Data Warehouses

These slides are a modified version of the slides of the book "Database System Concepts" (Chapter 18), 5th Ed., [McGraw-Hill](), by Silberschatz, Korth and Sudarshan.
Original slides are available at [www.db-book.com]()

Decision Support Systems

- Decision-support systems are used to make business decisions, often based on data collected by on-line transaction-processing systems.
- Examples of business decisions:
  - What items to stock?
  - What insurance premium to change?
  - To whom to send advertisements?
- Examples of data used for making decisions
  - Retail sales transaction details
  - Customer profiles (income, age, gender, etc.)

OLTP systems

- Transaction Processing Systems - On-Line Transaction Processing
  - Systems that records information about transactions
  - ACID properties of transactions
  - Organizations accumulate a vast amount of information generated by these systems

OLAP systems

- Decision Support Systems - On-Line Analytical Processing
  - Get high level information out of the detailed information stored in transaction processing systems
  - Large databases
  - Read-only operations. Periodic updates of data

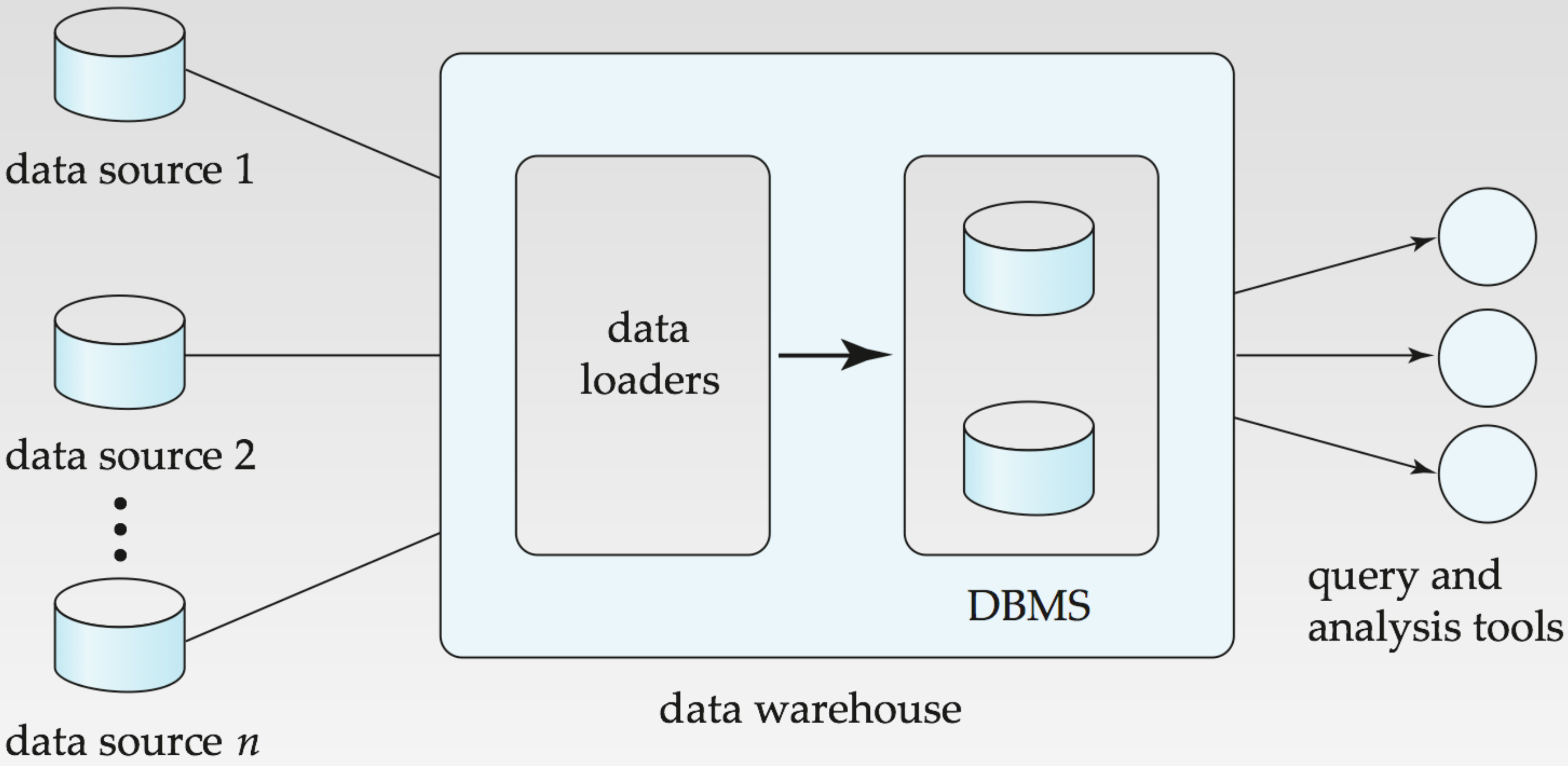
Decision-Support Systems: Overview

- **Data analysis** tasks are simplified by specialized tools and SQL extensions
  - Example tasks
    - For each product category and each region, what were the total sales in the last quarter and how do they compare with the same quarter last year
    - As above, for each product category and each customer category
- A **data warehouse** archives information gathered from multiple sources, and stores it under a unified schema, at a single site.
  - Important for large businesses that generate data from multiple divisions, possibly at multiple sites
  - Data may also be purchased externally
- **Data mining** seeks to discover knowledge automatically in the form of statistical rules and patterns from large databases.
- **Statistical analysis** packages (e.g., : S++) can be interfaced with databases
  - Statistical analysis is a large field, but not covered here

