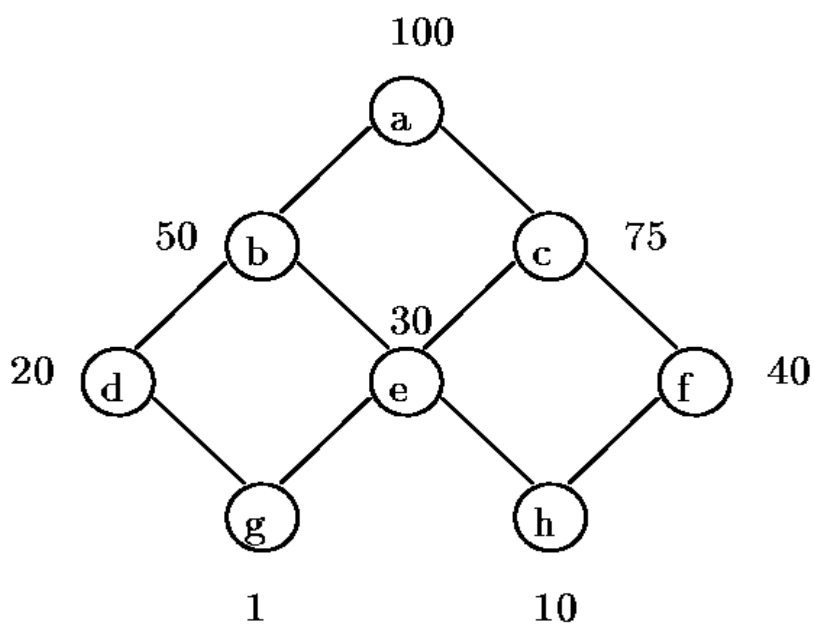
- Objective
  - Assume materializing a fixed number of views, regardless of the space they use
  - How to minimize the average time taken to evaluate a view?
- The greedy algorithm for materializing a set of k views

```
S = {top view};
for i=1 to k do begin
    select that view v not in S such that B(v,S) is maximized;
    S = S union {v};
end;
resulting S is the greedy selection;
```

- Performance: Greedy/Optimal $\geq 1 - (1 - 1/k)^k \geq (e - 1) / e$

Greedy algorithm: example 1

- Suppose we want to choose three views (k = 3)



Example lattice with space costs

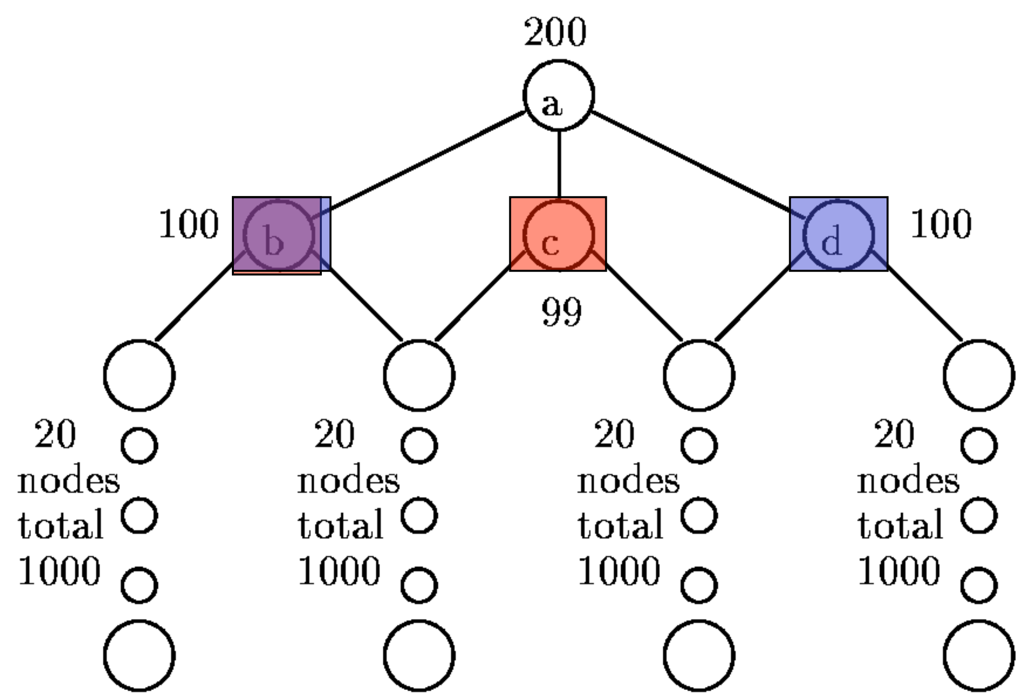
	First Choice	Second Choice	Third Choice
b	$\boxed{50 \times 5 = 250}$		
c	$25 \times 5 = 125$	$25 \times 2 = 50$	$25 \times 1 = 25$
d	$80 \times 2 = 160$	$30 \times 2 = 60$	$\boxed{30 \times 2 = 60}$
e	$70 \times 3 = 210$	$20 \times 3 = 60$	$20 + 20 + 10 = 50$
f	$60 \times 2 = 120$	$\boxed{60 + 10 = 70}$	
g	$99 \times 1 = 99$	$49 \times 1 = 49$	$49 \times 1 = 49$
h	$90 \times 1 = 90$	$40 \times 1 = 40$	$30 \times 1 = 30$

Benefits of possible choices at each round

- The selection is optimal (reduce cost from 800 to 420)

