Business Intelligence: OLAP, Data Warehousing, Materialized views

ΠΜΣ "Ερευνητικές Κατευθύνσεις στην Πληροφορική"

Επεξεργασία και Ανάλυση Δεδομένων

SPRING SEMESTER 2020

Why we still study OLAP/Data Warehouse in Big Data?

- Understand the Big Data history
  - How does the requirement of (big) data analytics/business intelligence evolve over the time?
  - What are the architecture and implementation techniques being developed? Will they still be useful in Big Data?
  - Understand their limitation and what factors have changed from 90's to now?
- NoSQL is not only SQL☺
- Hive/Impala aims to provide OLAP/BI for Big Data using Hadoop

Highlights

- OLAP
  - Multi-relational Data model
  - Operators
  - SQL
- Data warehouse (architecture, issues, optimizations)
- Materialized view maintenance

Let’s get back to the root in 70’s: Relational Database

Basic Structure

- Formally, given sets $D_1, D_2, \dots. D_n$ a **relation** r is a subset of

  $D_1 \times D_2 \times \dots \times D_n$

  Thus, a relation is a set of n-tuples $(a_1, a_2, \dots, a_n)$ where each $a_i \in D_i$
- Example:

  customer_name = {Jones, Smith, Curry, Lindsay}

  customer_street = {Main, North, Park}

  customer_city = {Harrison, Rye, Pittsfield}

  Then r = { (Jones, Main, Harrison),

  (Smith, North, Rye),

  (Curry, North, Rye),

  (Lindsay, Park, Pittsfield) }

  is a relation over

  customer_name , customer_street, customer_city

Relation Schema

- $A_1, A_2, \ldots, A_n$ are *attributes*

- $R = (A_1, A_2, \ldots, A_n)$ is a *relation schema*

  Example:

  Customer_schema = (*customer_name, customer_street, customer_city*)

- *r*(*R*) is a *relation* on the *relation schema R*

  Example:

  customer (Customer_schema)

Relation Instance

- The current values (*relation instance*) of a relation are specified by a table
- An element *t* of *r* is a *tuple*, represented by a *row* in a table

attributes
(or columns)

customer_name	*customer_street*	customer_city
Jones	Main	Harrison
Smith	North	Rye
Curry	North	Rye
Lindsay	Park	Pittsfield

tuples
(or rows)

customer

Database

- A database consists of multiple relations
- Information about an enterprise is broken up into parts, with each relation storing one part of the information

  account : stores information about accounts
  depositor : stores information about which customer owns which account
  customer : stores information about customers

- Storing all information as a single relation such as
  bank(*account_number, balance, customer_name*, ..)
  results in repetition of information (e.g., two customers own an account) and the need for null values (e.g., represent a customer without an account)

Banking Example

branch (branch-name, branch-city, assets)

customer (customer-name, customer-street, customer-city)

account (account-number, branch-name, balance)

loan (loan-number, branch-name, amount)

depositor (customer-name, account-number)

borrower (customer-name, loan-number)

Relational Algebra

- Primitives
  - Projection (π)
  - Selection (σ)
  - Cartesian product ($\times$)
  - Set union ($\cup$)
  - Set difference ($-$)
  - Rename (ρ)
- Other operations
  - Join ($\bowtie$)
  - Group by… aggregation
  - …

What happens next?

- SQL
- System R (DB2), INGRES, ORACLE, SQL-Server, Teradata
  - B+-Tree (select)
  - Transaction Management
  - Join algorithm

In early 90’s:
OLAP & Data Warehouse

Database Workloads

- OLTP (online transaction processing)
  - Typical applications: e-commerce, banking, airline reservations
  - User facing: real-time, low latency, highly-concurrent
  - Tasks: relatively small set of “standard” transactional queries
  - Data access pattern: random reads, updates, writes (involving relatively small amounts of data)
- OLAP (online analytical processing)
  - Typical applications: business intelligence, data mining
  - Back-end processing: batch workloads, less concurrency
  - Tasks: complex analytical queries, often ad hoc
  - Data access pattern: table scans, large amounts of data involved per query

OLTP

- Most database operations involve *On-Line Transaction Processing* (OTLP).
  - Short, simple, frequent queries and/or modifications, each involving a small number of tuples.
  - Examples: Answering queries from a Web interface, sales at cash registers, selling airline tickets.

OLAP

- Of increasing importance are *On-Line Application Processing* (OLAP) queries.
  - Few, but complex queries --- may run for hours.
  - Queries do not depend on having an absolutely up-to-date database.

OLAP Examples

1. Amazon analyzes purchases by its customers to come up with an individual screen with products of likely interest to the customer.
2. Analysts at Wal-Mart look for items with increasing sales in some region.

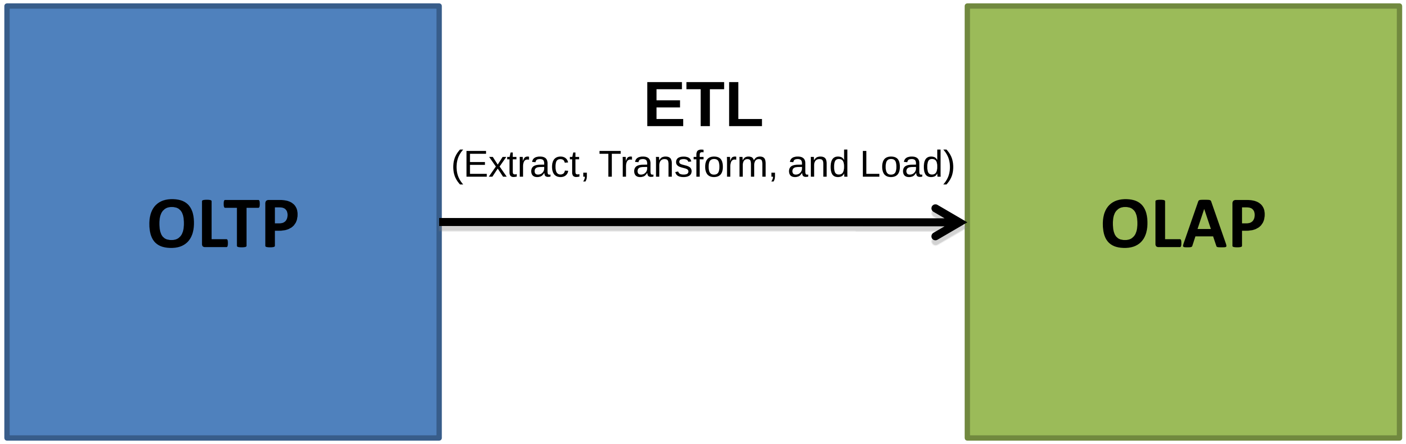
OLTP vs. OLAP

	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

One Database or Two?

- Downsides of co-existing OLTP and OLAP workloads
  - Poor memory management
  - Conflicting data access patterns
  - Variable latency
- Solution: separate databases
  - User-facing OLTP database for high-volume transactions
  - Data warehouse for OLAP workloads
  - How do we connect the two?

