Lecture 2: Data Warehouse and OLAP

Outline

- DSS, Data Warehouse, and OLAP
- Models & operations
  - OLAP operations
  - SQL:1999 Supports
- Implementation Techniques
  - View Materialization
  - Indexing
- Future directions

The Need for Data Analysis

- Managers must be able to track daily transactions to evaluate how the business is performing
- By tapping into the operational database, management can develop strategies to meet organizational goals
- Data analysis can provide information about short-term tactical evaluations and strategies

Decision Support Systems

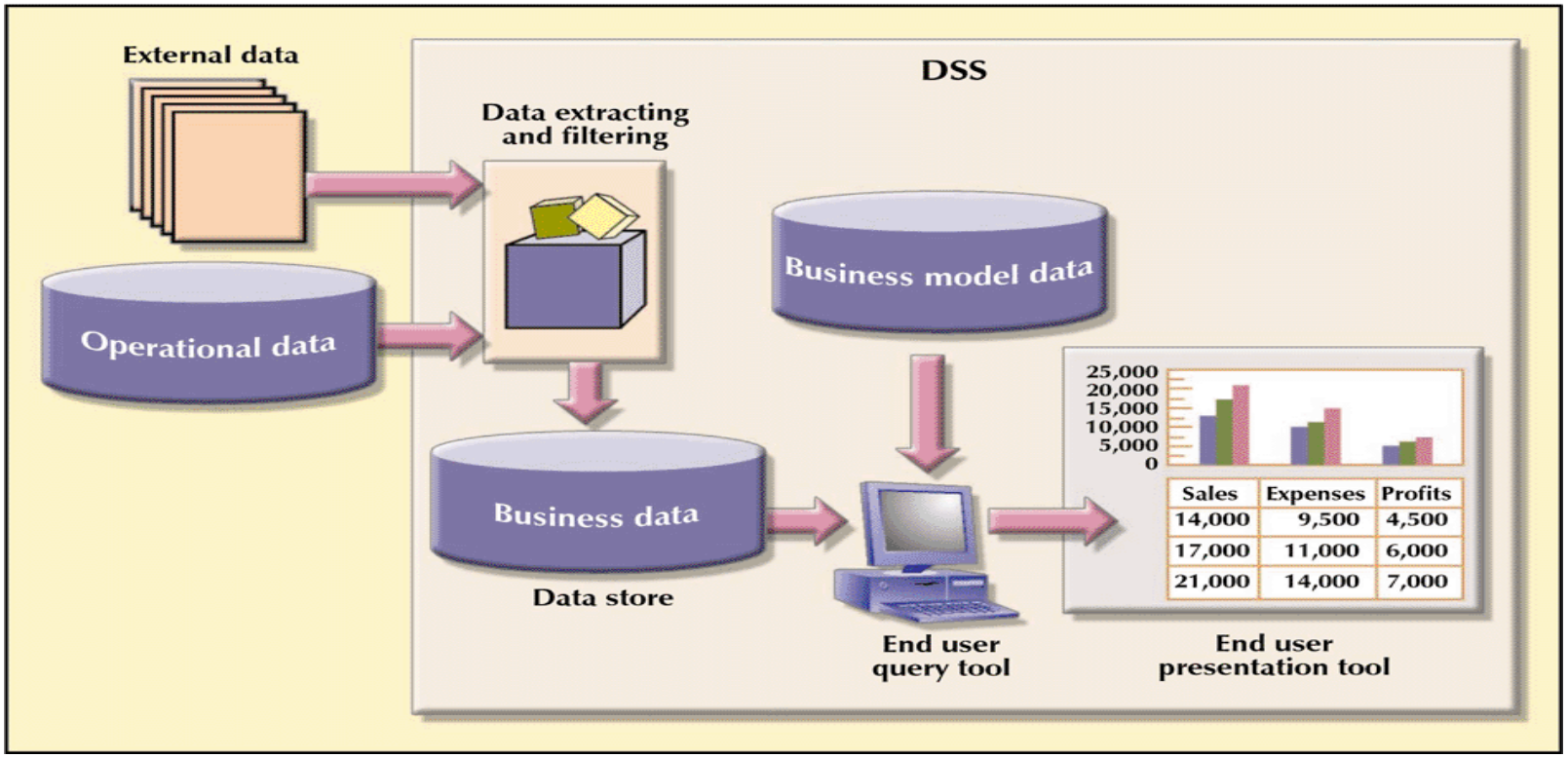
- Methodology (or series of methodologies) designed to extract information from data and to use such information as a basis for decision making
- Decision support system (DSS):
  - Arrangement of computerized tools used to assist managerial decision making within a business
  - Usually requires extensive data "massaging" to produce information
  - Used at all levels within an organization
  - Often tailored to focus on specific business areas
  - Provides ad hoc query tools to retrieve data and to display data in different formats

Decision Support Systems (continued)

- Composed of four main components:
  - Data store component
    - Basically a DSS database
  - Data extraction and filtering component
    - Used to extract and validate data taken from operational database and external data sources
  - End-user query tool
    - Used to create queries that access database
  - End-user presentation tool
    - Used to organize and present data

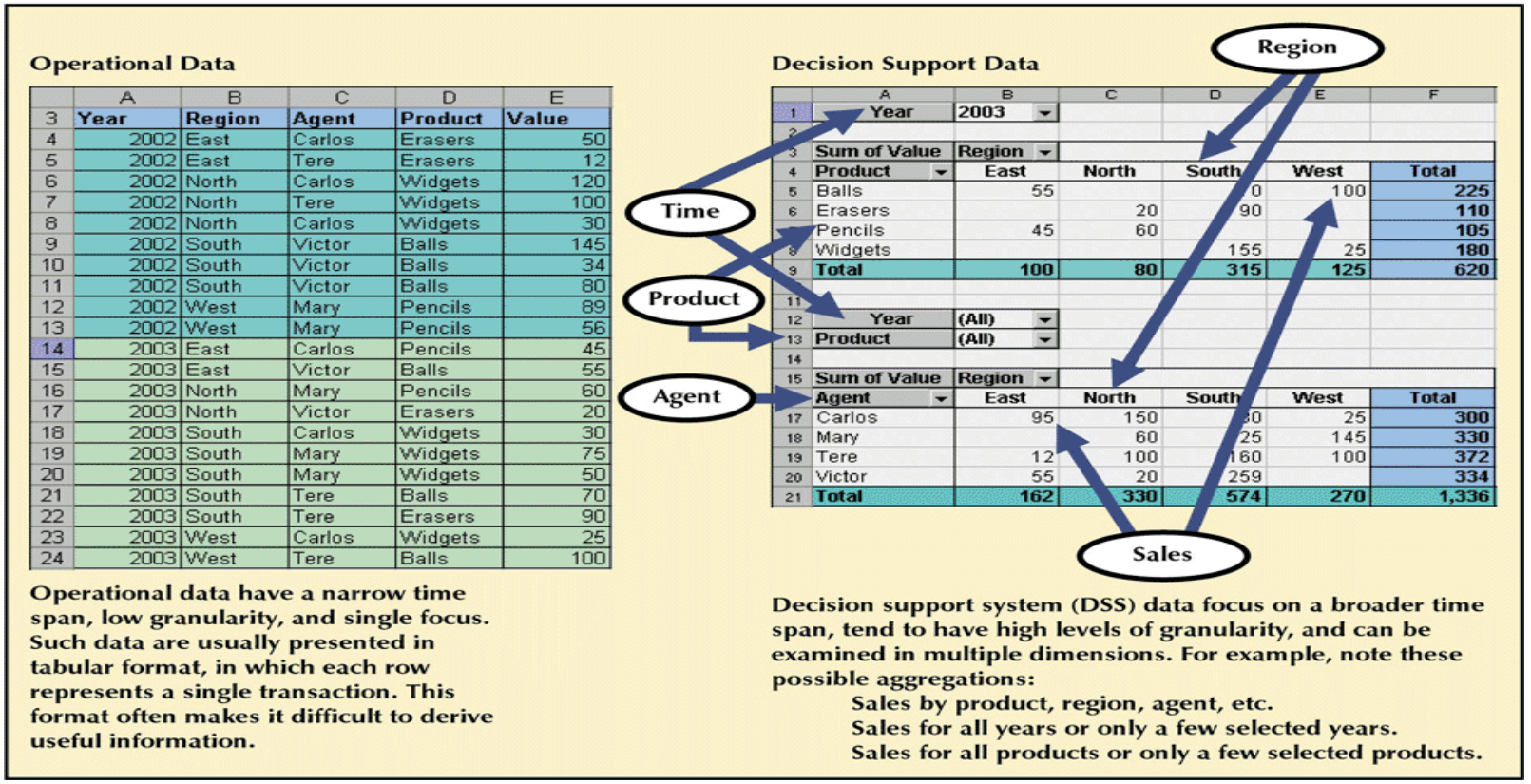
Main Components of a Decision Support System (DSS)

FIGURE 12.1 MAIN COMPONENTS OF A DECISION SUPPORT SYSTEM (DSS)



Transforming Operational Data Into Decision Support Data

FIGURE 12.2 TRANSFORMING OPERATIONAL DATA INTO DECISION SUPPORT DATA



Contrasting Operational and DSS Data Characteristics

TABLE 12.2 CONTRASTING OPERATIONAL AND DSS DATA CHARACTERISTICS

CHARACTERISTIC	OPERATIONAL DATA	DSS DATA
Data currency	Current operations Real-time data	Historic data Snapshot of company data Time component (week/month/year)
Granularity	Atomic-detailed data	Summarized data
Summarization level	Low; some aggregate yields	High; many aggregation levels
Data model	Highly normalized Mostly relational DBMS	Nonnormalized Complex structures Some relational, but mostly multidimensional DBMS
Transaction type	Mostly updates	Mostly query
Transaction volumes	High update volumes	Periodic loads and summary calculations
Transaction speed	Updates are critical	Retrievals are critical
Query activity	Low to medium	High
Query scope	Narrow range	Broad range
Query complexity	Simple to medium	Very complex
Data volumes	Hundreds of megabytes and up to gigabytes	Hundreds of gigabytes to terabytes

Ten-Year Sales History for a Single Department, in Millions of Dollars

TABLE 12.3 TEN-YEAR SALES HISTORY FOR A SINGLE DEPARTMENT, IN MILLIONS OF DOLLARS

YEAR	SALES
1994	8,227
1995	9,109
1996	10,104
1997	11,553
1998	10,018
1999	11,875
2000	12,699
2001	14,875
2002	16,301
2003	19,986

Yearly Sales Summaries, Two Stores and Two Departments per Store, in Millions of Dollars

TABLE 12.4 YEARLY SALES SUMMARIES, TWO STORES AND TWO DEPARTMENTS PER STORE, IN MILLIONS OF DOLLARS

YEAR	STORE	DEPARTMENT	SALES
1994	A	1	1,985
1994	A	2	2,401
1994	B	1	1,879
1994	B	2	1,962
YEAR	STORE	DEPARTMENT	SALES
…	…	…	…
1998	A	1	3,912
1998	A	2	4,158
1998	B	1	3,426
1998	B	2	1,203
…	…	…	…
2003	A	1	7,683
2003	A	2	6,912
2003	B	1	3,768
2003	B	2	1,623

The Data Warehouse

- Integrated, subject-oriented, time-variant, nonvolatile database that provides support for decision making
- Growing industry: $8 billion in 1998
- Range from desktop to huge:
  - Walmart: 900-CPU, 2,700 disk, 23TB Teradata system
- Lots of buzzwords, hype
  - slice & dice, rollup, MOLAP, pivot, ...

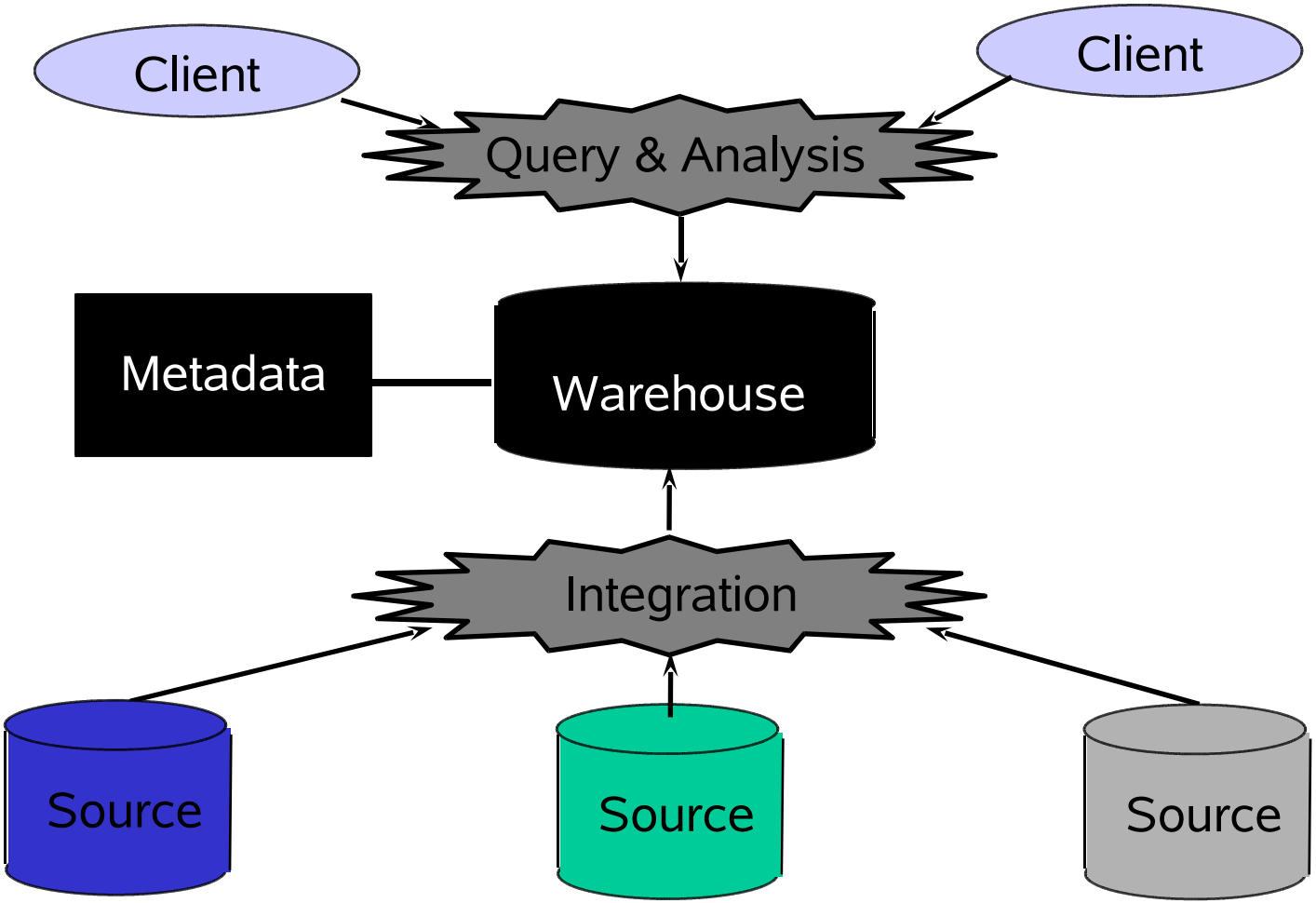
What is a Warehouse?