Greedy algorithm: example 2

- Suppose k = 2
  - Greedy algorithm picks c and b: benefit = 101*41+100*21 = 6241
  - Optimal selection is b and d: benefit = 100*41+100*41 = 8200
  - However, greedy/optimal = 6241/8200 > 3/4



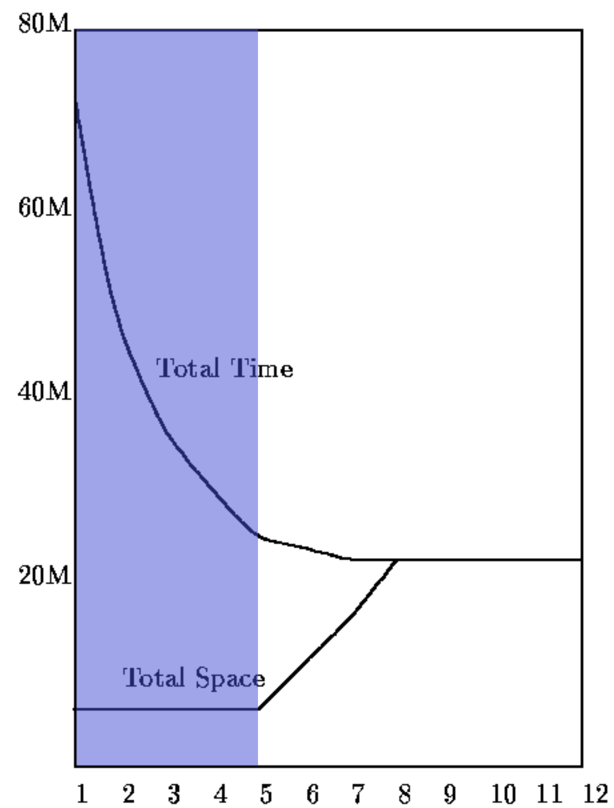
A lattice where the greedy algorithm does poorly

An experiment: how many views should be materialized?

- Time and space for the greedy selection for the TPC-D-based example (full materialization is not efficient)

Number	Selection	Benefit	Total Time	Total Space
1.	*cp*	infinite	72M	6M
2.	*ns*	24M	48M	6M
3.	*nt*	12M	36M	6M
4.	*c*	5.9M	30.1M	6.1M
5.	*p*	5.8M	24.3M	6.3M
6.	*cs*	1M	23.3M	11.3M
7.	*np*	1M	22.3M	16.3M
8.	*ct*	0.01M	22.3M	22.3M
9.	*t*	small	22.3M	22.3M
10	*n*	small	22.3M	22.3M
11.	*s*	small	22.3M	22.3M
12.	none	small	22.3M	22.3M

Greedy order of view selection for TPC-D-based example



Number of materialized views

Indexing OLAP Data: Bitmap Index

- 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

Base table

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

Index on Region

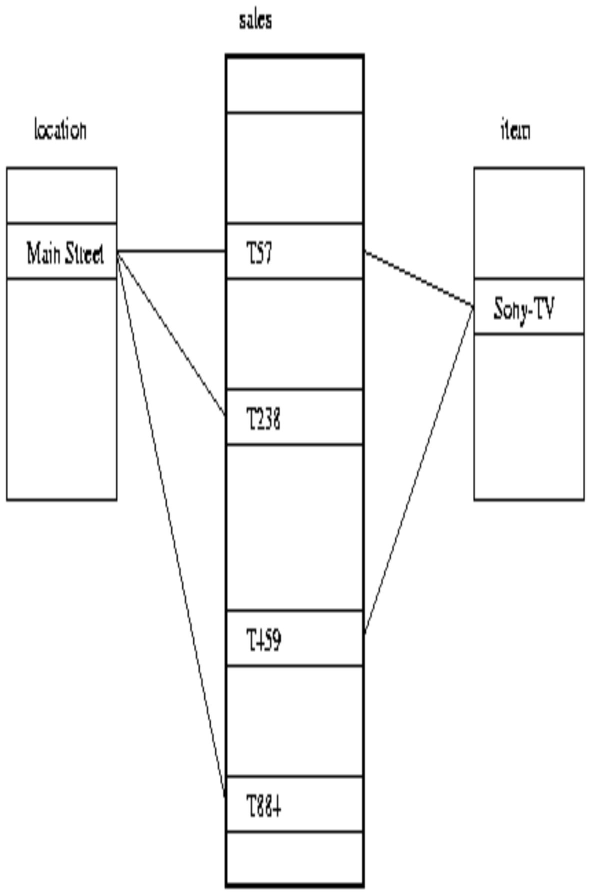
RecID	Asia	Europe	America
1	1	0	0
2	0	1	0
3	1	0	0
4	0	0	1
5	0	1	0

Index on Type

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

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 — a rather costly operation
- 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



Summary

- Data warehouse
  - A subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management's decision-making process
- A multi-dimensional model of a data warehouse
  - Star schema, snowflake schema, fact constellations
  - A data cube consists of dimensions & measures
- OLAP operations: drilling, rolling, slicing, dicing and pivoting
- OLAP servers: ROLAP, MOLAP, HOLAP
- Efficient computation of data cubes
  - Partial vs. full vs. no materialization
  - Multiway array aggregation
  - Bitmap index and join index implementations
- Further development of data cube technology
  - Discovery-drive and multi-feature cubes
  - From OLAP to OLAM (on-line analytical mining)