OLTP/OLAP Architecture



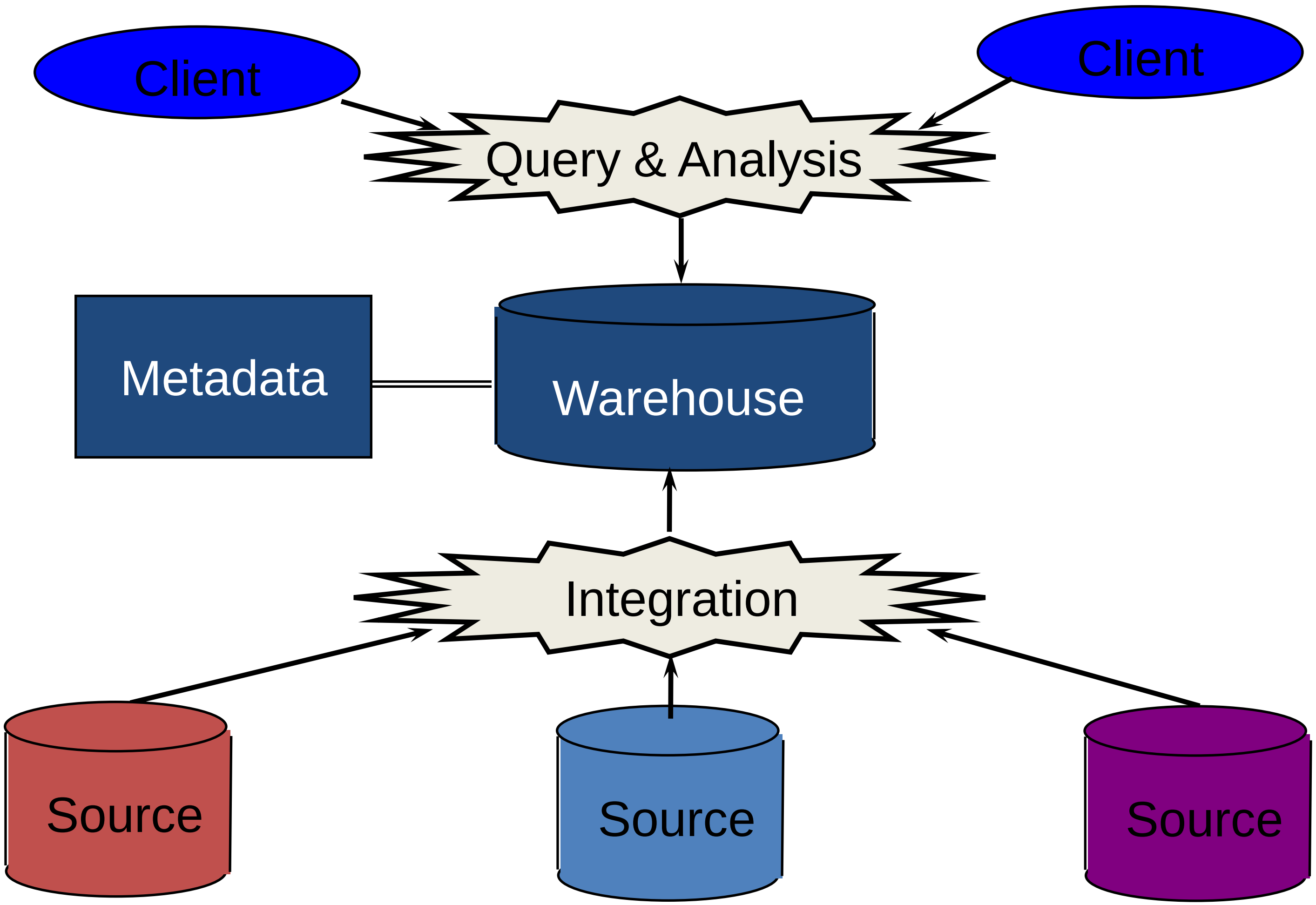
OLTP/OLAP Integration

- OLTP database for user-facing transactions
  - Retain records of all activity
  - Periodic ETL (e.g., nightly)
- Extract-Transform-Load (ETL)
  - Extract records from source
  - Transform: clean data, check integrity, aggregate, etc.
  - Load into OLAP database
- OLAP database for data warehousing
  - Business intelligence: reporting, ad hoc queries, data mining, etc.
  - Feedback to improve OLTP services

The Data Warehouse

- The most common form of data integration.
  - Copy sources into a single DB (*warehouse*) and try to keep it up-to-date.
  - Usual method: periodic reconstruction of the warehouse, perhaps overnight.
  - Frequently essential for analytic queries.

Warehouse Architecture



Star Schemas

- A *star schema* is a common organization for data at a warehouse. It consists of:
  1. *Fact table* : a very large accumulation of facts such as sales.
     - Often "insert-only."
  2. *Dimension tables* : smaller, generally static information about the entities involved in the facts.

Example: Star Schema

- Suppose we want to record in a warehouse information about every beer sale: the bar, the brand of beer, the drinker who bought the beer, the day, the time, and the price charged.
- The fact table is a relation:

Sales(bar, beer, drinker, day, time, price)

Example, Continued

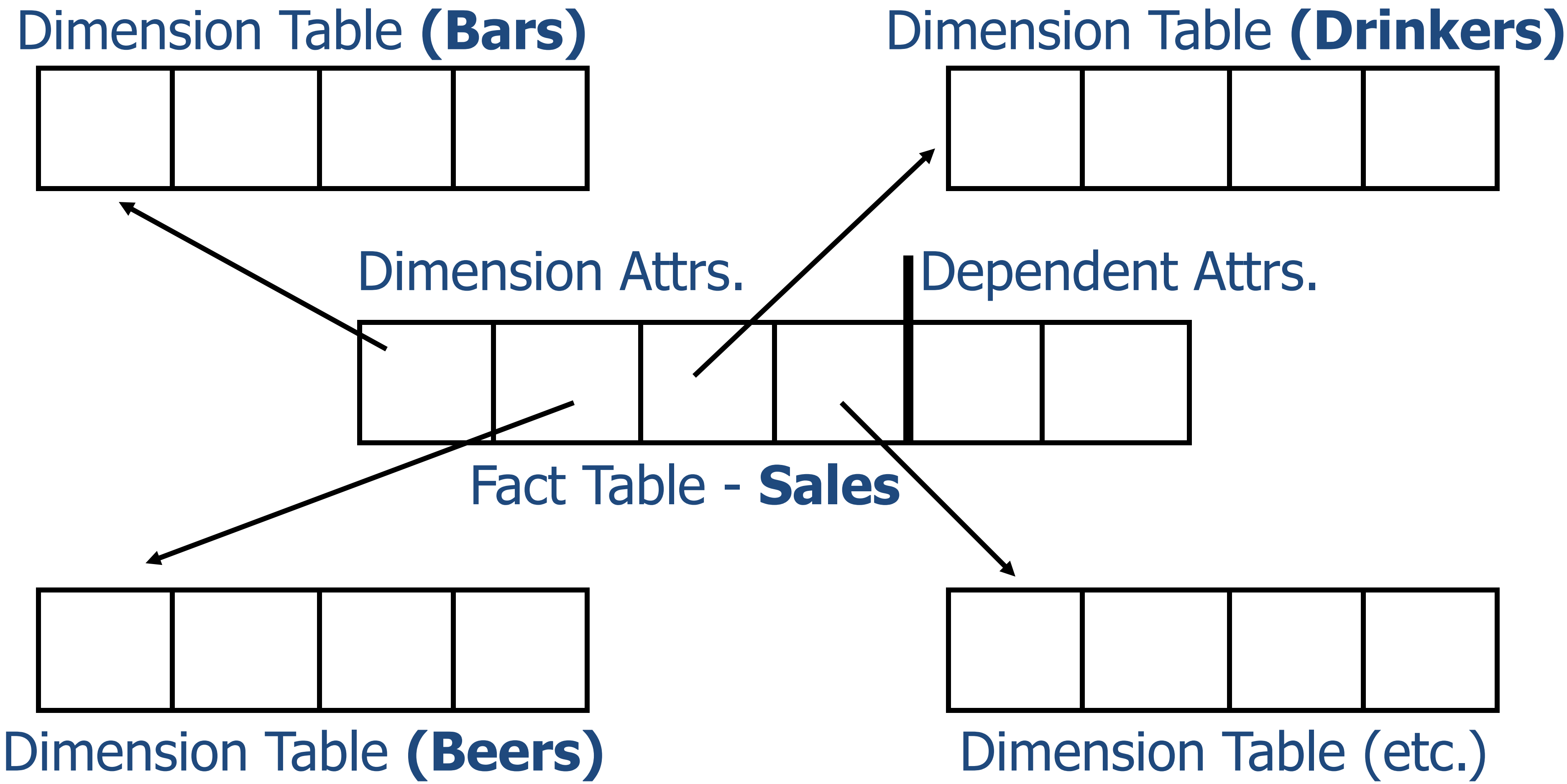
- The dimension tables include information about the bar, beer, and drinker "dimensions":

  Bars(bar, addr, license)

  Beers(beer, manf)

  Drinkers(drinker, addr, phone)

Visualization – Star Schema



Dimensions and Dependent Attributes

- Two classes of fact-table attributes:
  1. *Dimension attributes* : the key of a dimension table.
  2. *Dependent attributes* : a value determined by the dimension attributes of the tuple.