- Collection of diverse data
  - subject oriented
  - aimed at executive, decision maker
  - often a copy of operational data
  - with value-added data (e.g., summaries, history)
  - integrated
  - time-varying
  - non-volatile

What is a Warehouse?

- Collection of tools
  - gathering data
  - cleansing, integrating, ...
  - querying, reporting, analysis
  - data mining
  - monitoring, administering warehouse

Warehouse Architecture

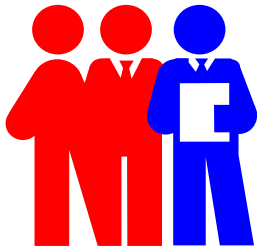


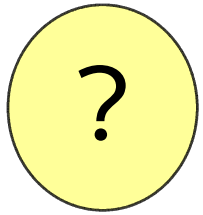
Motivating Examples

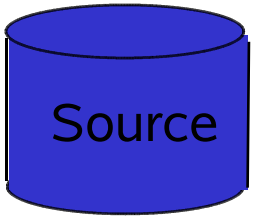
- Forecasting
- Comparing performance of units
- Monitoring, detecting fraud
- Visualization

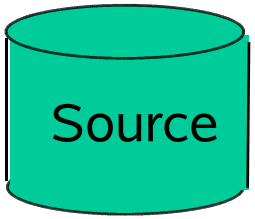
Why a Warehouse?

- Two Approaches:
  - Query-Driven (Lazy)
  - Warehouse (Eager)

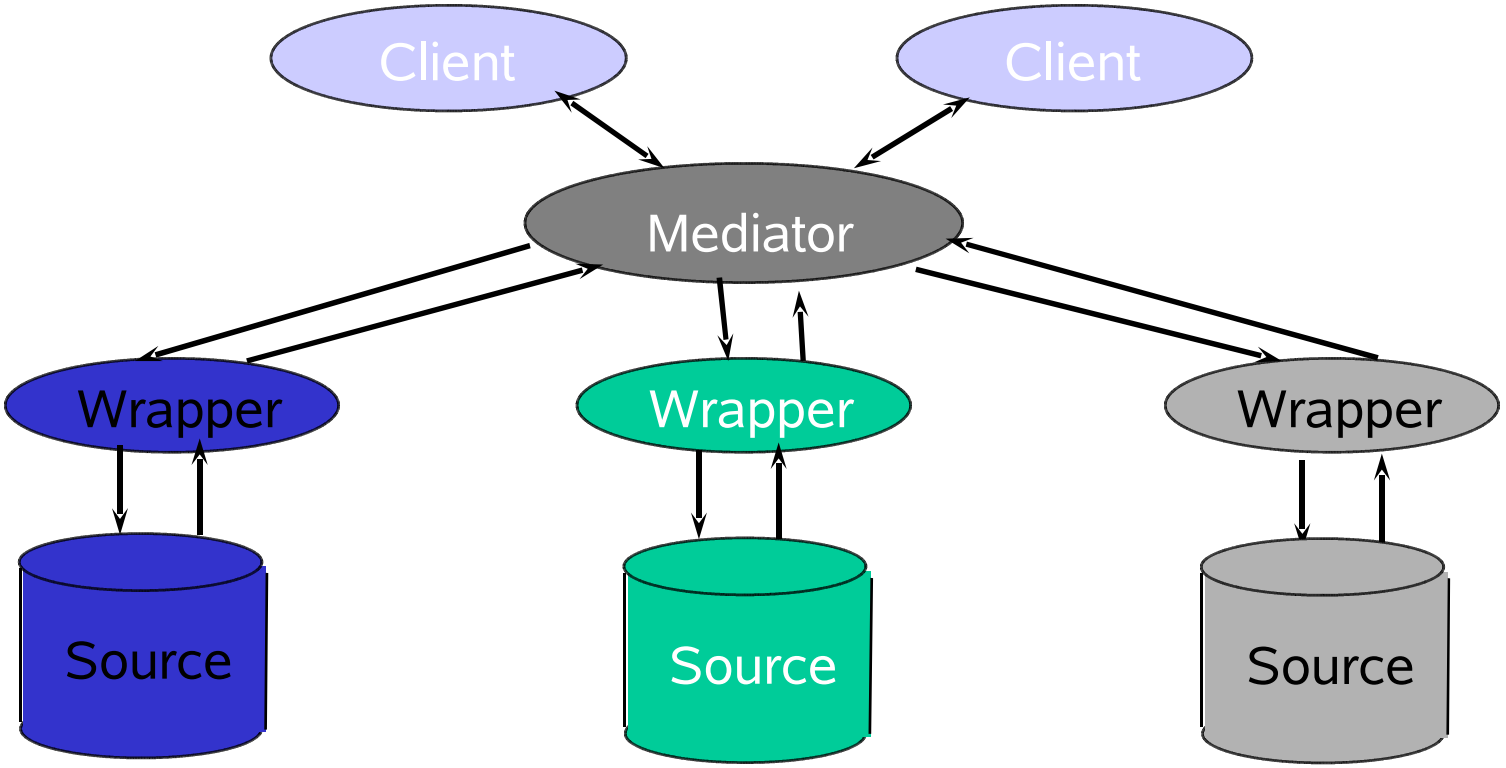








Query-Driven Approach



Advantages of Warehousing

- High query performance
- Queries not visible outside warehouse
- Local processing at sources unaffected
- Can operate when sources unavailable
- Can query data not stored in a DBMS
- Extra information at warehouse
  - Modify, summarize (store aggregates)
  - Add historical information

Advantages of Query-Driven

- No need to copy data
  - less storage
  - no need to purchase data
- More up-to-date data
- Query needs can be unknown
- Only query interface needed at sources

Data Marts

- Smaller warehouses
- Spans part of organization
  - e.g., marketing (customers, products, sales)
- Do not require enterprise-wide consensus
  - but long term integration problems?

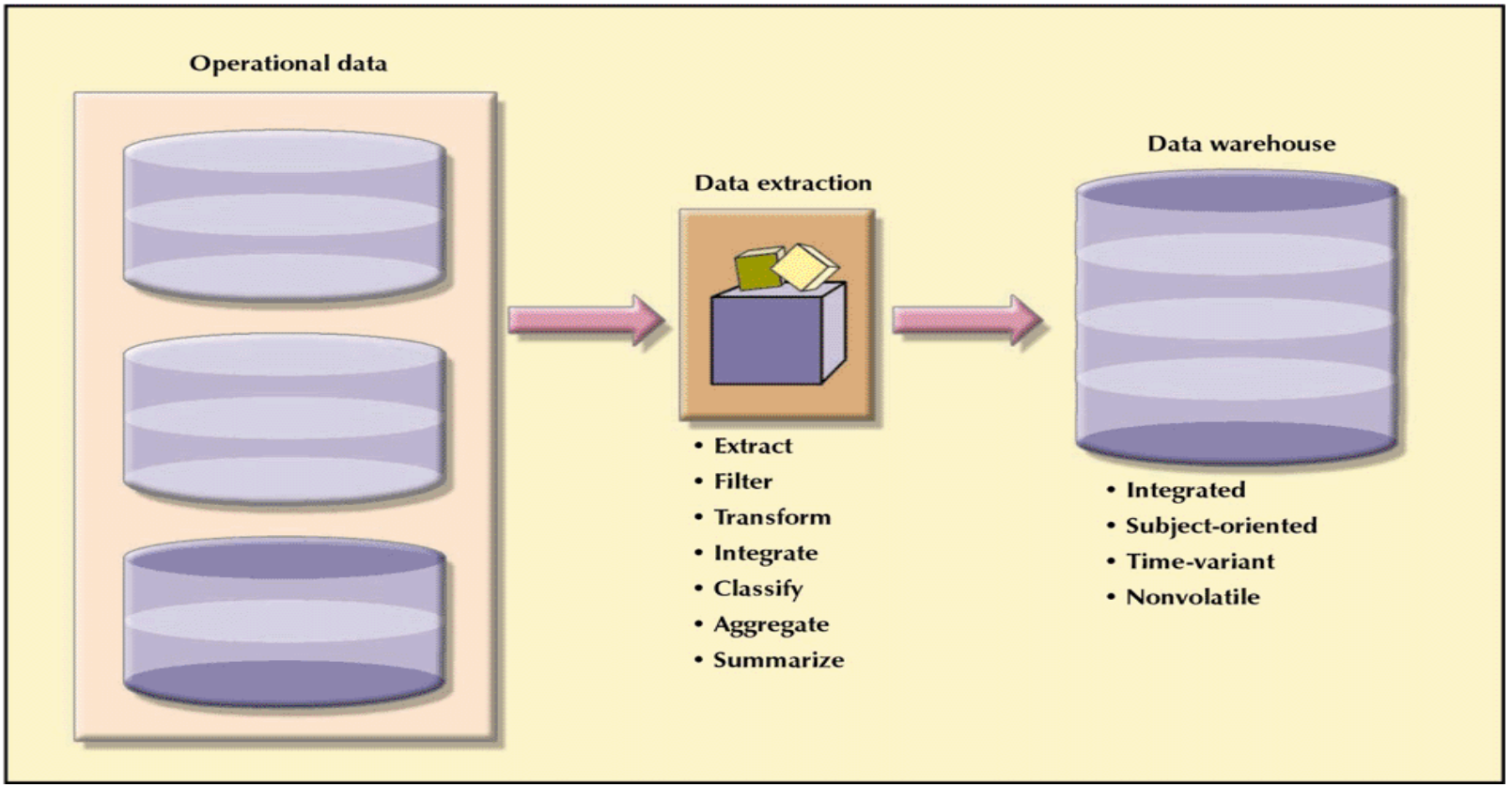
A Comparison of Data Warehouse and Operational Database Characteristics

TABLE 12.5 A COMPARISON OF DATA WAREHOUSE AND OPERATIONAL DATABASE CHARACTERISTICS

CHARACTERISTIC	OPERATIONAL DATABASE DATA	DATA WAREHOUSE DATA
Integrated	Similar data can have different representations or meanings. For example, Social Security numbers may be stored as ###-##-#### or as #########, and a given condition may be labeled as T/F or 0/1 or Y/N. A sales value may be shown in thousands or in millions.	Provide a unified view of all data elements with a common definition and representation for all business units.
Subject-oriented	Data are stored with a functional, or process, orientation. For example, data may be stored for invoices, payments, credit amounts, and so on.	Data are stored with a subject orientation that facilitates multiple views of the data and facilitates decision making. For example, sales may be recorded by product, by division, by manager, or by region.
Time-variant	Data are recorded as current transactions. For example, the sales data may be the sale of a product on a given date, such as $342.78 on 12-MAY-2004.	Data are recorded with a historical perspective in mind. Therefore, a time dimension is added to facilitate data analysis and various time comparisons.
Nonvolatile	Data updates are frequent and common. For example, an inventory amount changes with each sale. Therefore, the data environment is fluid.	Data cannot be changed. Data are only added periodically from historical systems. Once the data are properly stored, no changes are allowed. Therefore, the data environment is relatively static.

Creating a Data Warehouse

FIGURE 12.3 CREATING A DATA WAREHOUSE



Online Analytical Processing

- Advanced data analysis environment that supports decision making, business modeling, and operations research
- OLAP systems share four main characteristics:
  - Use multidimensional data analysis techniques
  - Provide advanced database support
  - Provide easy-to-use end-user interfaces
  - Support client/server architecture

OLTP vs. OLAP

- OLTP: On Line Transaction Processing
  - Describes processing at operational sites
- OLAP: On Line Analytical Processing
  - Describes processing at warehouse

OLTP vs. OLAP

OLTP

- Mostly updates
- Many small transactions
- Mb-Tb of data
- Raw data
- Clerical users
- Up-to-date data
- Consistency, recoverability critical

OLAP

- Mostly reads
- Queries long, complex
- Gb-Tb of data
- Summarized, consolidated data
- Decision-makers, analysts as users

Operational vs. Multidimensional View of Sales

FIGURE 12.4 OPERATIONAL VS. MULTIDIMENSIONAL VIEW OF SALES

Database name: Ch12_Text

Table name: DW_INVOICE

INV_NUM	INV_DATE	CUS_NAME	INV_TOTAL
2034	15-May-04	Dartonik	$1,400.00
2035	15-May-04	Summer Lake	$1,200.00
2036	16-May-04	Dartonik	$1,350.00
2037	16-May-04	Summer lake	$3,100.00
2038	16-May-04	Trydon	$400.00

Table name: DW_LINE

INV_NUM	LINE_NUM	PROD_DESCRIPTION	LINE_PRICE	LINE_QUANTITY	LINE_AMOUNT
2034	1	Optical Mouse	$45.00	20	$900.00
2034	2	vWireless RF remote and laser pointer	$50.00	10	$500.00
2035	1	Everlast Hard Drive, 60 GB	$200.00	6	$1,200.00
2036	1	Optical Mouse	$45.00	30	$1,350.00
2037	1	Optical Mouse	$45.00	10	$450.00
2037	2	Roadster 56KB Ext. Modem	$120.00	5	$600.00
2037	3	Everlast Hard Drive, 60 GB	$205.00	10	$2,050.00
2038	1	NoTech Speaker Set	$50.00	8	$400.00