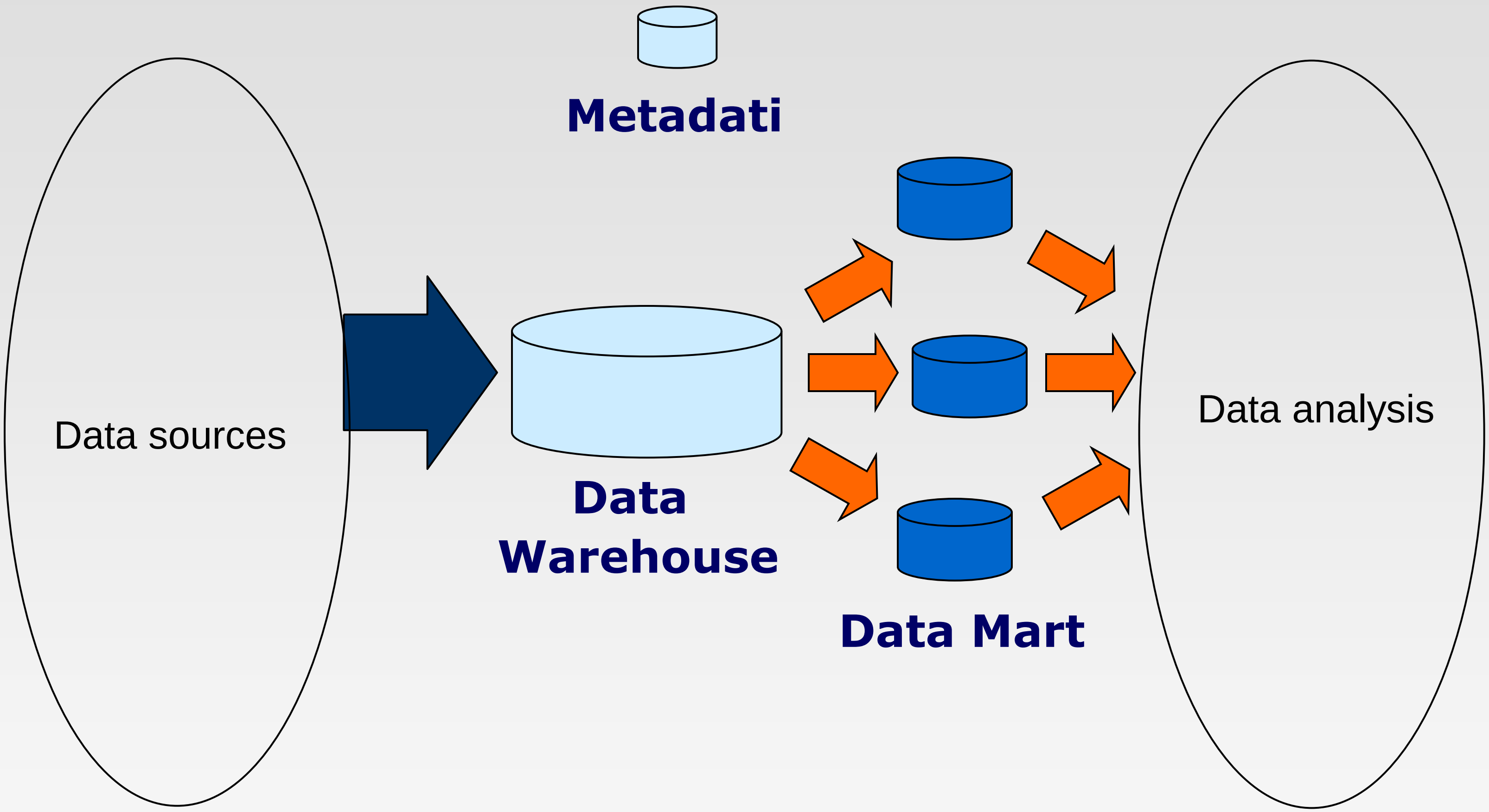
Data Warehousing

- Data sources often store only current data, not historical data
- Corporate decision making requires a unified view of all organizational data, including historical data
- A **data warehouse** is a repository (archive) of information gathered from multiple sources, stored under a unified schema, at a single site
  - Greatly simplifies querying, permits study of historical trends
  - Shifts decision support query load away from transaction processing systems