Warehouse Models & Operators

- Data Models
  - relations
  - stars & snowflakes
  - cubes
- Operators
  - slice & dice
  - roll-up, drill down
  - pivoting
  - other

Star

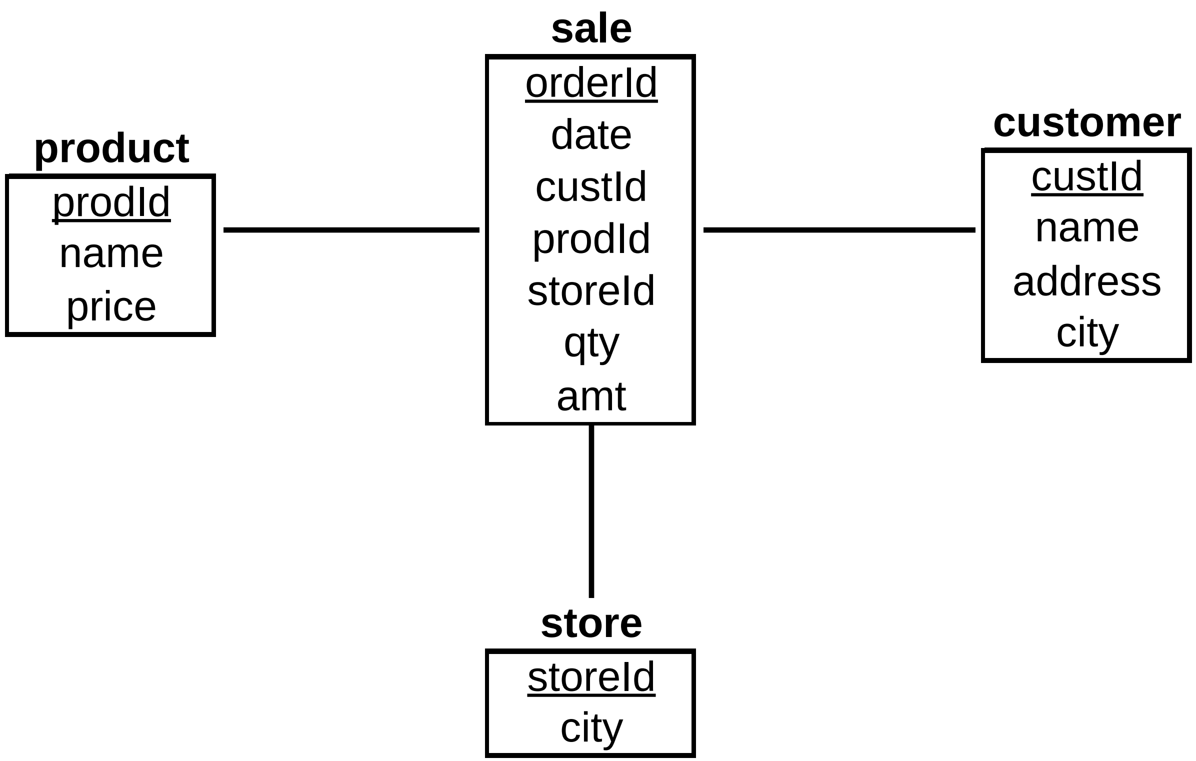
product	prodId	name	price
	p1	bolt	10
	p2	nut	5

store	storeId	city
	c1	nyc
	c2	sfo
	c3	la

sale	oderId	date	custId	prodId	storeId	qty	amt
	o100	1/7/97	53	p1	c1	1	12
	o102	2/7/97	53	p2	c1	2	11
	105	3/8/97	111	p1	c3	5	50

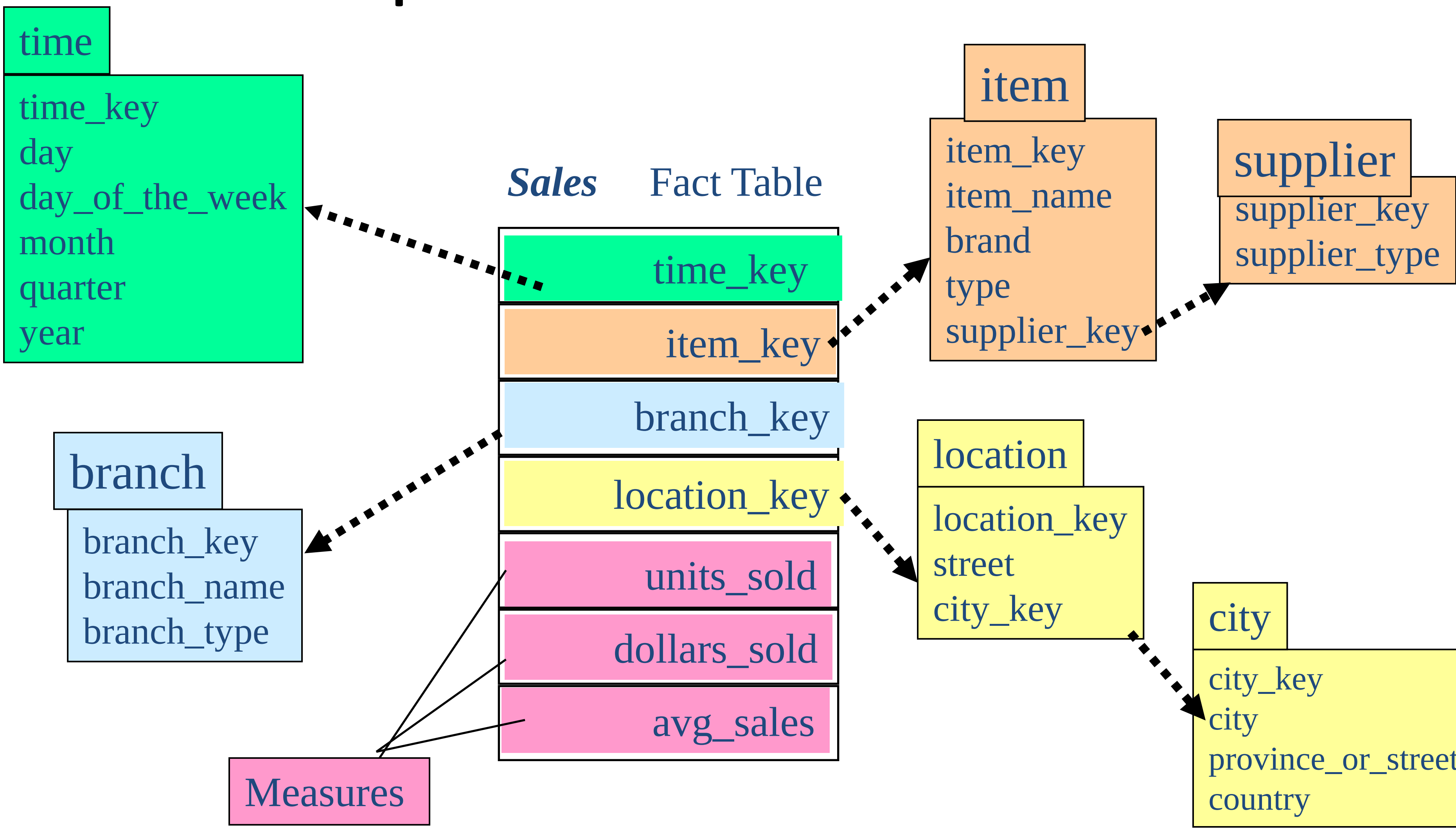
customer	custId	name	address	city
	53	joe	10 main	sfo
	81	fred	12 main	sfo
	111	sally	80 willow	la