Multidimensional View of Sales

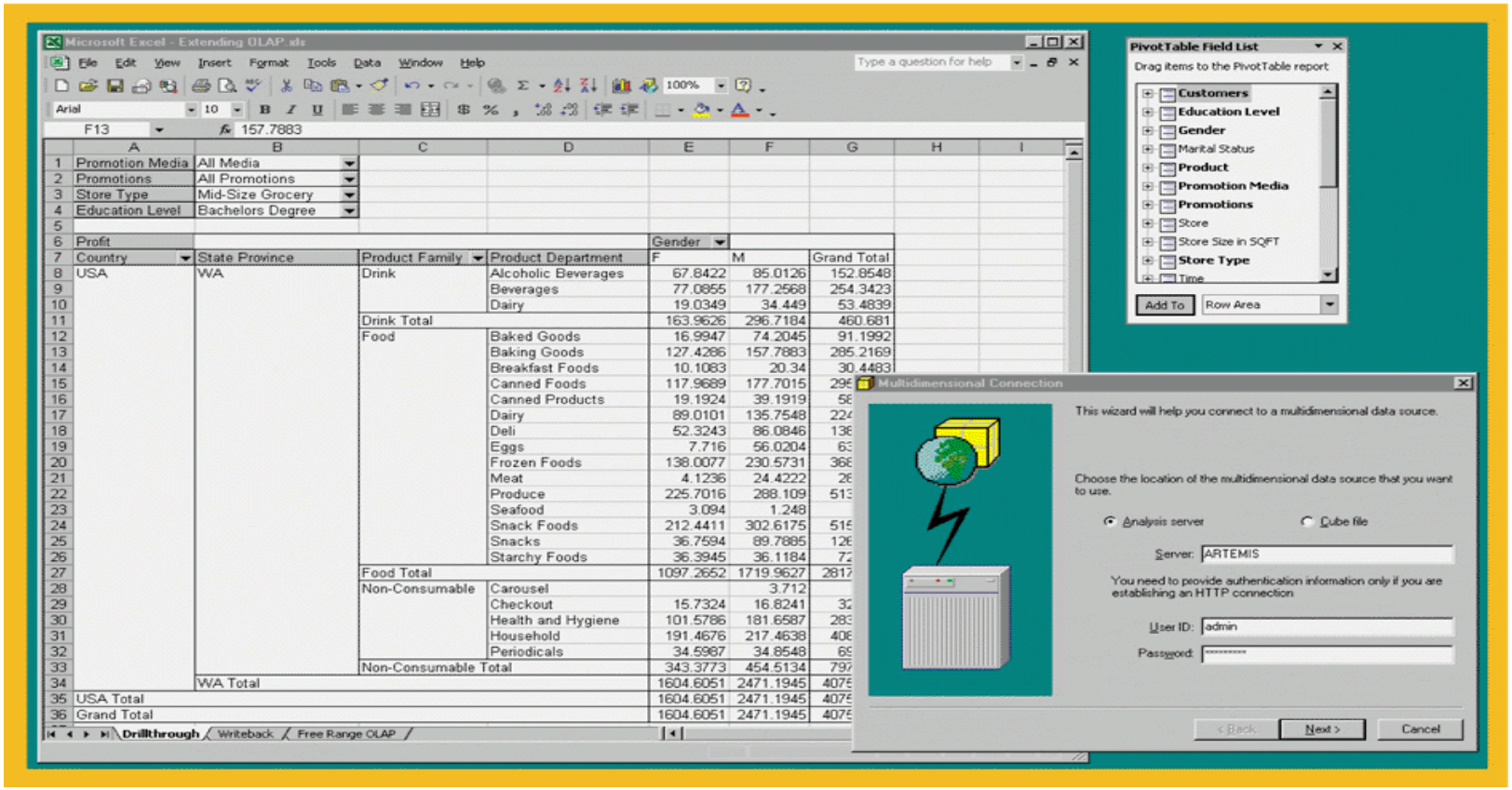
	Time Dimension		
Customer Dimension	15-May-04	16-May-04	Totals
Dartonik	$1,400.00	$1,350.00	$2,750.00
Summer Lake	$1,800.00	$3,100.00	$4,900.00
Trydon		$400.00	$400.00
Totals	$3,200.00	$4,850.00	$8,050.00

Sales are located in the intersection of a customer row and time column

Aggregations are provided for both dimensions

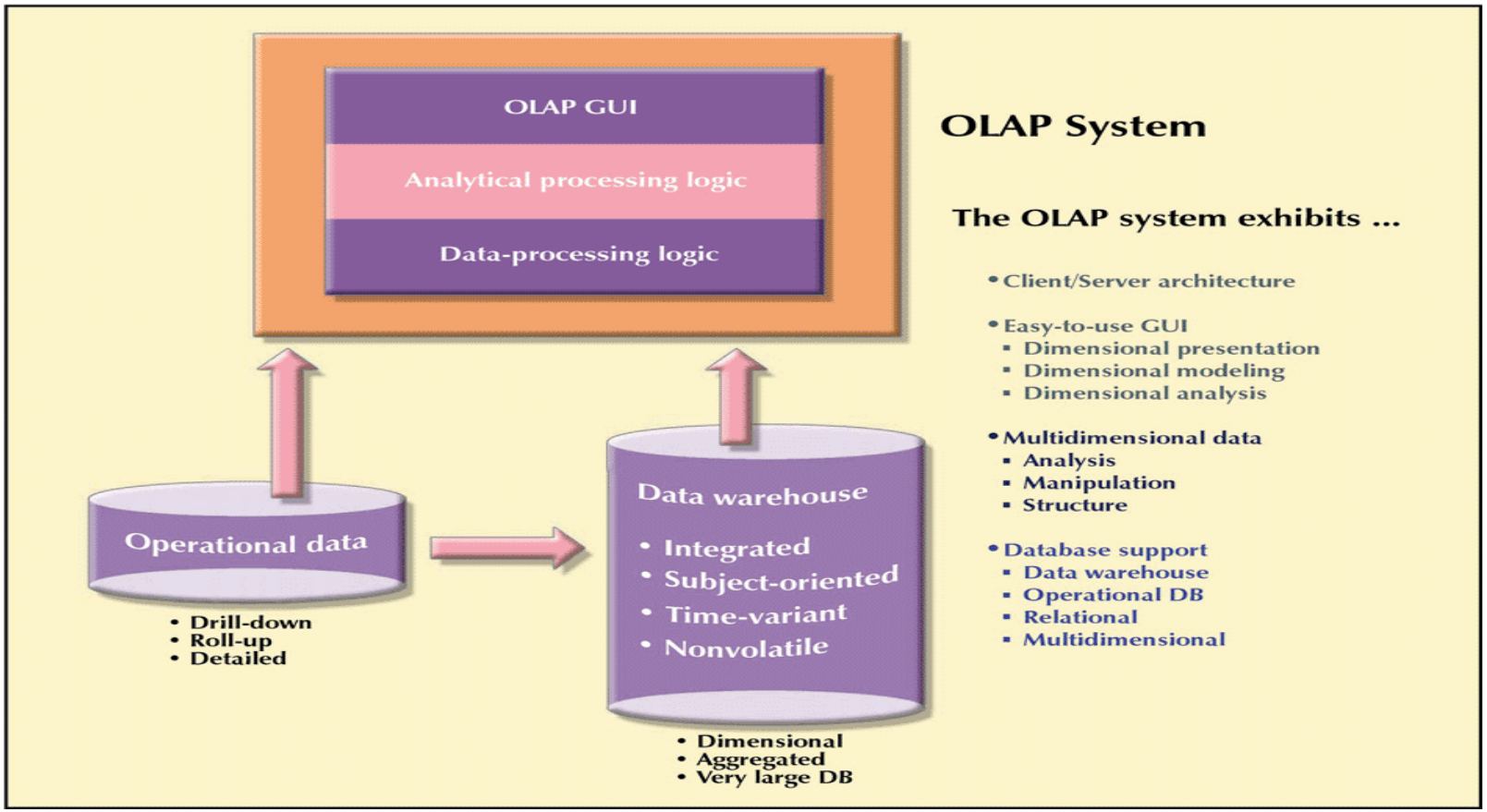
Integration of OLAP with a Spreadsheet Program