Star Schema

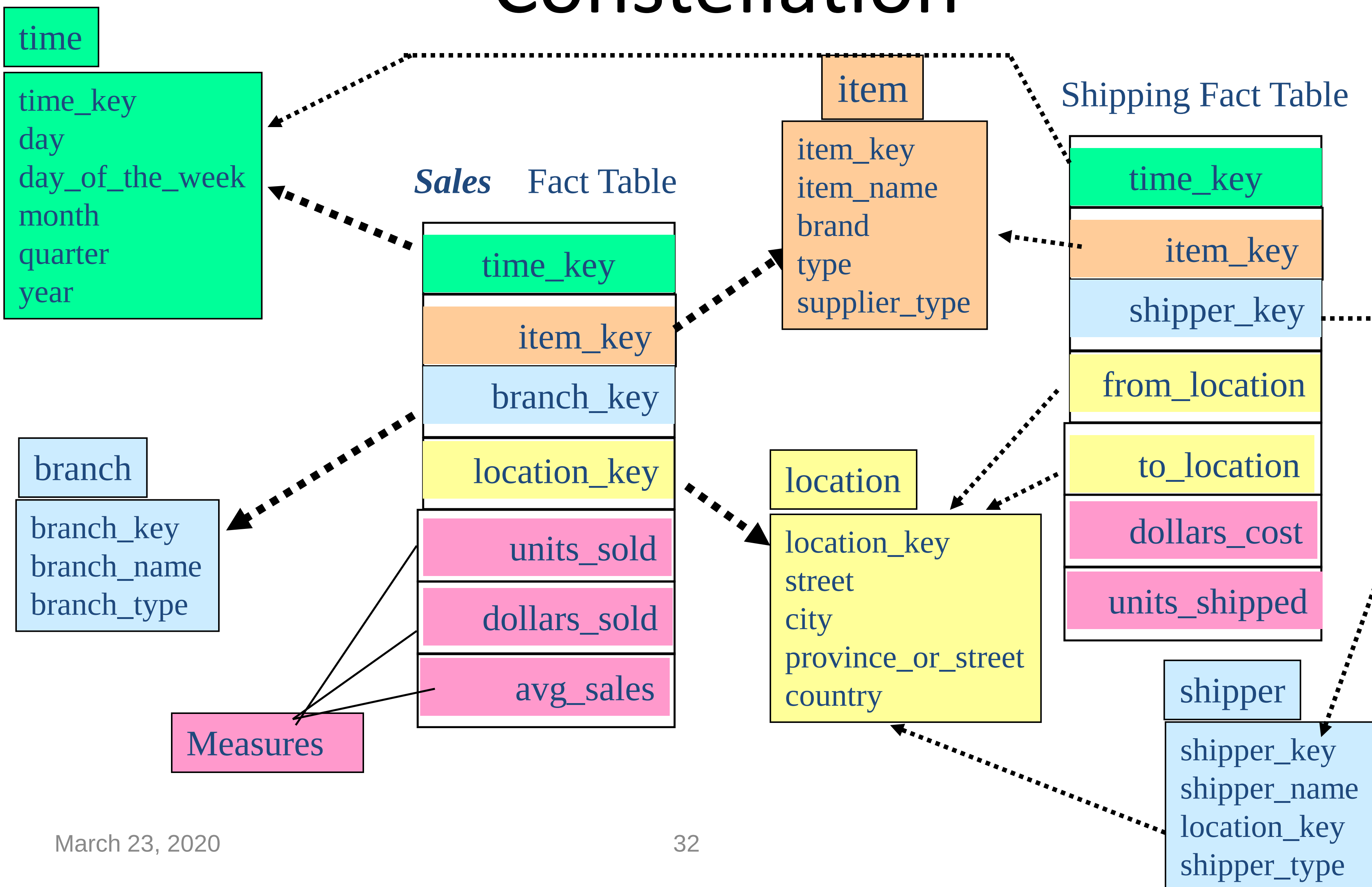


sale
orderId
date
custId
prodId
storeId
qty
amt
product
prodId
name
price
customer
custId
name
address
city
store
storeId
city

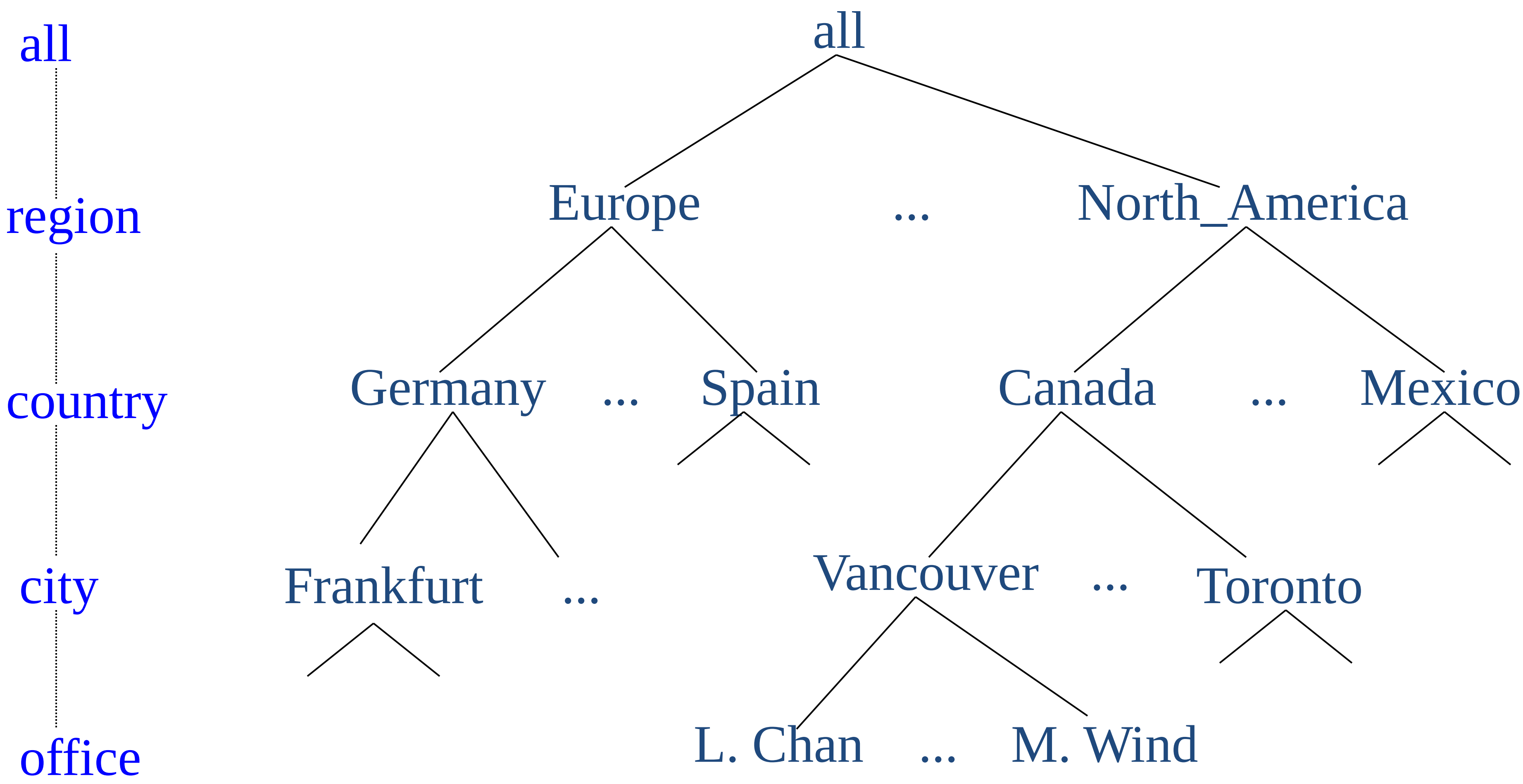
Example of Snowflake Schema



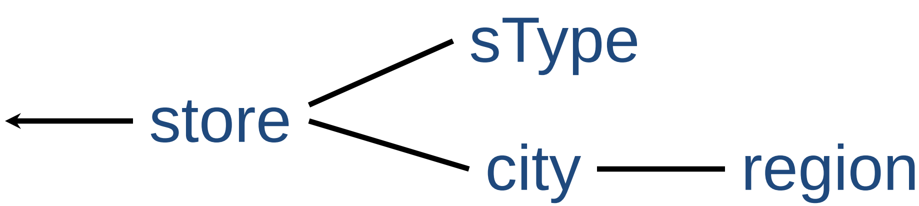
Example of Fact Constellation



A Concept Hierarchy: Dimension (location)



Dimension Hierarchies



store	storeId	cityId	tId	mgr
	s5	sfo	t1	joe
	s7	sfo	t2	fred
	s9	la	t1	nancy

sType	tId	size	location
	t1	small	downtown
	t2	large	suburbs

city	cityId	pop	regId
	sfo	1M	north
	la	5M	south

region	regId	name
	north	cold region
	south	warm region

➔ snowflake schema

➔ constellations

Aggregates

- Add up amounts for day 1
- In SQL: SELECT sum(amt) FROM SALE
  WHERE date = 1

sale	prodId	storeId	date	amt
	p1	c1	1	12
	p2	c1	1	11
	p1	c3	1	50
	p2	c2	1	8
	p1	c1	2	44
	p1	c2	2	4

⇒ 81

Aggregates

- Add up amounts by day
- In SQL: SELECT date, sum(amt) FROM SALE
  GROUP BY date

sale	prodId	storeId	date	amt
	p1	c1	1	12
	p2	c1	1	11
	p1	c3	1	50
	p2	c2	1	8
	p1	c1	2	44
	p1	c2	2	4

ans	date	sum
	1	81
	2	48

Another Example

- Add up amounts by day, product
- In SQL: SELECT date, sum(amt) FROM SALE GROUP BY date, prodId

sale	prodId	storeId	date	amt
	p1	c1	1	12
	p2	c1	1	11
	p1	c3	1	50
	p2	c2	1	8
	p1	c1	2	44
	p1	c2	2	4

sale	prodId	date	amt
	p1	1	62
	p2	1	19
	p1	2	48

rollup →

← drill-down

ROLAP vs. MOLAP

- ROLAP:
  Relational On-Line Analytical Processing
- MOLAP:
  Multi-Dimensional On-Line Analytical Processing

Cube

Fact table view:

sale	prodId	storeId	amt
	p1	c1	12
	p2	c1	11
	p1	c3	50
	p2	c2	8

↔

Multi-dimensional cube:

	c1	c2	c3
p1	12		50
p2	11	8	

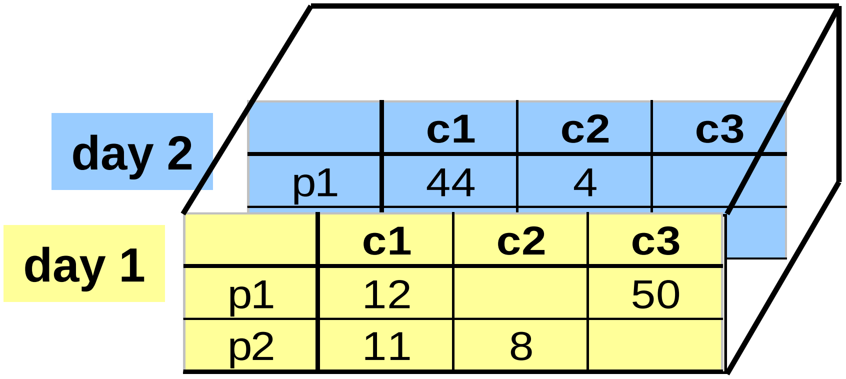
dimensions = 2

3-D Cube

Fact table view:

sale	prodId	storeId	date	amt
	p1	c1	1	12
	p2	c1	1	11
	p1	c3	1	50
	p2	c2	1	8
	p1	c1	2	44
	p1	c2	2	4

Multi-dimensional cube:

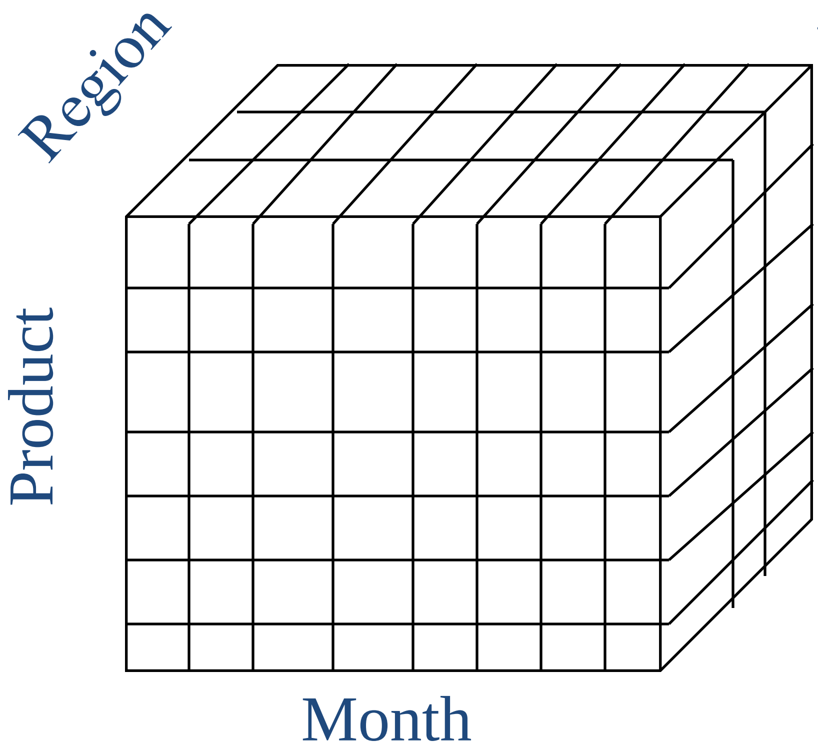


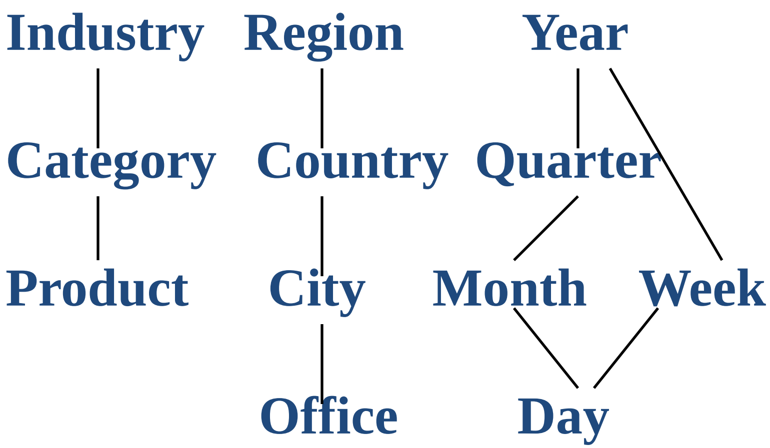
dimensions = 3

Multidimensional Data

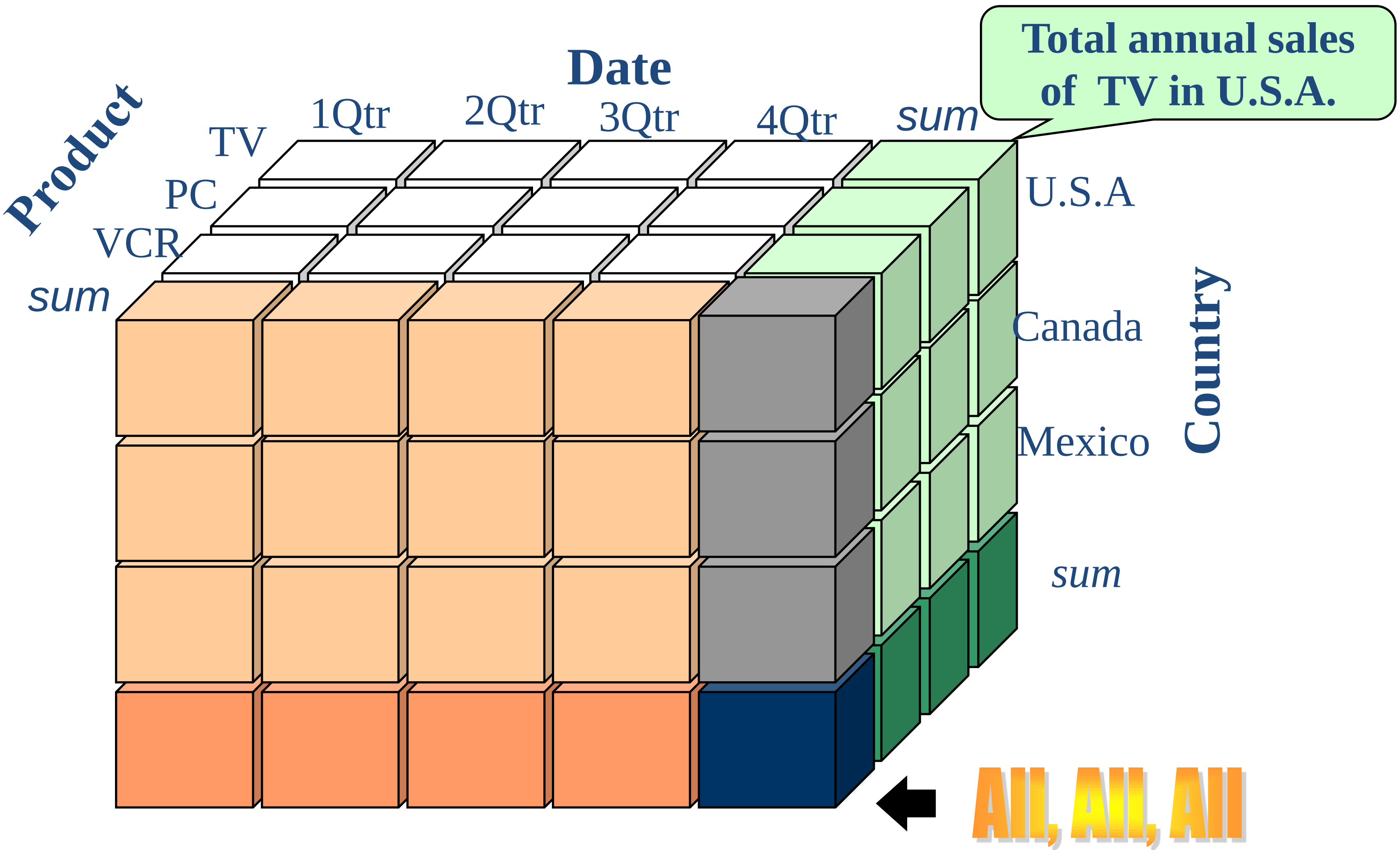
- Sales volume as a function of product, month, and region