FIGURE 12.5 INTEGRATION OF OLAP WITH A SPREADSHEET PROGRAM



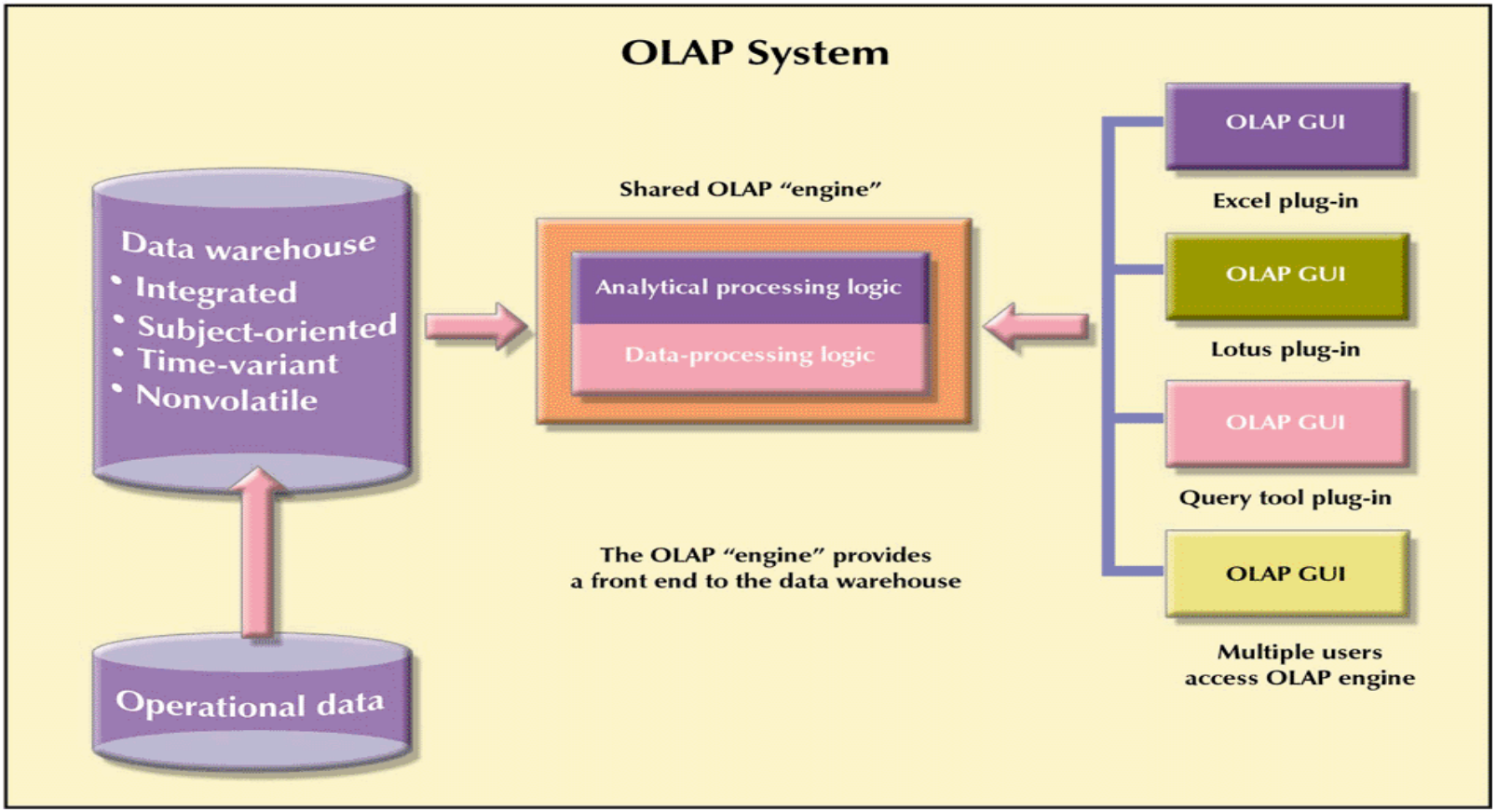
OLAP Client/Server Architecture

FIGURE 12.6 OLAP CLIENT/SERVER ARCHITECTURE



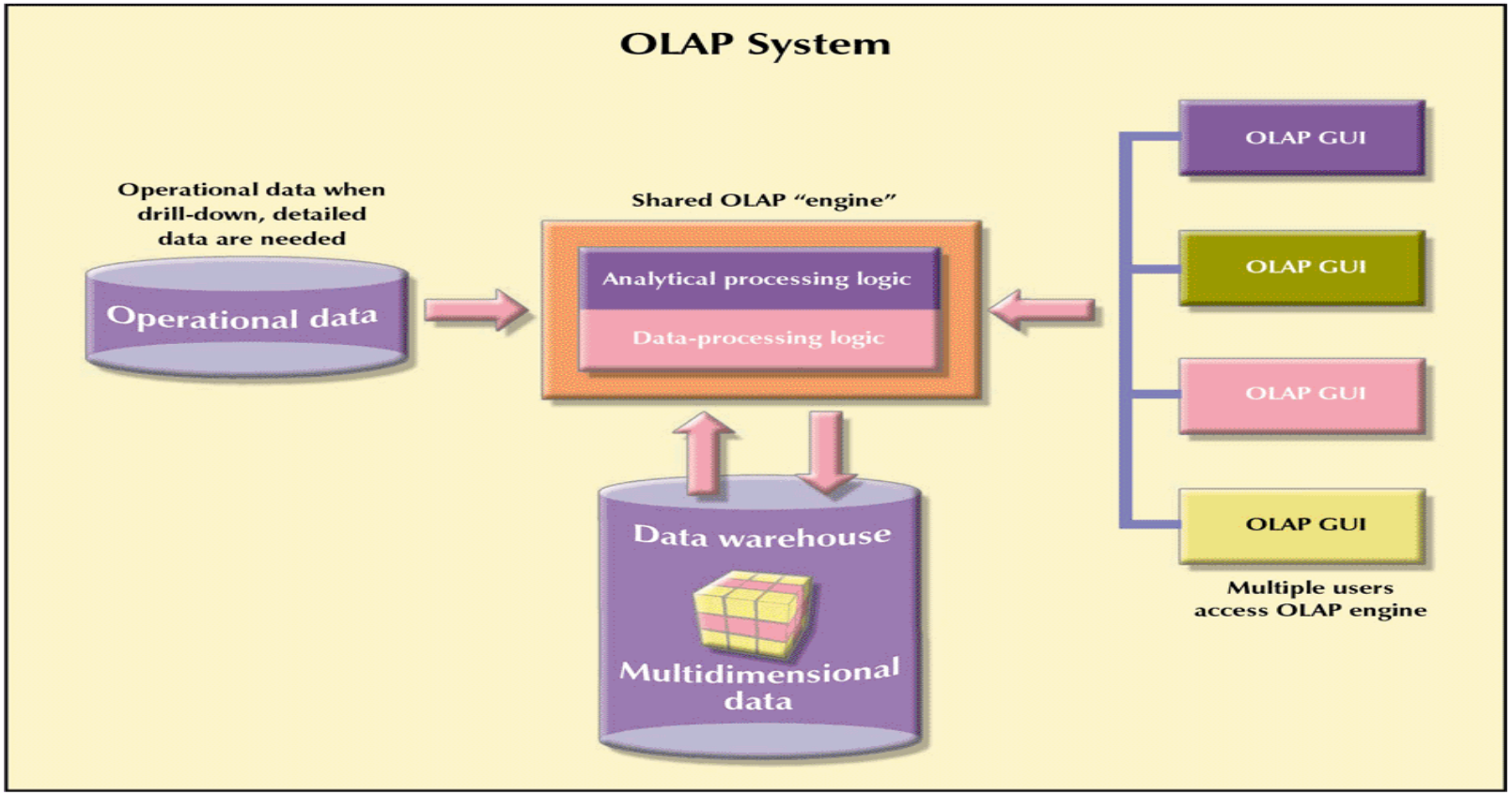
OLAP Server Arrangement

FIGURE 12.7 OLAP SERVER ARRANGEMENT



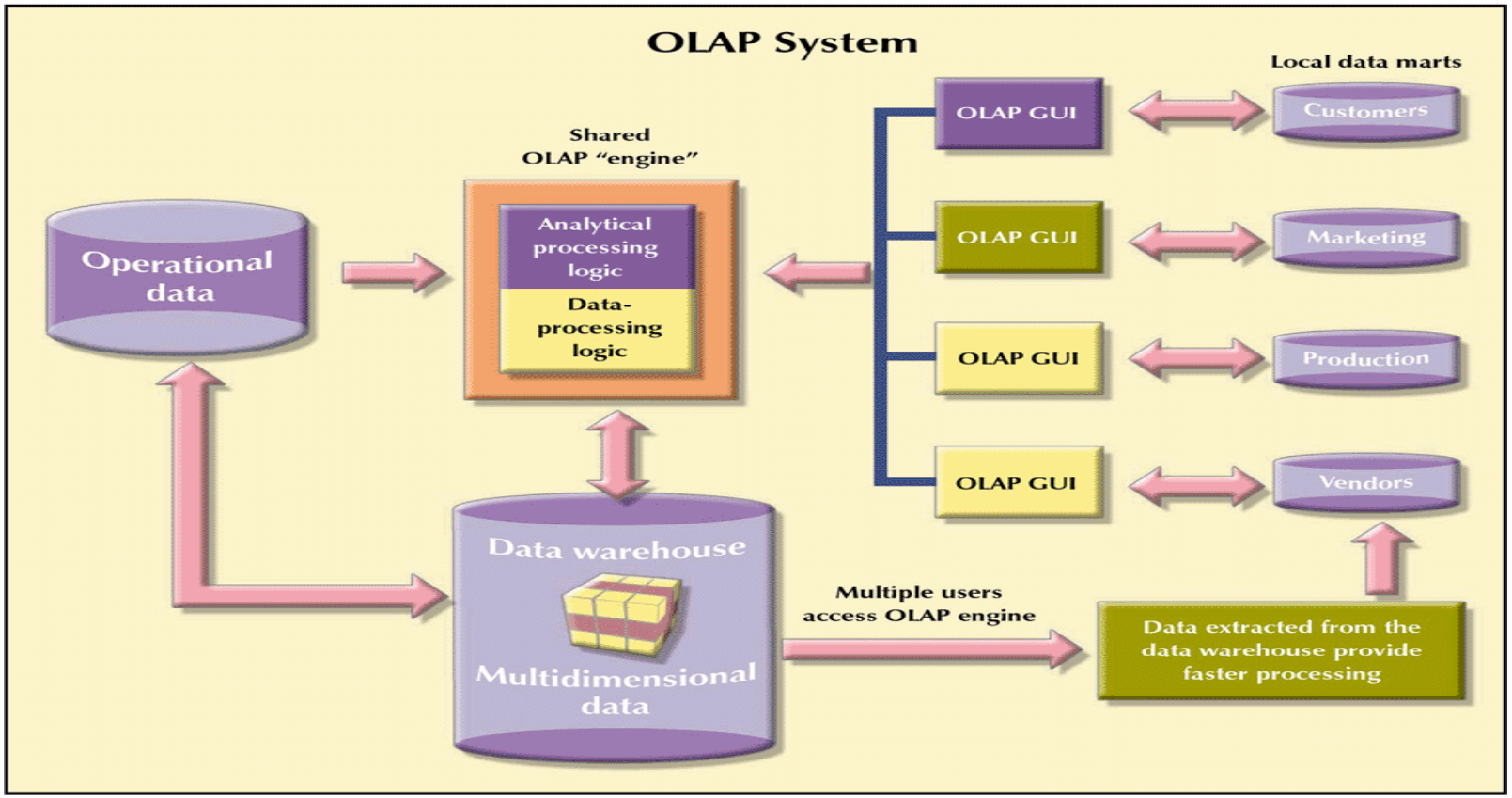
OLAP Server with Multidimensional Data Store Arrangement

FIGURE 12.8 OLAP SERVER WITH MULTIDIMENSIONAL DATA STORE ARRANGEMENT



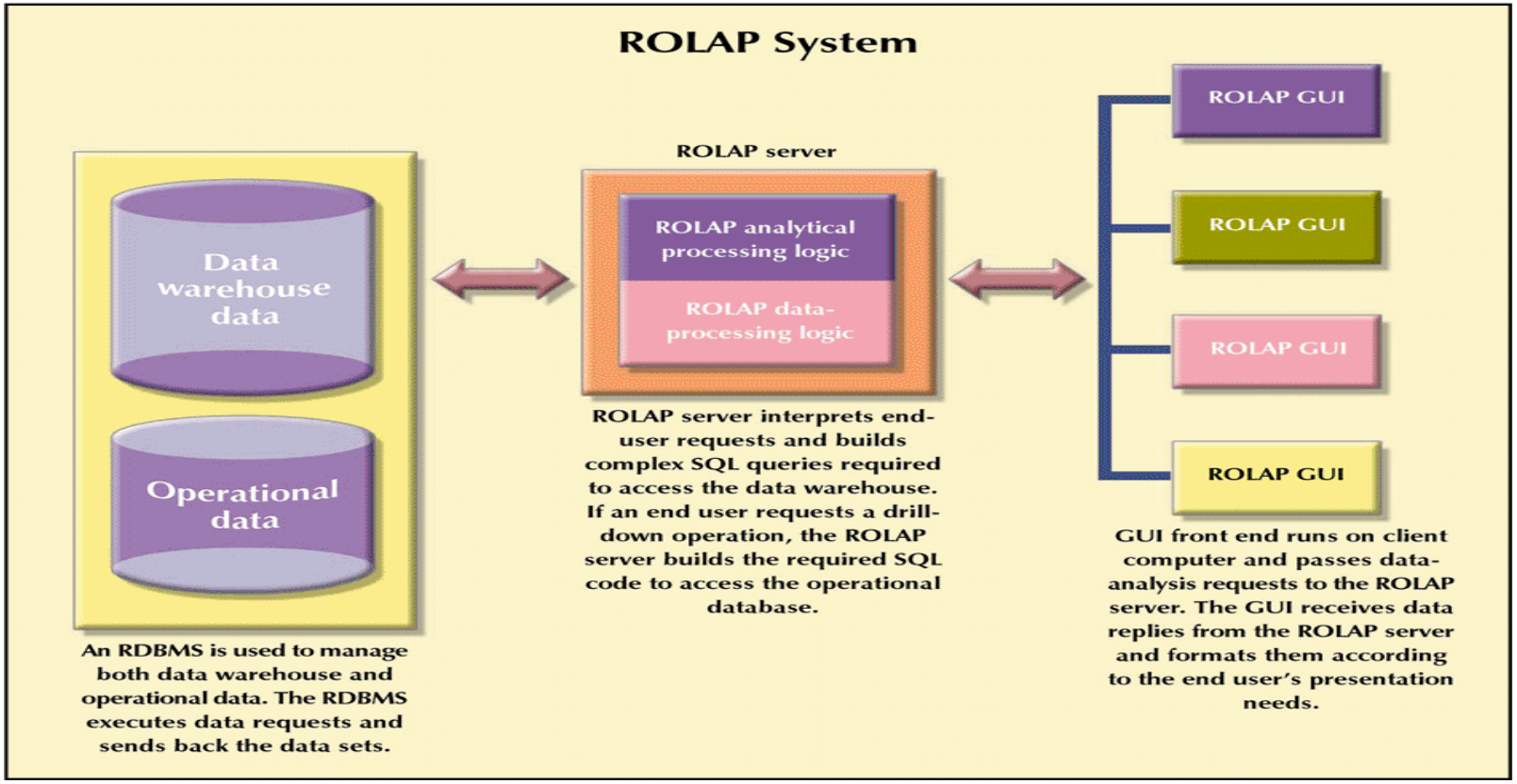
OLAP Server With Local Mini Data Marts

FIGURE 12.9 OLAP SERVER WITH LOCAL MINI DATA MARTS



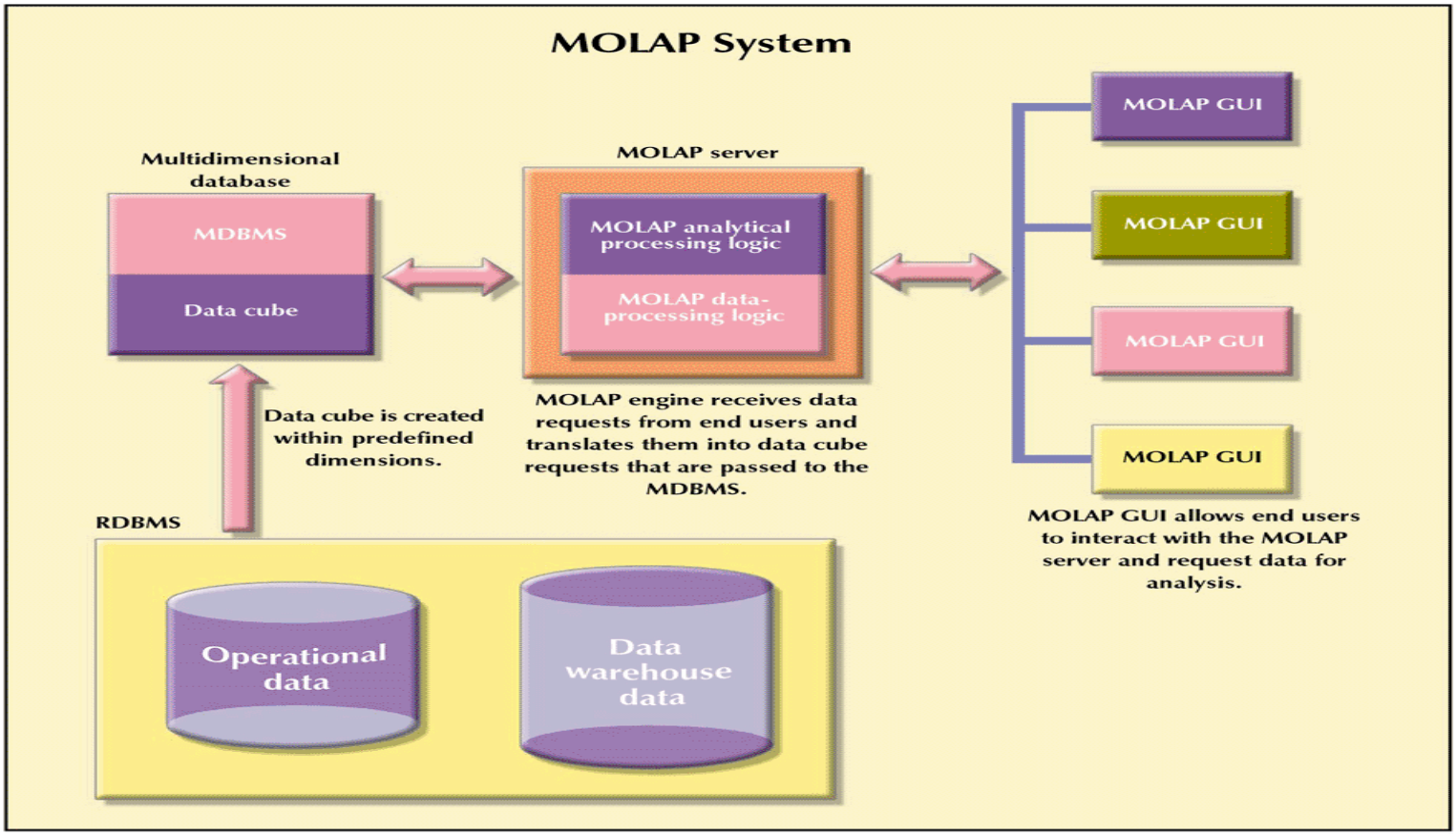
Typical ROLAP Client/Server Architecture

FIGURE 12.10 TYPICAL ROLAP CLIENT/SERVER ARCHITECTURE



MOLAP Client/Server Architecture

FIGURE 12.11 MOLAP CLIENT/SERVER ARCHITECTURE



Relational vs. Multidimensional OLAP

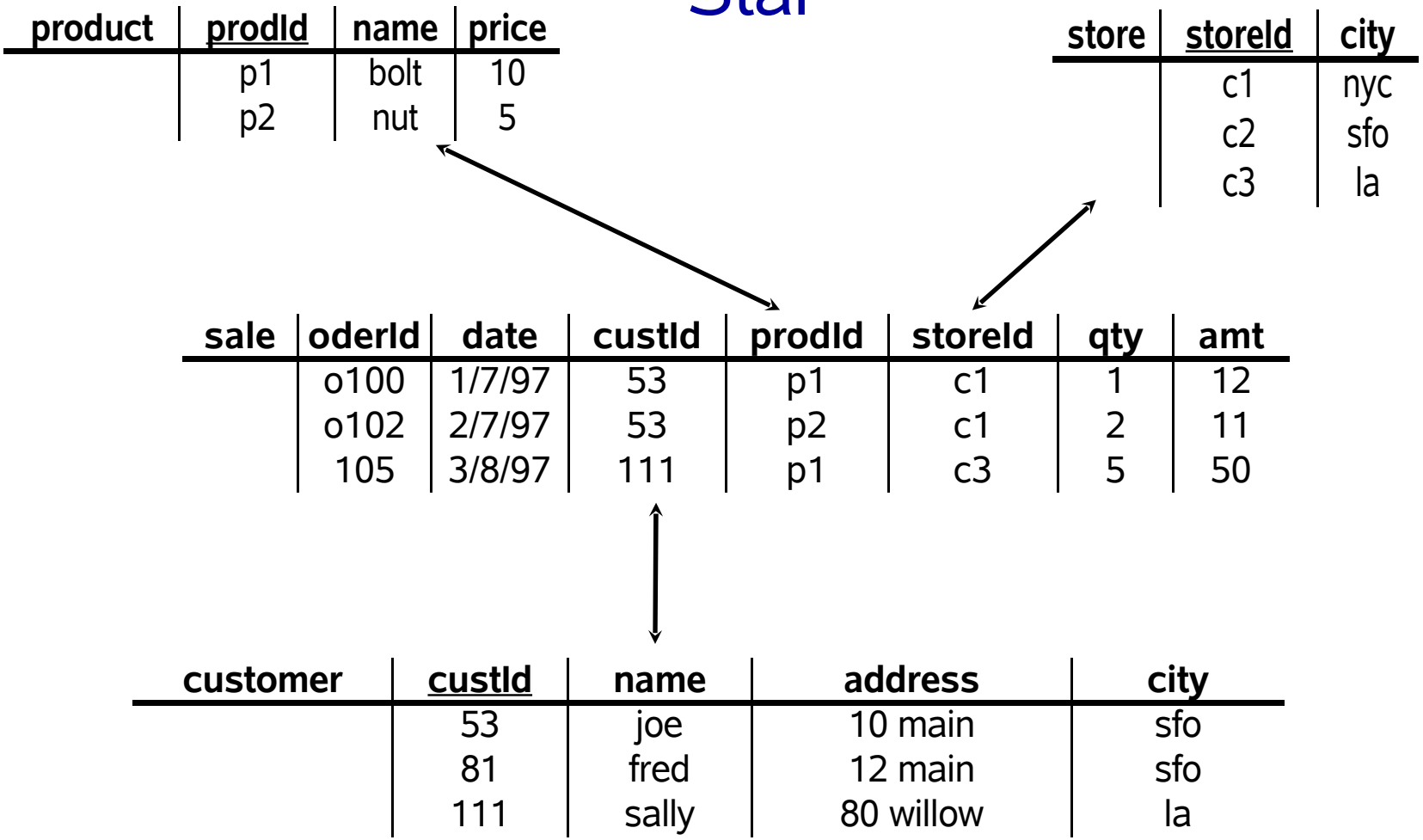
TABLE 12.8 RELATIONAL VS. MULTIDIMENSIONAL OLAP