Dimensions: Product, Location, Time
Hierarchical summarization paths

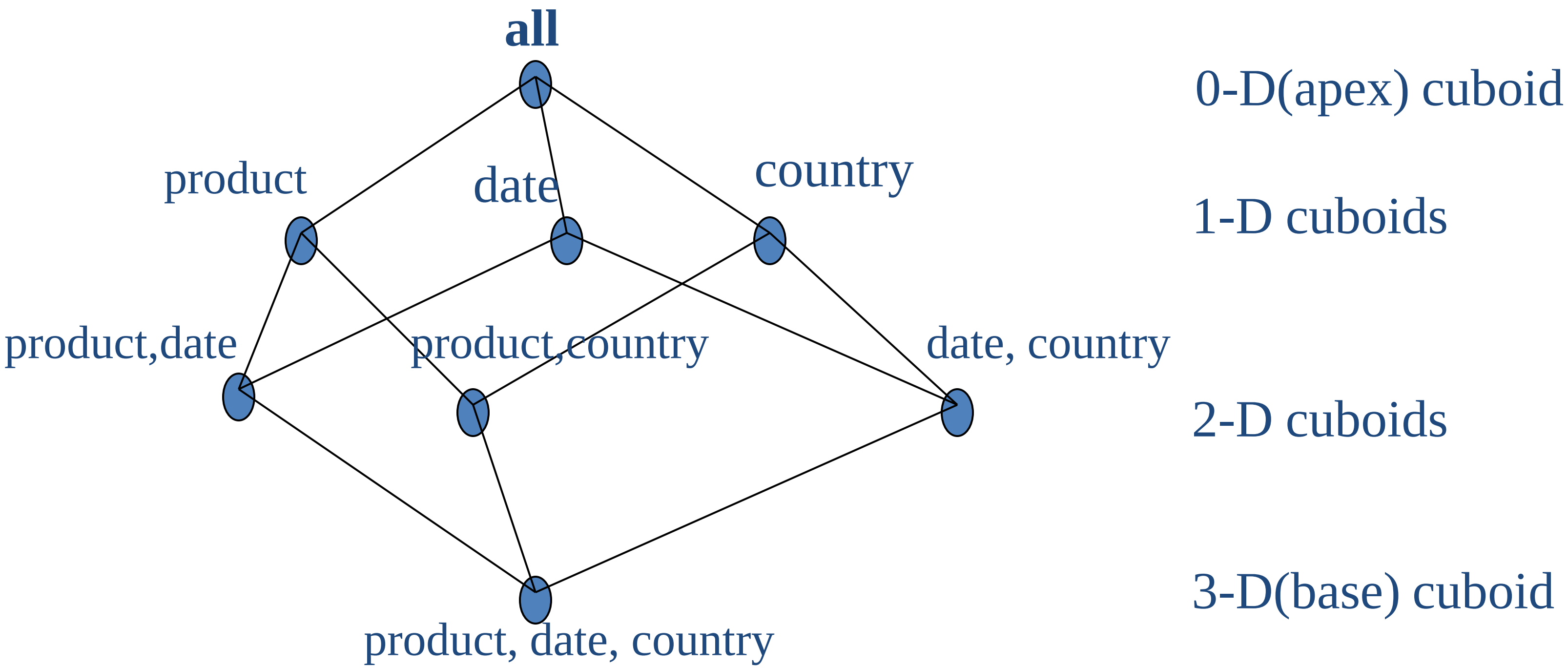




A Sample Data Cube



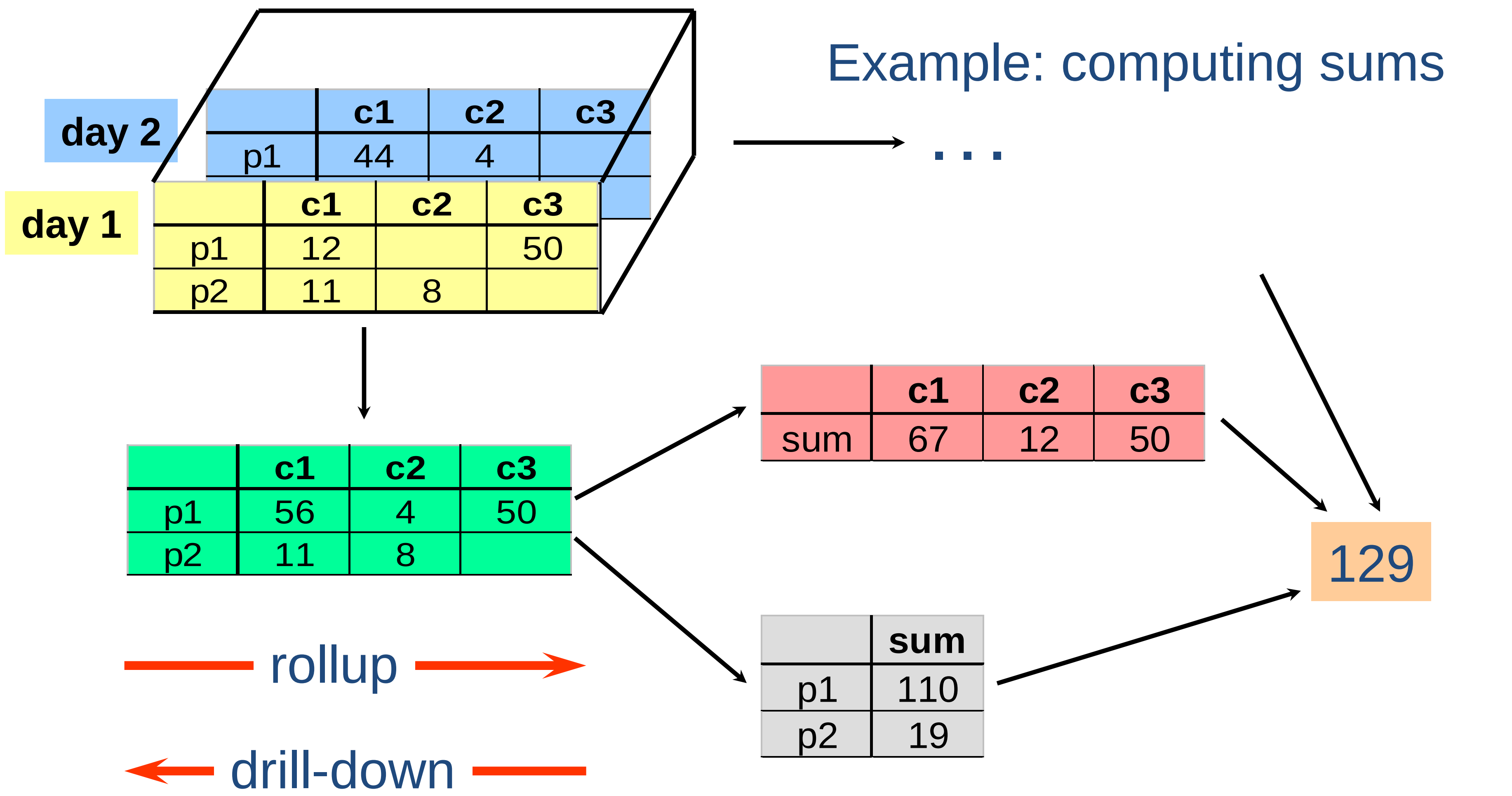
Cuboids Corresponding to the Cube



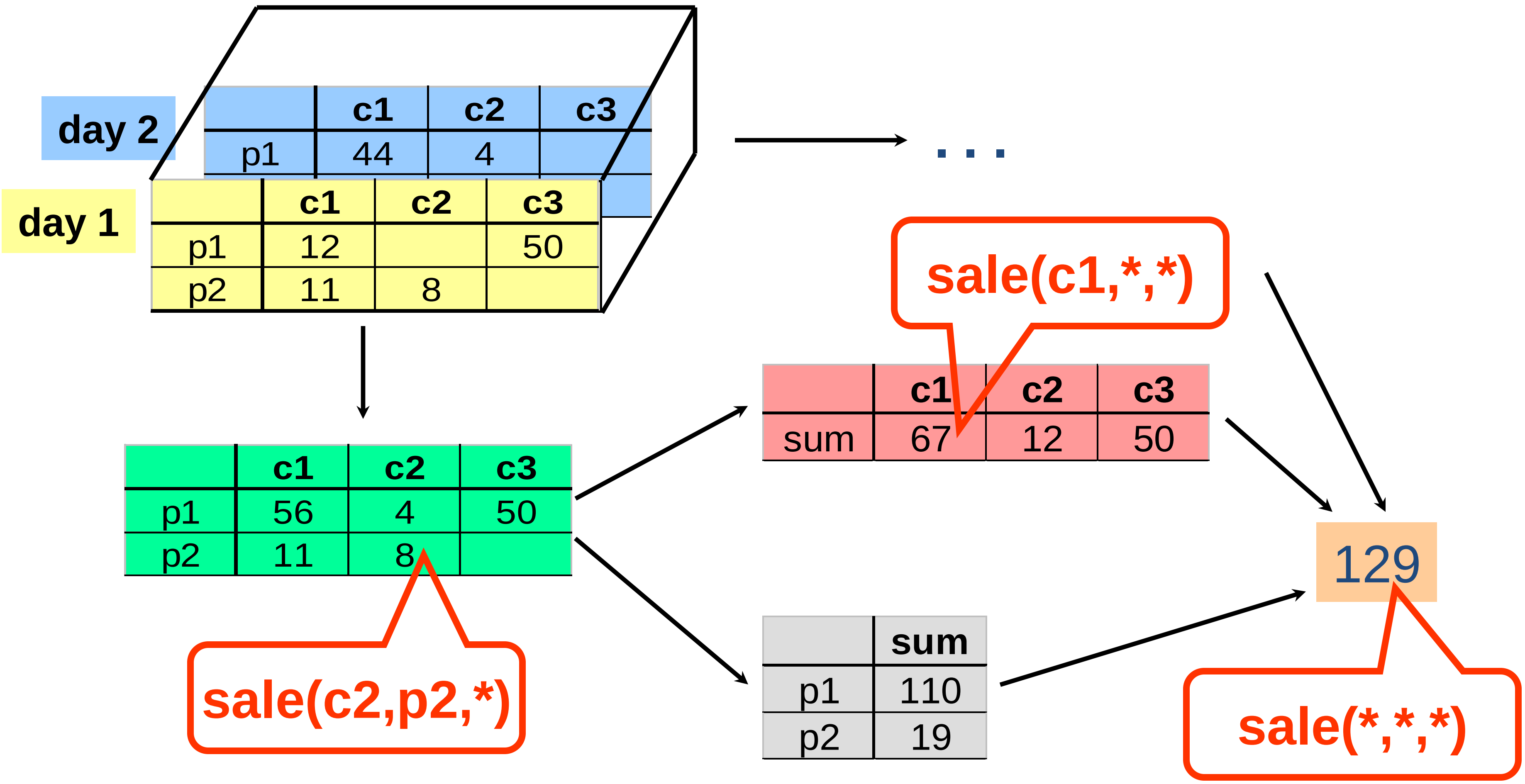
Typical OLAP Operations

- 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)*

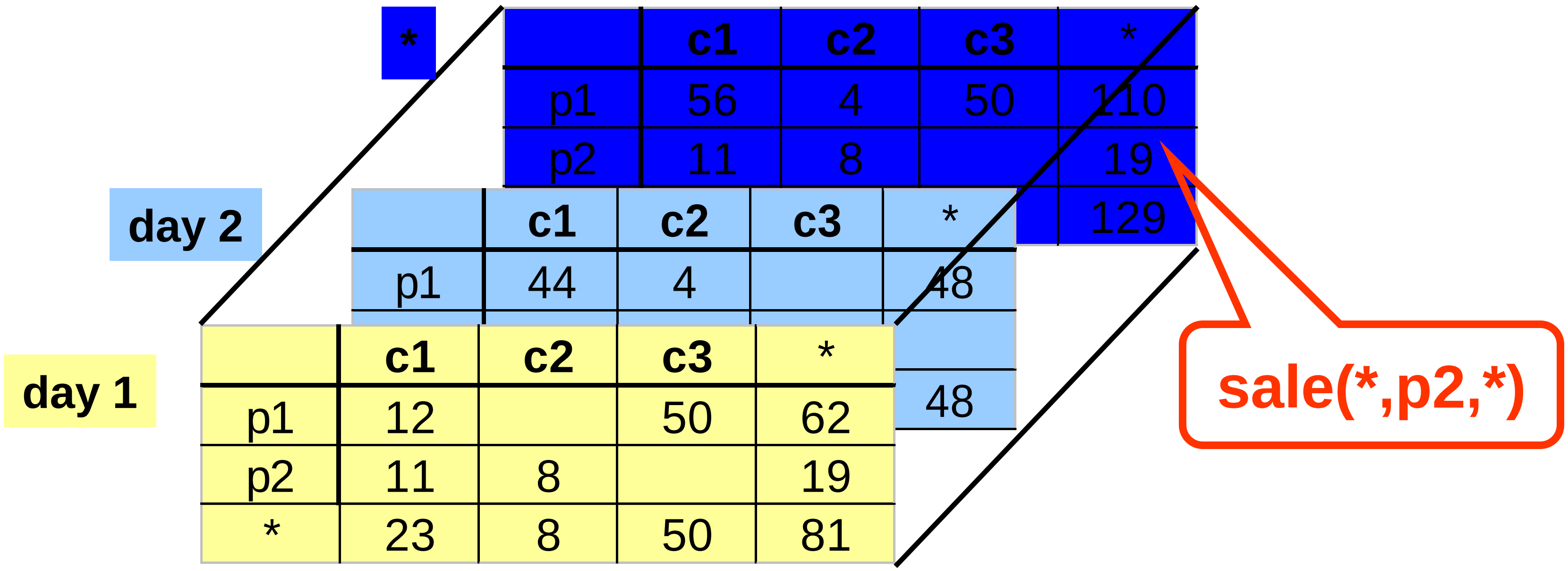
Cube Aggregation



Cube Operators



Extended Cube



Aggregation Using Hierarchies

day 2

	c1	**c2**	**c3**
p1	44	4	

day 1

	c1	**c2**	**c3**
p1	12		50
p2	11	8	

	region A	**region B**
p1	56	54
p2	11	8

customer

region

country

(customer c1 in Region A;
customers c2, c3 in Region B)

Pivoting

Fact table view:

sale	prodId	storeId	date	amt
	p1	c1	1	12
	p2	c1	1	11
	p1	c3	1	50
	p2	c2	1	8
	p1	c1	2	44
	p1	c2	2	4

Multi-dimensional cube:

day 2

	c1	c2	c3
p1	44	4	

day 1

	c1	c2	c3
p1	12		50
p2	11	8	

	c1	c2	c3
p1	56	4	50
p2	11	8	

CUBE Operator (SQL-99)

Chevy Sales Cross Tab				
Chevy	*1990*	*1991*	*1992*	*Total (ALL)*
black	50	85	154	289
white	40	115	199	354
Total (ALL)	90	200	353	1286

SELECT model, year, color, sum(sales) as sales

FROM sales

WHERE model in ('Chevy')

AND year BETWEEN 1990 AND 1992

GROUP BY CUBE (model, year, color);

CUBE Contd.

```
SELECT      model, year, color, sum(sales) as sales
FROM        sales
WHERE       model in ('Chevy')
AND         year BETWEEN 1990 AND 1992
GROUP BY    CUBE (model, year, color);
```

- Computes union of 8 different groupings:
  - {(model, year, color), (model, year), (model, color), (year, color), (model), (year), (color), ()}

Example Contd.

SALES			
Model	Year	Color	Sales
Chevy	1990	red	5
Chevy	1990	white	87
Chevy	1990	blue	62
Chevy	1991	red	54
Chevy	1991	white	95
Chevy	1991	blue	49
Chevy	1992	red	31
Chevy	1992	white	54
Chevy	1992	blue	71
Ford	1990	red	64
Ford	1990	white	62
Ford	1990	blue	63
Ford	1991	red	52
Ford	1991	white	9
Ford	1991	blue	55
Ford	1992	red	27
Ford	1992	white	62
Ford	1992	blue	39

CUBE

DATA CUBE			
Model	Year	Color	Sales
ALL	ALL	ALL	942
chevy	ALL	ALL	510
ford	ALL	ALL	432
ALL	1990	ALL	343
ALL	1991	ALL	314
ALL	1992	ALL	285
ALL	ALL	red	165
ALL	ALL	white	273
ALL	ALL	blue	339
chevy	1990	ALL	154
chevy	1991	ALL	199
chevy	1992	ALL	157
ford	1990	ALL	189
ford	1991	ALL	116
ford	1992	ALL	128
chevy	ALL	red	91
chevy	ALL	white	236
chevy	ALL	blue	183
ford	ALL	red	144
ford	ALL	white	133
ford	ALL	blue	156
ALL	1990	red	69
ALL	1990	white	149
ALL	1990	blue	125
ALL	1991	red	107
ALL	1991	white	104
ALL	1991	blue	104
ALL	1992	red	59
ALL	1992	white	116
ALL	1992	blue	110

Aggregates

- Operators: sum, count, max, min,
  median, ave
- “Having” clause
- Cube (& Rollup) operator
- Using dimension hierarchy
  - average by region (within store)
  - maximum by month (within date)

Query & Analysis Tools

- Query Building
- Report Writers (comparisons, growth, graphs,…)
- Spreadsheet Systems
- Web Interfaces
- Data Mining