CHARACTERISTIC	ROLAP	MOLAP
Schema	Uses star schema Additional dimensions can be added dynamically	Uses data cubes Additional dimensions require re-creation of the data cube
Database size	Medium to large	Small to medium
Architecture	Client/server Standards-based Open	Client/server Proprietary
Access	Supports ad hoc requests Unlimited dimensions	Limited to predefined dimensions
Resources	High	Very high
Flexibility	High	Low
Scalability	High	Low
Speed	Good with small data sets; average for medium to large data sets	Faster for small to medium data sets; average for large data sets

Models & Operators

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

Star



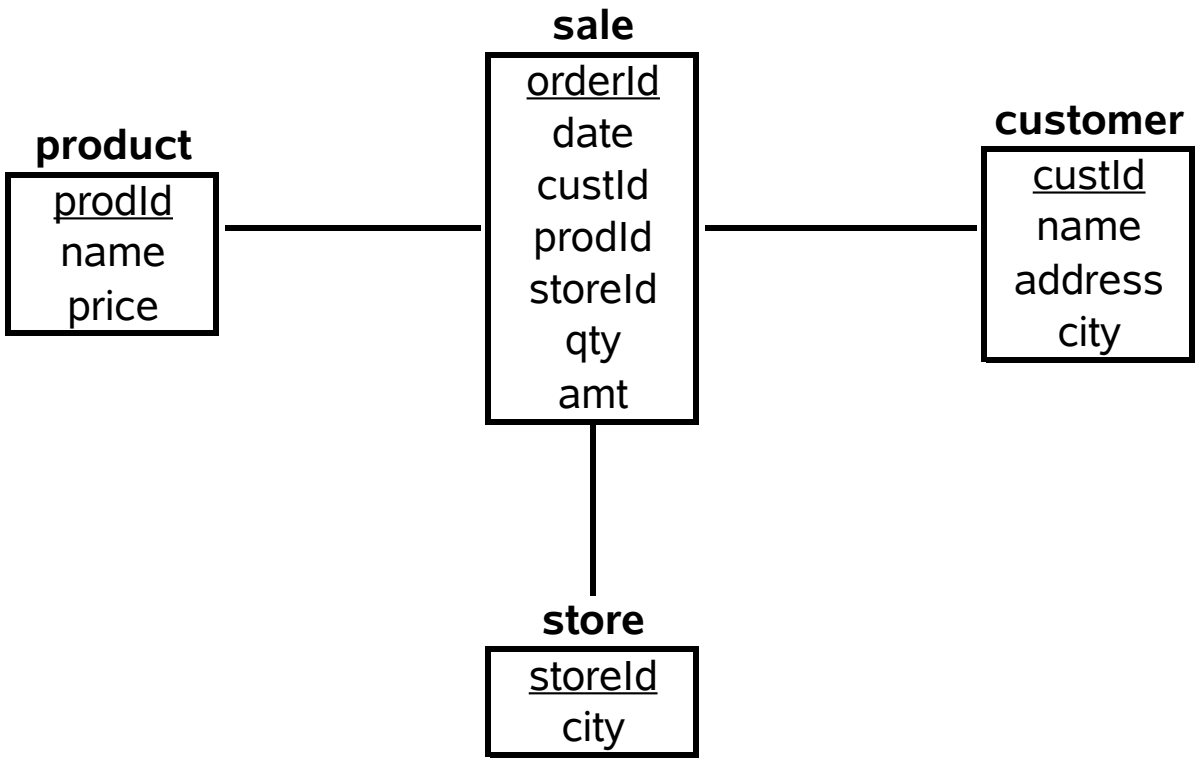
product	prodId	name	price
	p1	bolt	10
	p2	nut	5

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

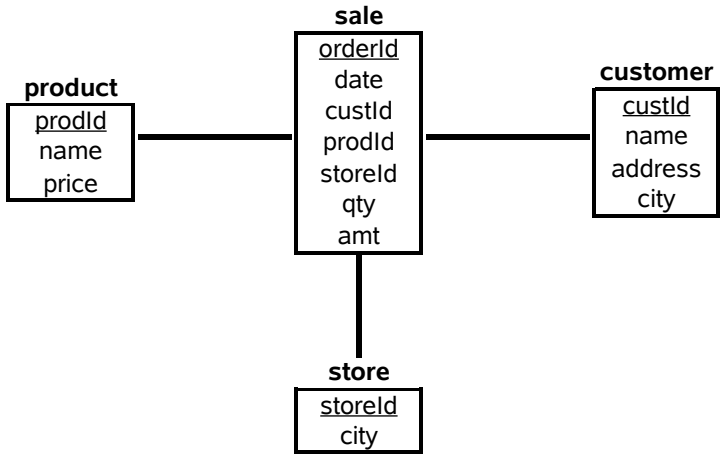
customer	custId	name	address	city
	53	joe	10 main	sfo
	81	fred	12 main	sfo
	111	sally	80 willow	la

Star Schema


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

Terms

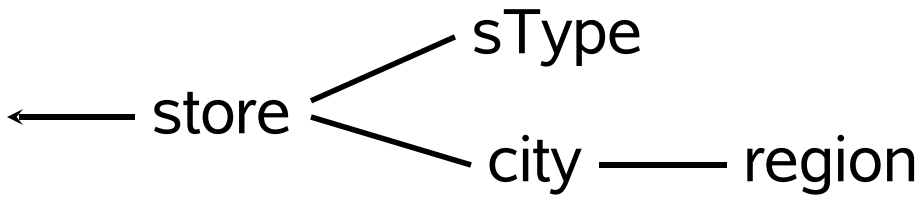
- Fact table
- Dimension tables
- Measures



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.

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

Review

- DSS, Data Warehouse, OLAP
- What is the architecture of Data Warehouse?
- What is the difference between operational data (OLTP) and Data Warehouse (OLAP) data

Review-2

- What is the database schema in data warehouse (OLAP)?
- What are fact tables, dimension tables?
- What are dimensional attributes and measure (dependent) attributes

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	

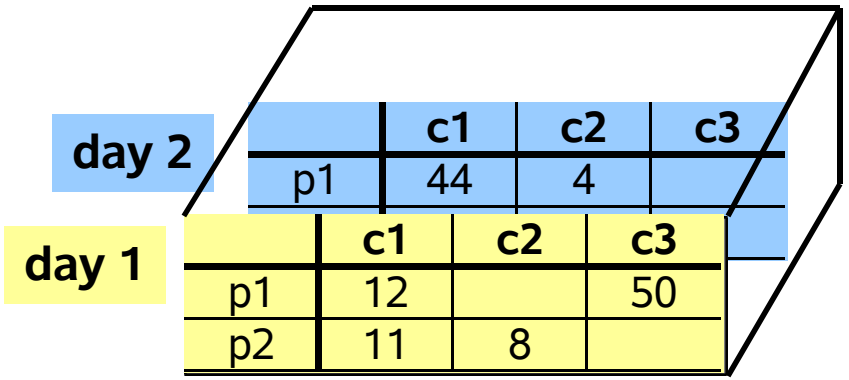
dimensions = 2

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

ROLAP vs. MOLAP

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

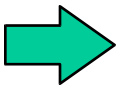
Typical OLAP Queries

- Often, OLAP queries begin with a "star join": the natural join of the fact table with all or most of the dimension tables.
- The typical OLAP query will:
  1. Start with a star join.
  2. Select for interesting tuples, based on dimension data.
  3. Group by one or more dimensions.
  4. Aggregate certain attributes of the result.

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

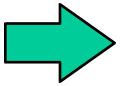


81

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

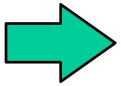


ans	date	sum
	1	81
	2	48

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

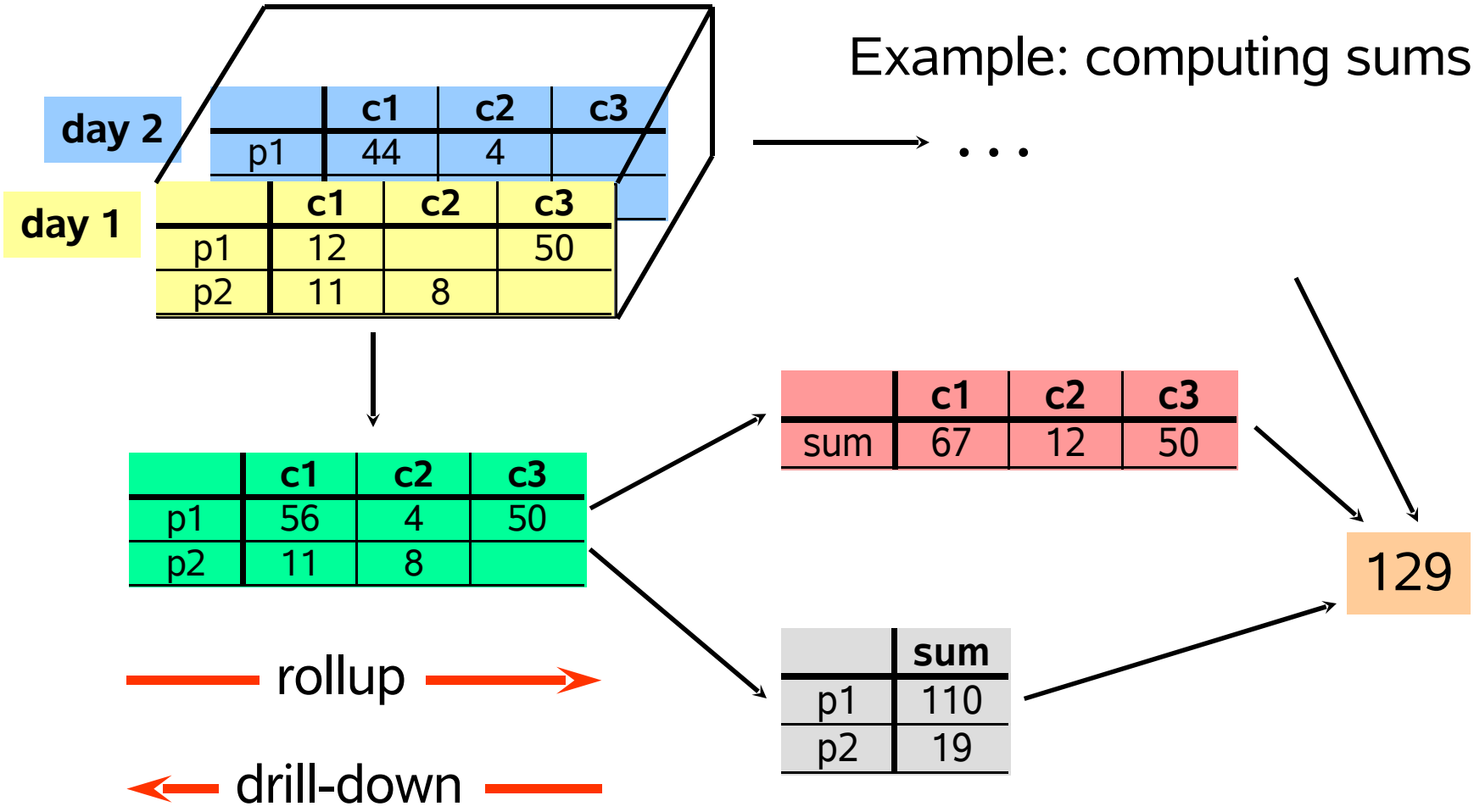


sale	prodId	date	amt
	p1	1	62
	p2	1	19
	p1	2	48

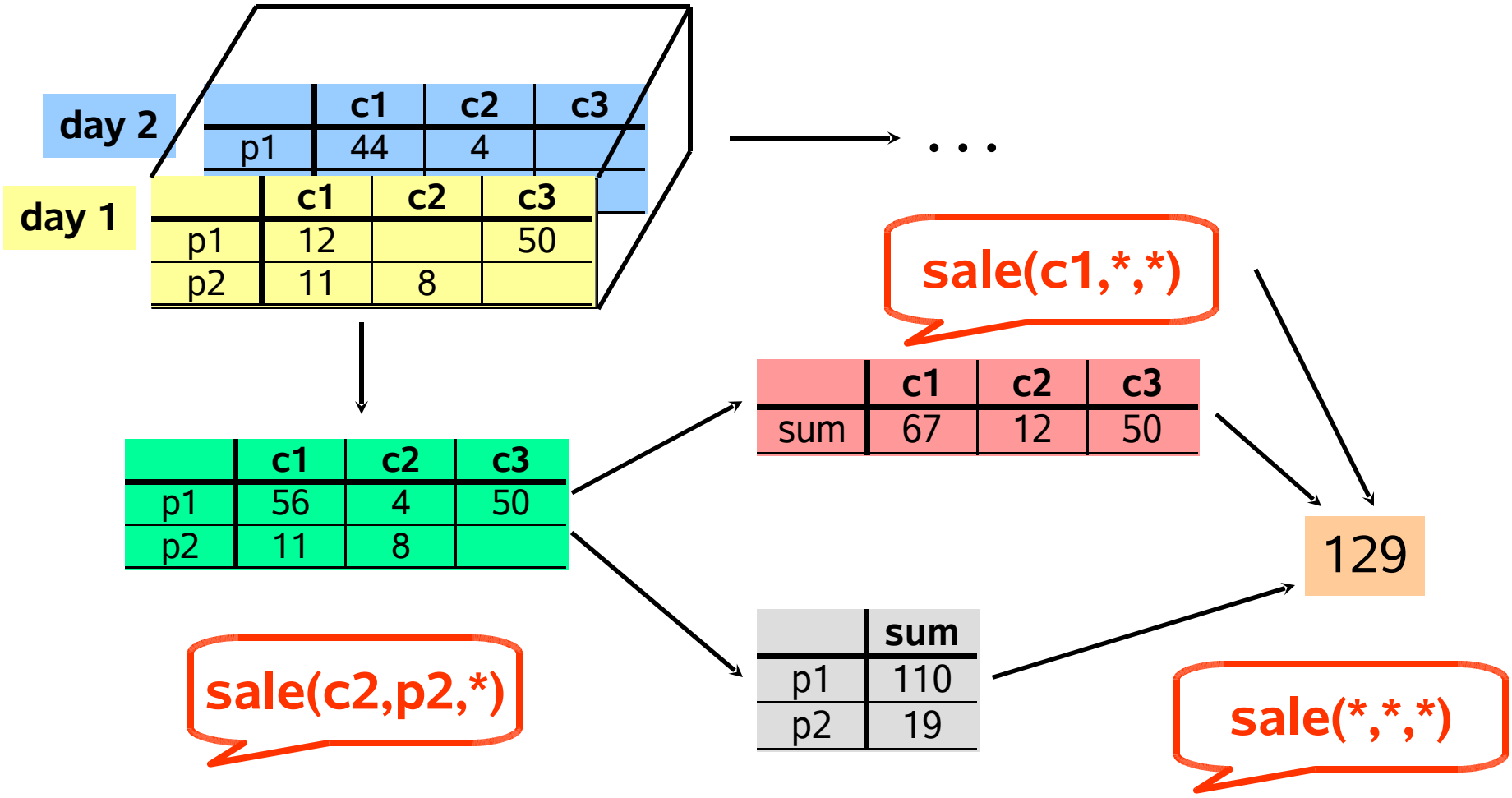
Aggregates

- Operators: sum, count, max, min, median, ave
- "Having" clause
- Using dimension hierarchy
  - average by region (within store)
  - maximum by month (within date)

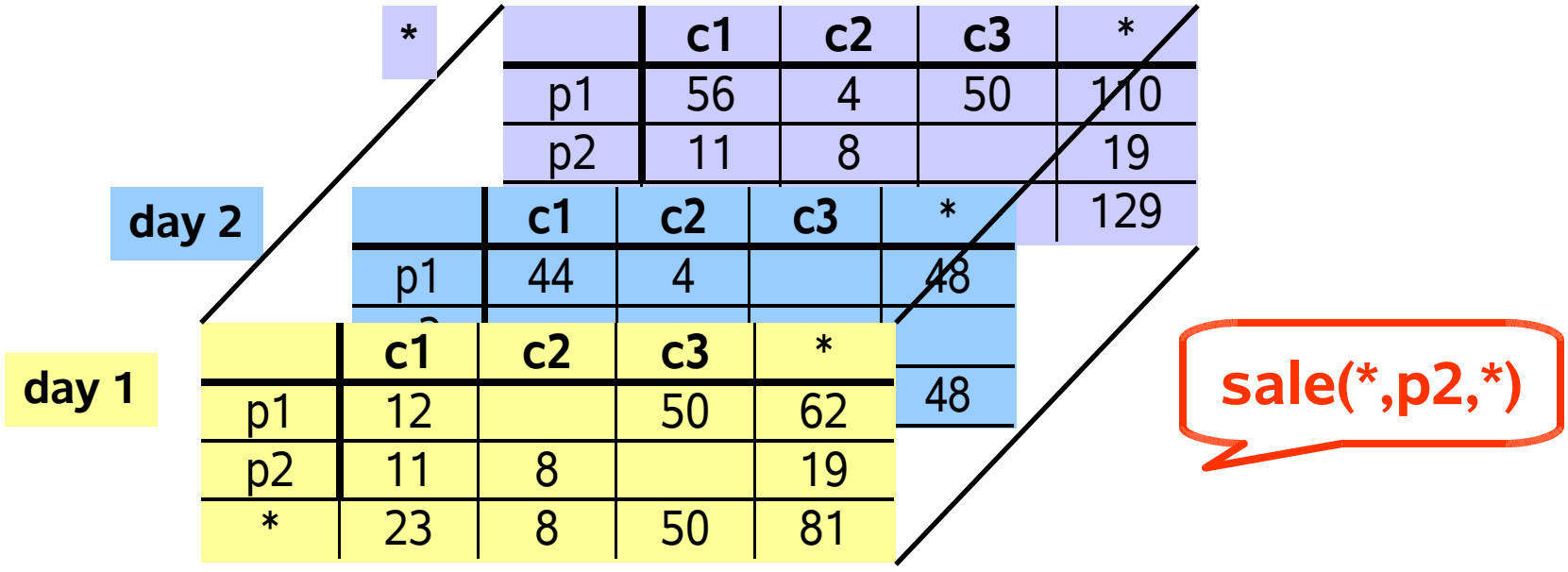
Cube Aggregation



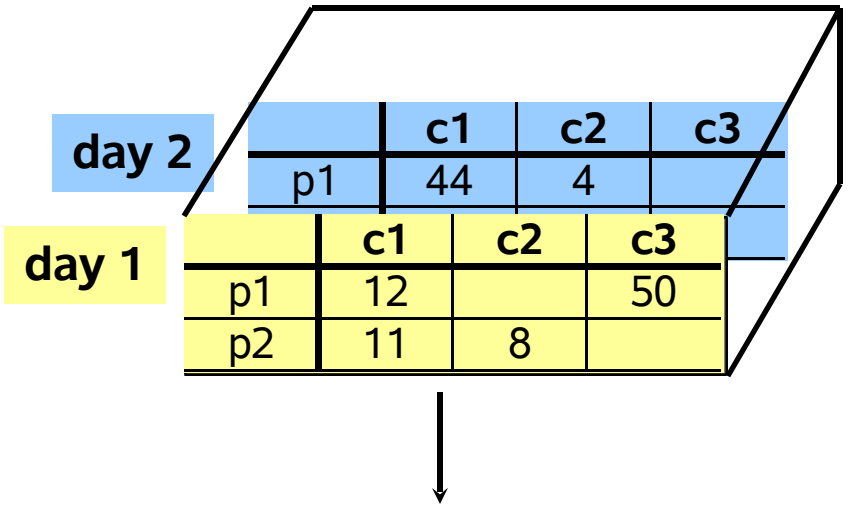
Cube Operators



Extended Cube



Aggregation Using Hierarchies



	region A	region B
p1	56	54
p2	11	8

customer
region
country

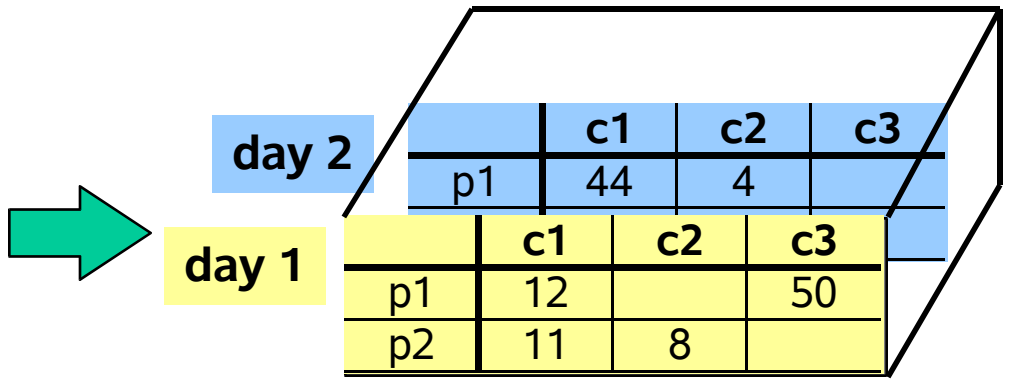
(customer c1 in Region A;
customers c2, c3 in Region B)

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:



	c1	c2	c3
p1	56	4	50
p2	11	8	

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 Mining

- Decision Trees
- Clustering
- Association Rules

SQL:1999 additional supports

- Extended Aggregation
- Ranking
- Windowing

Extended Aggregation in SQL:1999

- The **cube** operation computes union of **group by**'s on every subset of the specified attributes
- E.g. consider the query

  select *item-name, color, size,* **sum**(*number*)
  from *sales*
  group by cube(*item-name, color, size*)

  This computes the union of eight different groupings of the *sales* relation:

  { (*item-name, color, size*), (*item-name, color*),
  (*item-name, size*), (*color, size*),
  (*item-name*), (*color*),
  (*size*), ( ) }

  where ( ) denotes an empty **group by** list.
- For each grouping, the result contains the null value for attributes not present in the grouping.

60

Extended Aggregation (Cont.)

- Relational representation of cross-tab that we saw earlier, but with *null* in place of **all**, can be computed by

  select *item-name*, *color*, **sum**(*number*)
  from *sales*
  group by cube(*item-name, color*)

- The function **grouping()** can be applied on an attribute
  - ★ Returns 1 if the value is a null value representing all, and returns 0 in all other cases.

  select *item-name, color, size*, **sum**(*number*),
  grouping(*item-name*) **as** *item-name-flag*,
  grouping(*color*) **as** *color-flag*,
  grouping(*size*) **as** *size-flag*,
  from *sales*
  group by cube(*item-name, color, size*)

- Can use the function **decode()** in the **select** clause to replace such nulls by a value such as **all**
  - ★ E.g. replace *item-name* in first query by

    decode(**grouping**(item-*name*), 1, ‘all’, *item-name*)

61

Extended Aggregation (Cont.)

- The **rollup** construct generates union on every prefix of specified list of attributes
- E.g.

  select *item-name*, *color*, *size*, **sum**(*number*)
  from *sales*
  group by rollup(*item-name, color, size*)

  Generates union of four groupings:

  { (*item-name, color, size*), (*item-name, color*), (*item-name*), ( ) }

- Rollup can be used to generate aggregates at multiple levels of a hierarchy.
- E.g., suppose table *itemcategory*(*item-name, category*) gives the category of each item. Then

  select *category, item-name*, **sum**(*number*)
  from *sales, itemcategory*
  where *sales.item-name = itemcategory.item-name*
  group by rollup(*category, item-name*)

  would give a hierarchical summary by *item-name* and by *category.*

62

Extended Aggregation (Cont.)

- Multiple rollups and cubes can be used in a single group by clause
  - ★ Each generates set of group by lists, cross product of sets gives overall set of group by lists
- E.g.,

  select *item-name, color, size*, **sum**(*number*)
  from *sales*
  group by rollup(*item-name*), **rollup**(*color, size*)

  generates the groupings

  {item-name, ()} X {(color, size), (color), ()}

  = { (*item-name, color, size*), (*item-name, color*), (*item-name*), (*color, size*), (*color*), ( ) }

63

Ranking

- Ranking is done in conjunction with an order by specification.
- Given a relation student-marks(student-id, marks) find the rank of each student.

  select *student-id*, **rank**( ) **over** (**order by** *marks* **desc) as** *s-rank*
  from *student-marks*

- An extra **order by** clause is needed to get them in sorted order

  select *student-id*, **rank** ( ) **over** (**order by** *marks* **desc**) **as** *s-rank*
  from *student-marks*
  order by *s-rank*

- Ranking may leave gaps: e.g. if 2 students have the same top mark, both have rank 1, and the next rank is 3
  - ★ **dense_rank** does not leave gaps, so next dense rank would be 2

Ranking (Cont.)

- Ranking can be done within partition of the data.
- "Find the rank of students within each section."

  select *student-id, section*,
  rank ( ) **over** (**partition by** *section* **order by** *marks* **desc**)
  as *sec-rank*
  from *student-marks, student-section*
  where *student-marks.student-id = student-section.student-id*
  order by *section, sec-rank*

- Multiple **rank** clauses can occur in a single **select** clause
- Ranking is done *after* applying **group by** clause/aggregation

65

Ranking (Cont.)

- Other ranking functions:
  - ★ **percent_rank** (within partition, if partitioning is done)
  - ★ **cume_dist** (cumulative distribution)
    - ✔ fraction of tuples with preceding values
  - ★ **row_number** (non-deterministic in presence of duplicates)
- SQL:1999 permits the user to specify **nulls first** or **nulls last**

select *student-id*,
rank ( ) **over** (**order by** *marks* **desc nulls last**) **as** *s-rank*
from *student-marks*

66

Ranking (Cont.)

- For a given constant *n*, the ranking the function *ntile*(*n*) takes the tuples in each partition in the specified order, and divides them into *n* buckets with equal numbers of tuples.
- E.g.:

```
select threetile, sum(salary)
from (
        select salary, ntile(3) over (order by salary) as threetile
        from employee) as s
group by threetile
```


67

Windowing

- Used to smooth out random variations.
- E.g.: moving average: “Given sales values for each date, calculate for each date the average of the sales on that day, the previous day, and the next day”
- **Window specification** in SQL:
  - ★ Given relation *sales(date, value)*

    select *date*, ***sum***(*value*) **over**
    (**order by** *date* **between rows** 1 **preceding and** 1 **following**)
    from *sales*
- Examples of other window specifications:
  - ★ **between rows unbounded preceding and current**
  - ★ **rows unbounded preceding**
  - ★ **range between** 10 **preceding and current row**
    - ✔ All rows with values between current row value –10 to current value
  - ★ **range interval** 10 **day preceding**
    - ✔ Not including current row

68

Windowing (Cont.)

- Can do windowing within partitions
- E.g. Given a relation *transaction* (*account-number, date-time, value*), where value is positive for a deposit and negative for a withdrawal
  - ★ "Find total balance of each account after each transaction on the account"

```
select account-number, date-time,
    sum (value ) over
                (partition by account-number
                order by date-time
                rows unbounded preceding)
   as balance
from transaction
order by account-number, date-time
```


69

Review

- Data Models
  - Stars
  - Cubes
  - ROLAP/MOLAP
- Operators
  - slice & dice
  - roll-up, drill down
  - pivoting
  - data cube

Review SQL'99

- Advanced Aggregation
  - Data Cube
  - Roll up
  - Rank
  - Window

SQL Examples

1. **select** *item-name, color, size,* **sum**(*number*)
   from *sales*
   group by cube(*item-name, color, size*)
2. **select** *item-name*, *color*, *size*, **sum**(*number*)
   from *sales*
   group by rollup(*item-name, color, size*)
3. **select** *item-name, color, size*, **sum**(*number*)
   from *sales*
   group by rollup(*item-name*), **rollup**(*color, size*)

SQL Example

1. **select** *student-id*, **rank** ( ) **over** (**order by** *marks* **desc**) **as** *s-rank*
   from *student-marks*
   order by *s-rank*
2. **select** *student-id, section*,
   rank ( ) **over** (**partition by** *section* **order by** *marks* **desc**)
   as *sec-rank*
   from *student-marks, student-section*
   where *student-marks.student-id = student-section.student-id*
   order by *section, sec-rank*
3. **select** *date,* ***sum***(*value*) **over**
   (**order by** *date* **between rows** 1 **preceding and** 1 **following**)
   from *sales*
4. **select** *account-number, date-time*,
   sum (*value*) **over**
   (**partition by** *account-number*
   order by *date-time*
   rows unbounded preceding)
   as *balance*
   from *transaction*
   order by *account-number, date-time*

OLAP Implementation Techniques

- View Materialization
- Indexing
  - Bitmap Indexing
  - Join Indexing

View Materialization

- What is View?
- Why do need to materialize views?
- What to materialize?