Other Operations

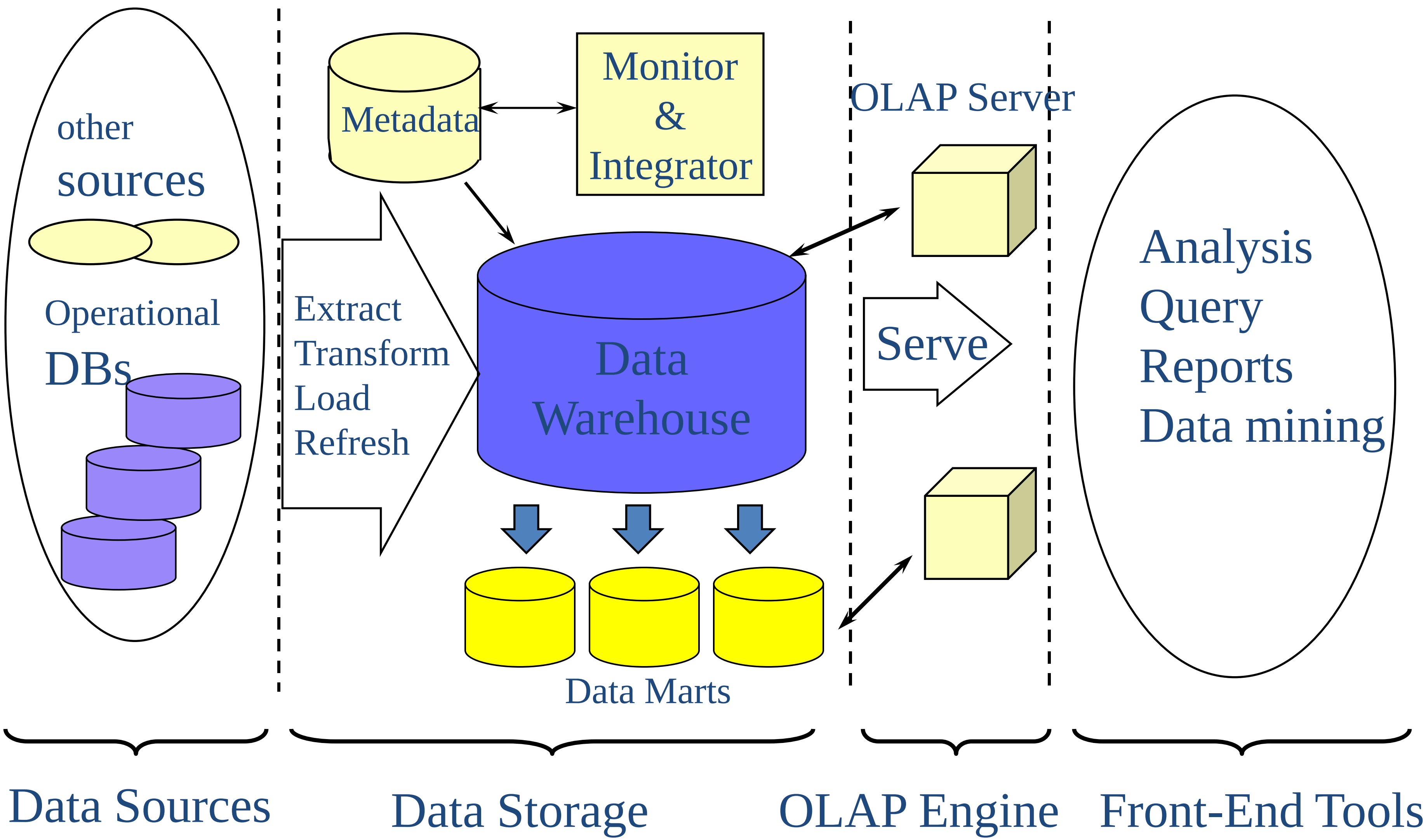
- Time functions
  - e.g., time average
- Computed Attributes
  - e.g., commission = sales * rate
- Text Queries
  - e.g., find documents with words X AND B
  - e.g., rank documents by frequency of words X, Y, Z

Data Warehouse Implementation

Implementing a Warehouse

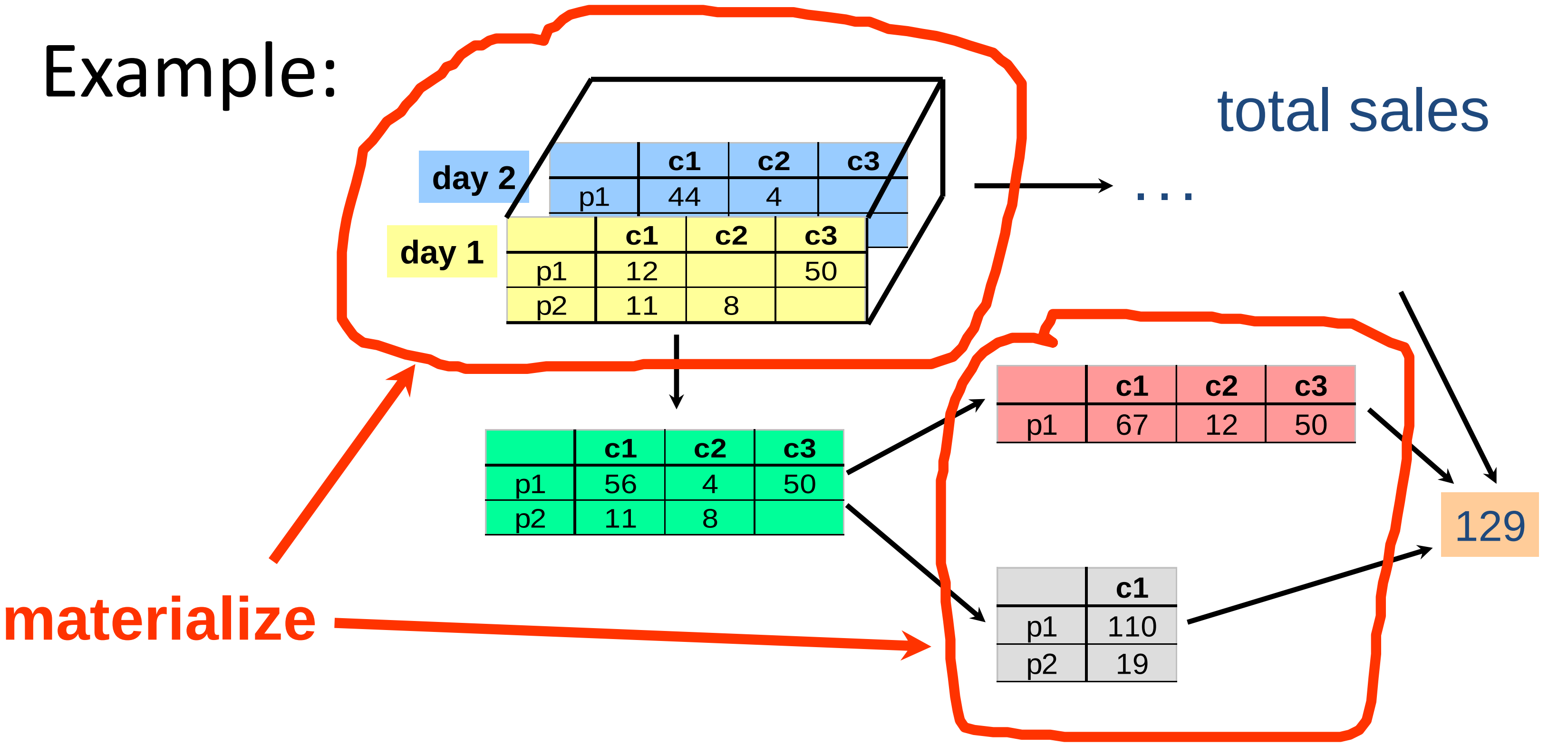
- *Monitoring*: Sending data from sources
- *Integrating*: Loading, cleansing,...
- *Processing*: Query processing, indexing, ...
- *Managing*: Metadata, Design, ...

Multi-Tiered Architecture



What to Materialize?

- Store in warehouse results useful for common queries
- Example:



Materialization Factors

- Type/frequency of queries
- Query response time
- Storage cost
- Update cost

Efficient Data Cube Computation

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

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n$$

Problem: How to Implement Data Cube Efficiently?

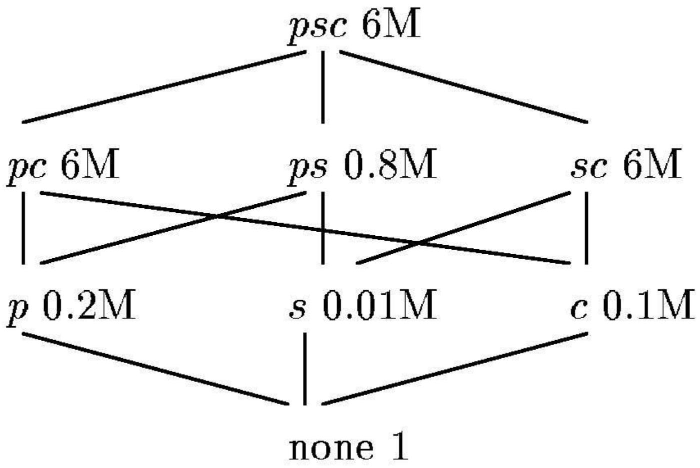
- Physically materialize the whole data cube
  - Space consuming in storage and time consuming in construction
  - Indexing overhead
- Materialize nothing
  - No extra space needed but very slow response times
- Materialize only part of the data cube
  - Intuition: precompute frequently-asked queries?
  - 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?

Static vs. Dynamic view selection

- Static:
  - Query frequencies are static
  - Views are selected from scratch
- Dynamic
  - Existing pool of materialized views
  - Changing query frequencies

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

A 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 $\geq 1 - (1 - 1/k)^{k} \geq (e - 1) / e$