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

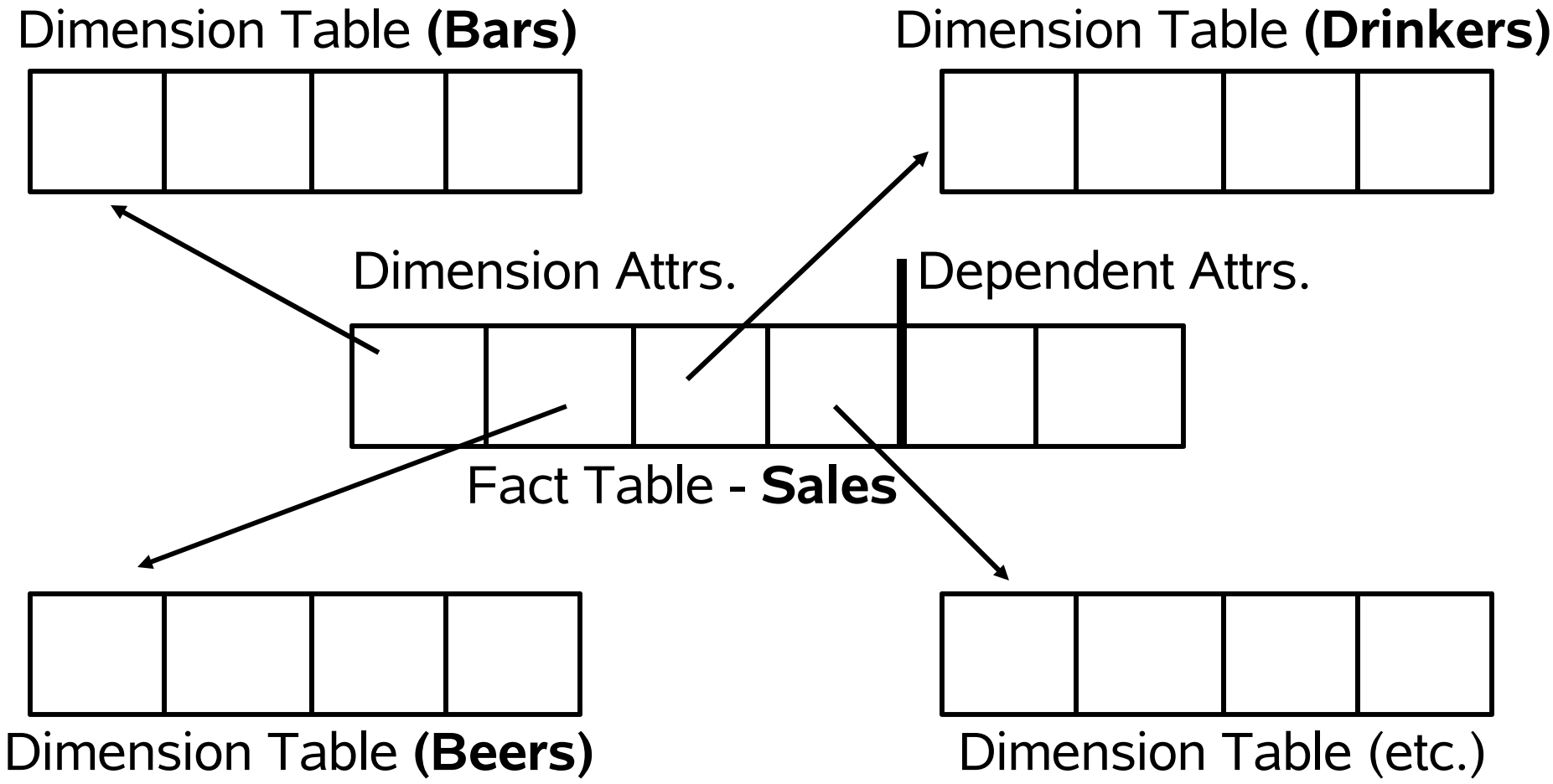
- The dimension tables include information about the bar, beer, and drinker "dimensions":

  Bars(bar, addr, license)

  Beers(beer, manf)

  Drinkers(drinker, addr, phone)

Visualization – Star Schema



Typical OLAP Queries

- Often, OLAP queries begin with a "star join": the natural join of the fact table with all or most of the dimension tables.
- Example:

```
SELECT *
FROM Sales, Bars, Beers, Drinkers
WHERE Sales.bar = Bars.bar AND
  Sales.beer = Beers.beer AND
  Sales.drinker = Drinkers.drinker;
```

Example: In SQL

```
SELECT bar, beer, SUM(price)
FROM Sales NATURAL JOIN Bars
  NATURAL JOIN Beers
WHERE addr = 'Palo Alto' AND
  manf = 'Anheuser-Busch'
GROUP BY bar, beer;
```

Using Materialized Views

- A direct execution of this query from Sales and the dimension tables could take too long.
- If we create a materialized view that contains enough information, we may be able to answer our query much faster.

Example: Materialized View

- Which views could help with our query?
- Key issues:
  1. It must join Sales, Bars, and Beers, at least.
  2. It must group by at least bar and beer.
  3. It must not select out Palo-Alto bars or Anheuser-Busch beers.
  4. It must not project out addr or manf.

Example --- Continued

- Here is a materialized view that could help:

```
CREATE VIEW BABMS(bar, addr,
     beer, manf, sales) AS
SELECT bar, addr, beer, manf,
     SUM(price) sales
FROM Sales NATURAL JOIN Bars
     NATURAL JOIN Beers
GROUP BY bar, addr, beer, manf;
```

Since bar -> addr and beer -> manf, there is no real grouping. We need addr and manf in the SELECT.

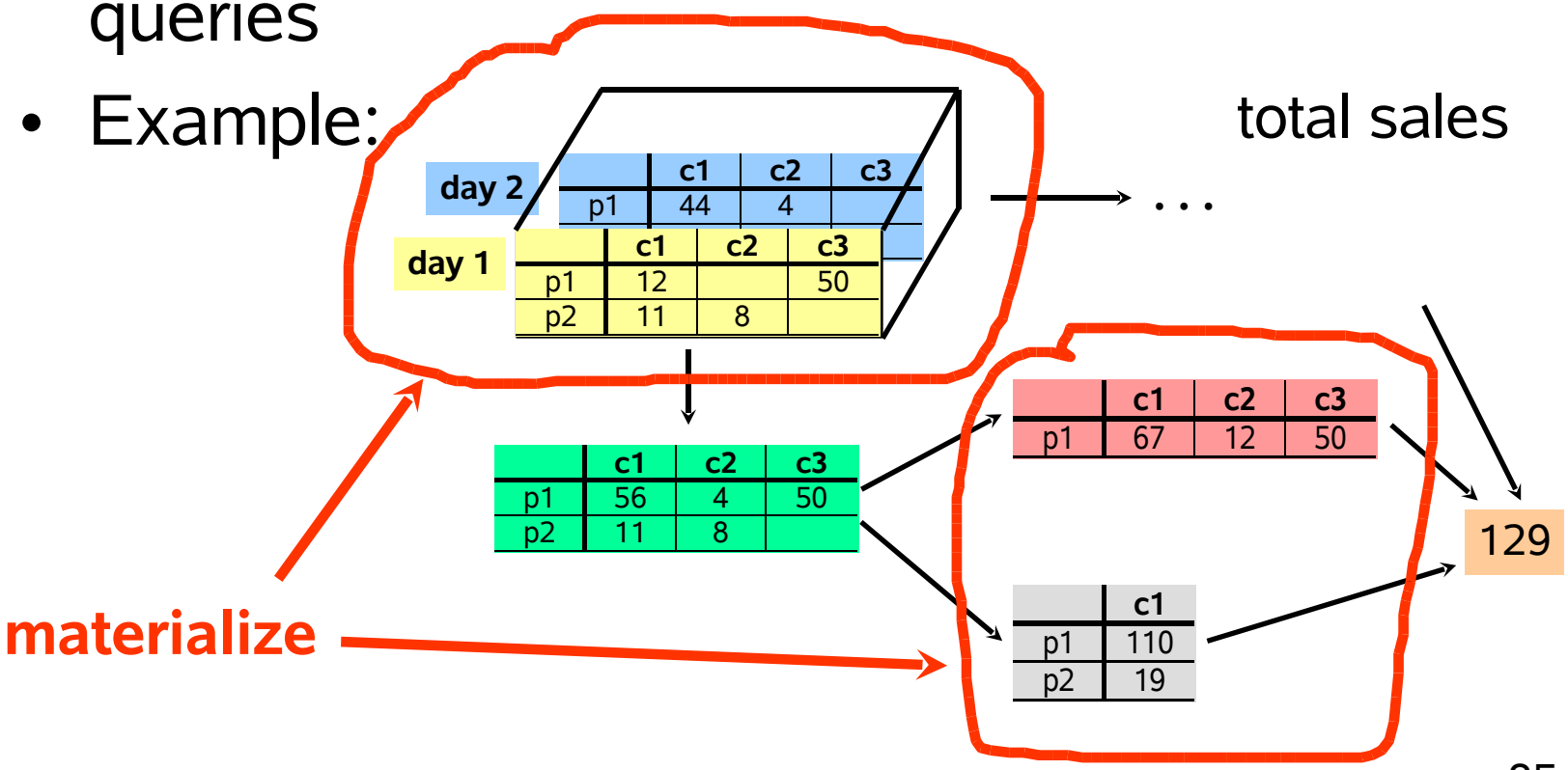
Example --- Concluded

- Here's our query using the materialized view BABMS:

```
SELECT bar, beer, sales
FROM BABMS
WHERE addr = 'Palo Alto' AND
      manf = 'Anheuser-Busch';
```

What to Materialize?

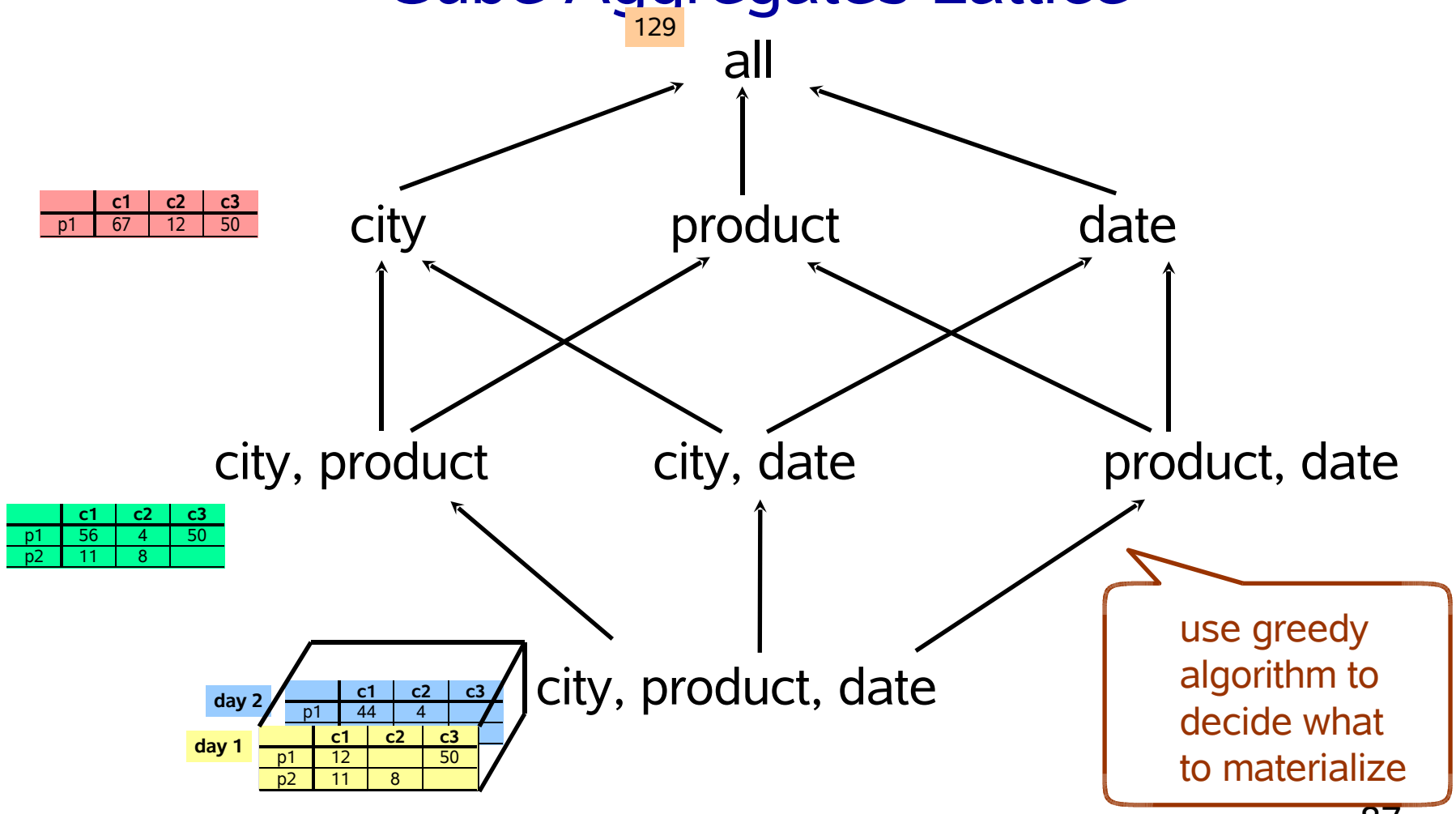
- Store in warehouse results useful for common queries
- Example:



Materialization Factors

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

Cube Aggregates Lattice



	c1	c2	c3
p1	67	12	50

	c1	c2	c3
p1	56	4	50
p2	11	8	

day 2

	c1	c2	c3
p1	44	4	

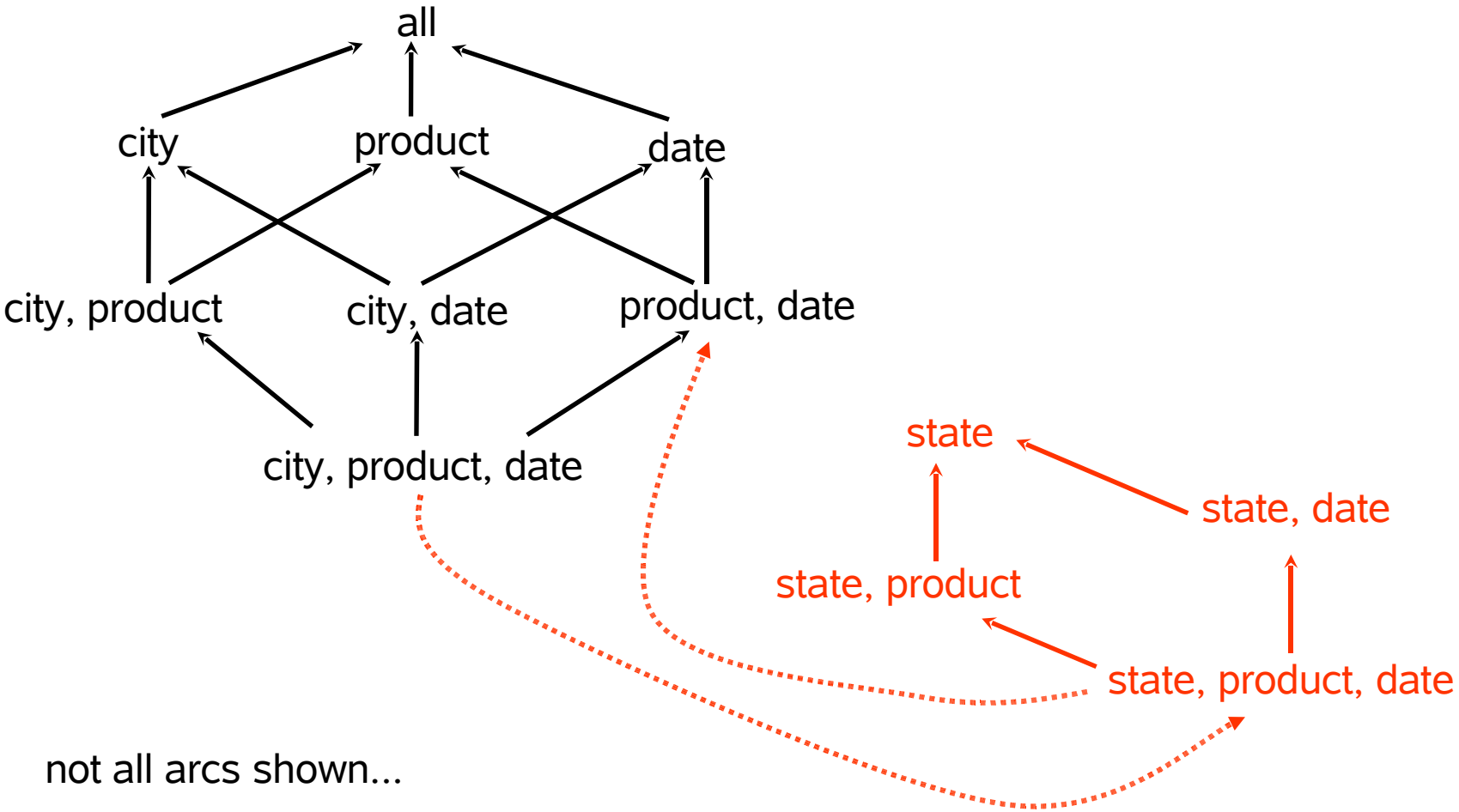
day 1

	c1	c2	c3
p1	12		50
p2	11	8	

Dimension Hierarchies

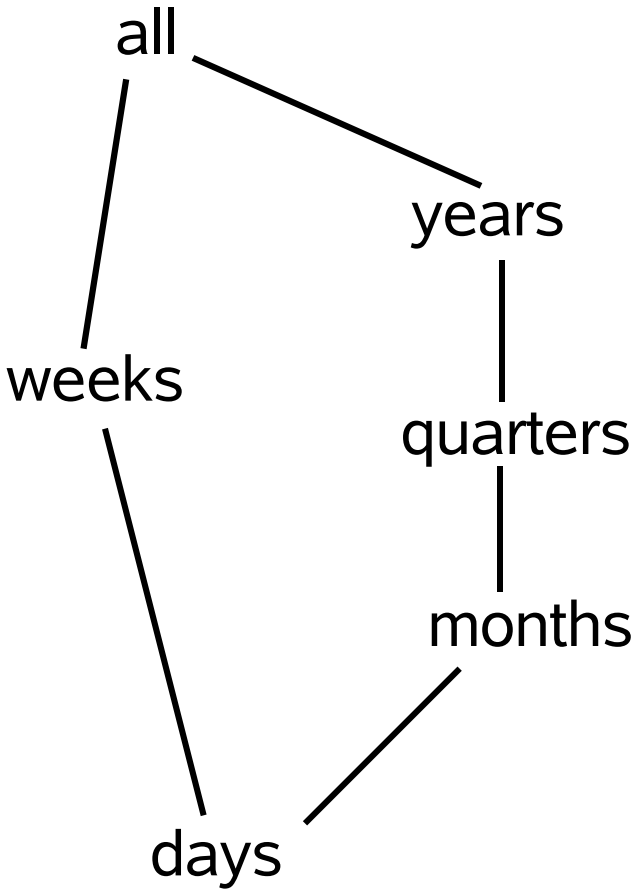
cities	city	state
	c1	CA
	c2	NY

Dimension Hierarchies



not all arcs shown...

Interesting Hierarchy



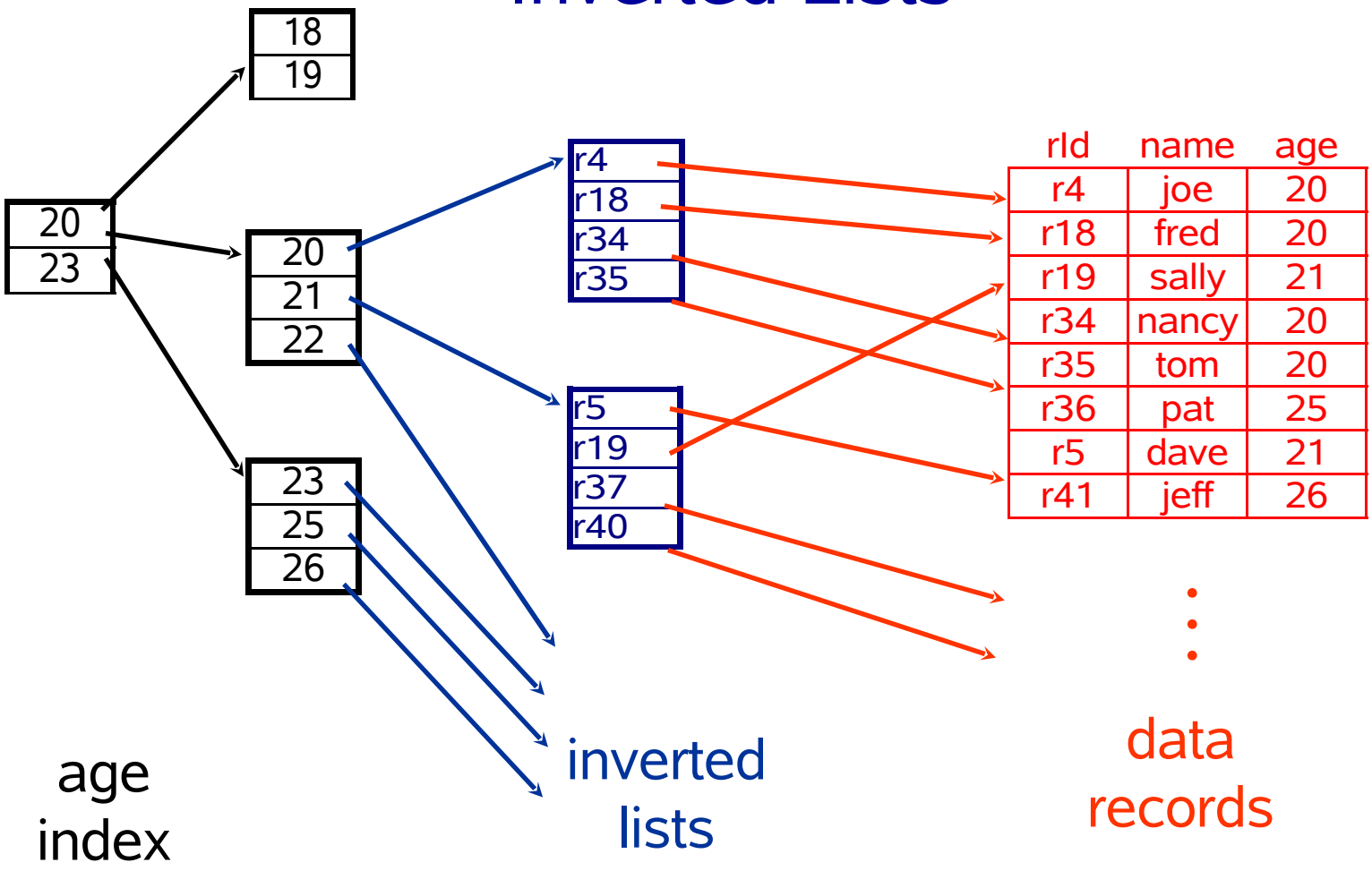
time	day	week	month	quarter	year
	1	1	1	1	2000
	2	1	1	1	2000
	3	1	1	1	2000
	4	1	1	1	2000
	5	1	1	1	2000
	6	1	1	1	2000
	7	1	1	1	2000
	8	2	1	1	2000

conceptual dimension table

Index Structures

- Traditional Access Methods
  - B-trees, hash tables, R-trees, grids, …
- Popular in Warehouses
  - inverted lists
  - bit map indexes
  - join indexes
  - text indexes

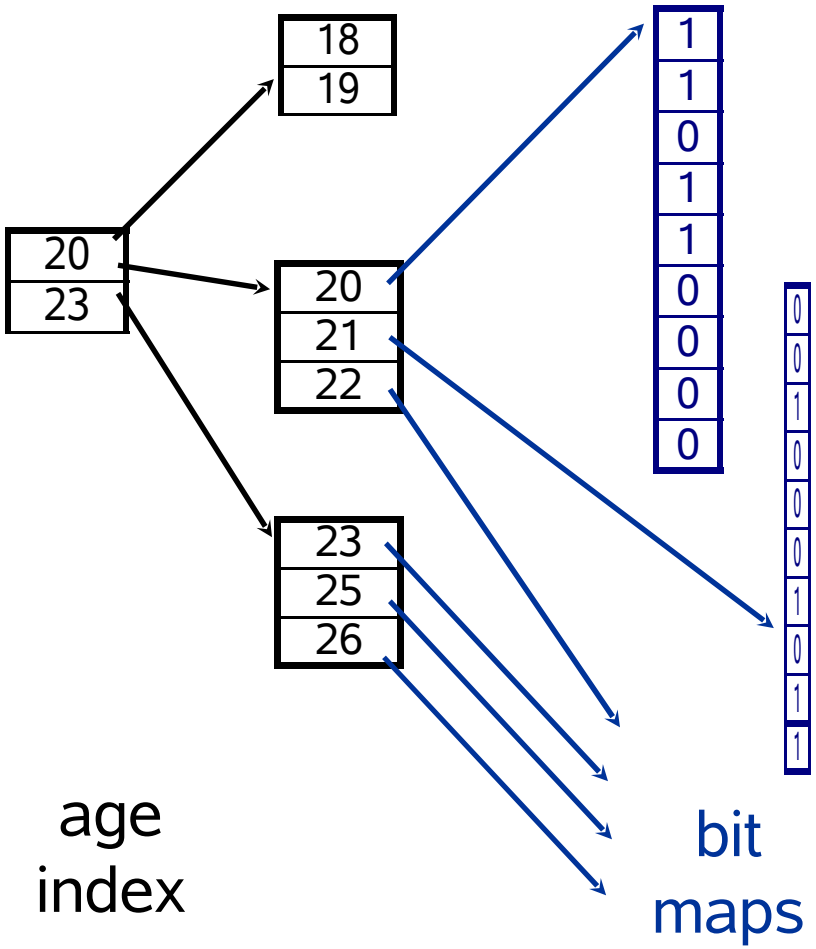
Inverted Lists



Using Inverted Lists

- Query:
  - Get people with age = 20 and name = “fred”
- List for age = 20: r4, r18, r34, r35
- List for name = “fred”: r18, r52
- Answer is intersection: r18

Bit Maps



id	name	age
1	joe	20
2	fred	20
3	sally	21
4	nancy	20
5	tom	20
6	pat	25
7	dave	21
8	jeff	26

data
records

Using Bit Maps

- Query:
  - Get people with age = 20 and name = “fred”
- List for age = 20: 1101100000
- List for name = “fred”: 0100000001
- Answer is intersection: 0100000000000

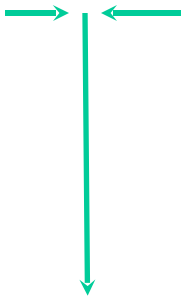
- Good if domain cardinality small
- Bit vectors can be compressed

Join

- "Combine" SALE, PRODUCT relations
- In SQL: SELECT * FROM SALE, PRODUCT

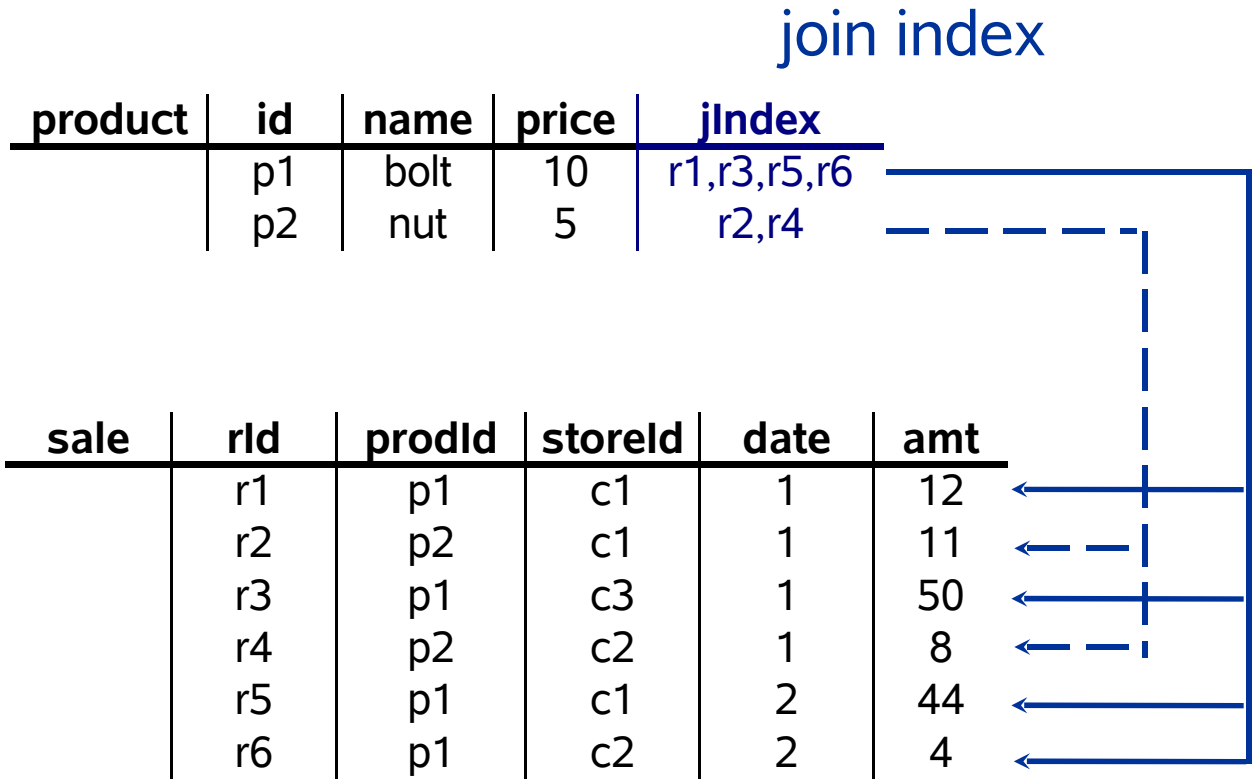
sale	prodId	storeId	date	amt
	p1	c1	1	12
	p2	c1	1	11
	p1	c3	1	50
	p2	c2	1	8
	p1	c1	2	44
	p1	c2	2	4

product	id	name	price
	p1	bolt	10
	p2	nut	5



joinTb	prodId	name	price	storeId	date	amt
	p1	bolt	10	c1	1	12
	p2	nut	5	c1	1	11
	p1	bolt	10	c3	1	50
	p2	nut	5	c2	1	8
	p1	bolt	10	c1	2	44
	p1	bolt	10	c2	2	4

Join Indexes



Current State of Industry

- Extraction and integration done off-line
  - Usually in large, time-consuming, batches
- Everything copied at warehouse
  - Not selective about what is stored
  - Query benefit vs storage & update cost
- Query optimization aimed at OLTP
  - High throughput instead of fast response
  - Process whole query before displaying anything

Future Directions

- Better performance
- Larger warehouses
- Easier to use
- What are companies & research labs working on?

Research (1)

- Incremental Maintenance
- Data Consistency
- Data Expiration
- Recovery
- Data Quality
- Error Handling (Back Flush)

Research (2)

- Rapid Monitor Construction
- Temporal Warehouses
- Materialization & Index Selection
- Data Fusion
- Data Mining
- Integration of Text & Relational Data

Conclusions

- Massive amounts of data and complexity of queries will push limits of current warehouses
- Need better systems:
  - easier to use
  - provide quality information