Data Warehousing

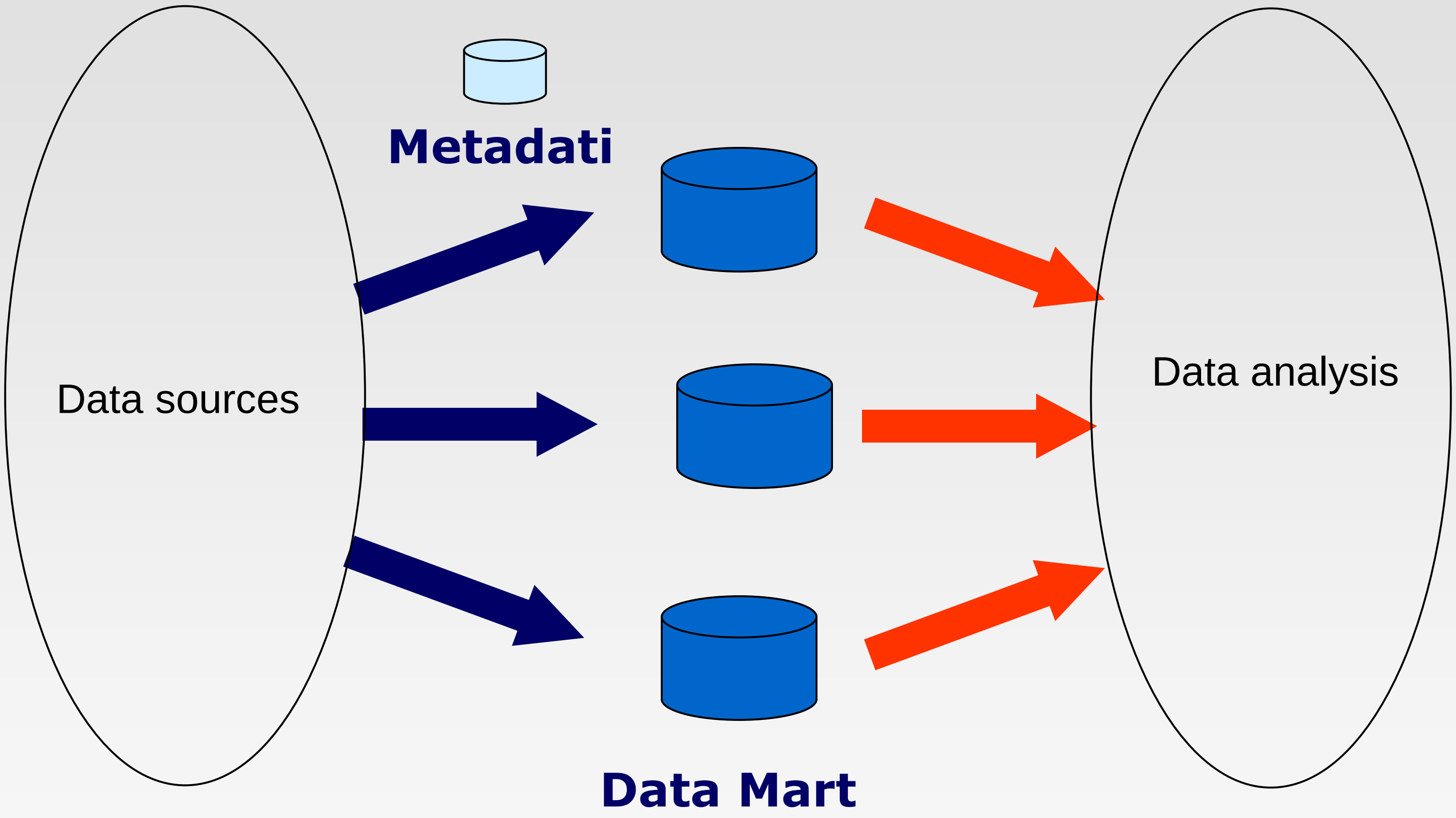


Data warehouse architecture



Metadata: description of data on sources and on DW
Data Mart: logical subset of the data warehouse for a data analysis

Another solution



Sometimes the datawarehouse is too complex, direct representation of Data Mart

Design Issues

- *When and how to gather data*
  - **Source driven architecture**: data sources transmit new information to warehouse, either continuously or periodically (e.g., at night)
  - **Destination driven architecture**: warehouse periodically requests new information from data sources
  - Keeping warehouse exactly synchronized with data sources (e.g., using two-phase commit) is too expensive
    - Usually OK to have slightly out-of-date data at warehouse
    - Data/updates are periodically downloaded form online transaction processing (OLTP) systems.
- *What schema to use*
  - Schema integration

More Warehouse Design Issues

- *Data cleaning*
  - E.g., correct mistakes in addresses (misspellings, zip code errors)
  - **Merge** address lists from different sources and **purge** duplicates
- *How to propagate updates*
  - Warehouse schema may be a (materialized) view of schema from data sources
- *What data to summarize*
  - Raw data may be too large to store on-line
  - Aggregate values (totals/subtotals) often suffice
  - Queries on raw data can often be transformed by query optimizer to use aggregate values

Multidimensional data

Consider an application where a shop wants to find out what kinds of clothes are popular.

- Let us suppose that clothes are characterized by item_name, colour, size and number. We have the relation *sales* with the schema:

  sales(item_name, colour, size, number)

Assume item_name	can take on values	(skirt, dress, shirt, pant)
colour	“ “	(dark, pastel, white)
size	“ “	(small, medium, large)

We can identify some of the attributes as measure attributes, since they measure some value, and can be aggregated upon.

Some other attributes of the relation are identified as dimension attributes, since they define the dimension on which measure attributes

Possible *Dimensions* on which measures are viewed for this application:

- *item information*
- *time*
- *sales location*
- *customer information*

Multidimensional data

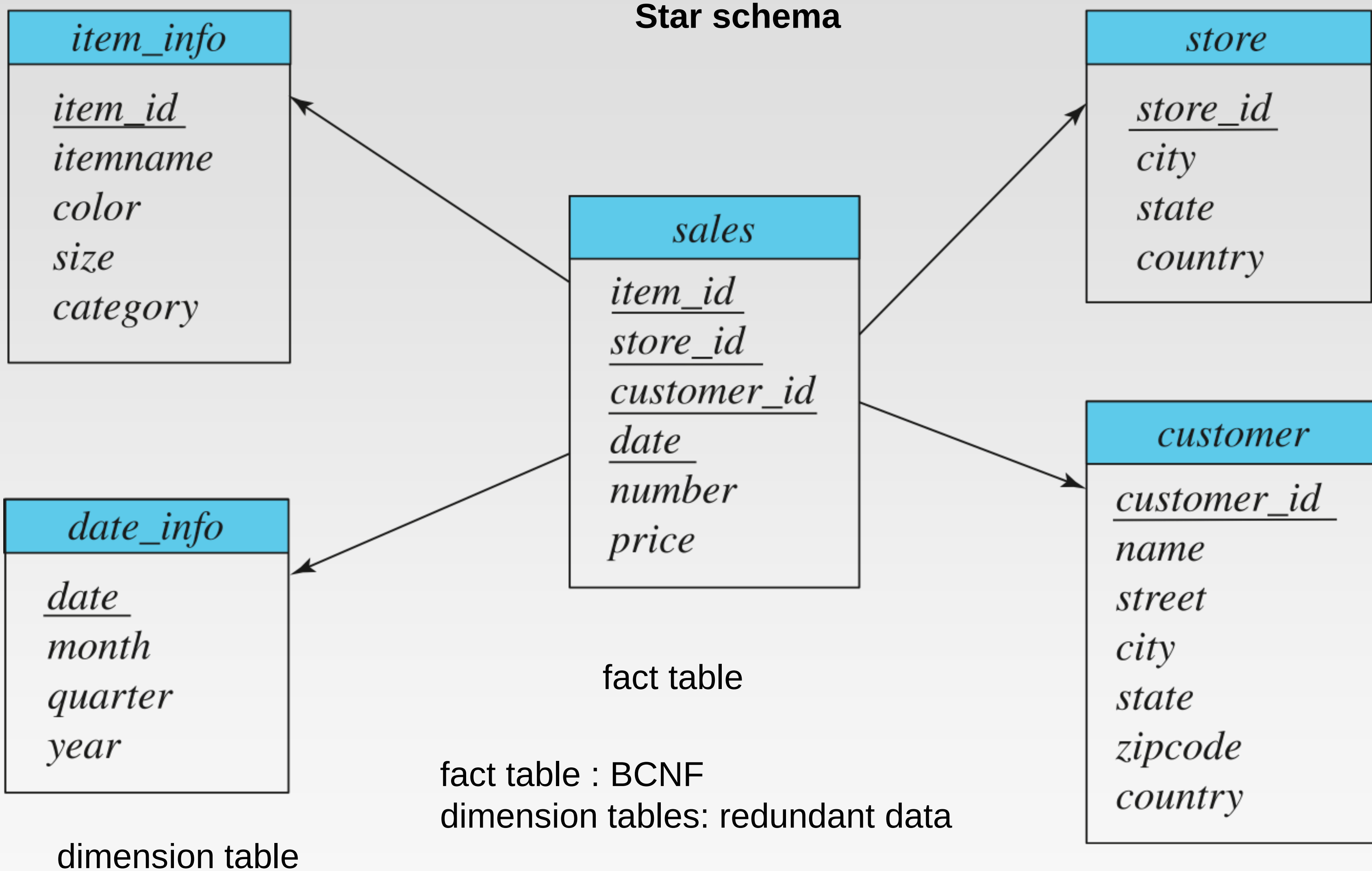
- Data that can be modeled as dimension attributes and measure attributes are called **multidimensional data**.
  - **Measure attributes**
    - measure some value
    - can be aggregated upon
    - e.g. the attribute *number* of the *sales* relation
  - **Dimension attributes**
    - define the dimensions on which measure attributes (or aggregates thereof) are viewed
    - e.g. the attributes *item_name, color,* and *size* of the *sales* relation are dimension attributes for dimension item_info

Warehouse Schemas

- Dimension values are usually encoded using small integers and mapped to full values via dimension tables
- Resultant schema is called a **Star schema**
  - More complicated schema structures
    - **Snowflake schema**: multiple levels of dimension tables
    - **Constellation**: multiple fact tables

Data Warehouse Schema

Star schema



fact table : BCNF
dimension tables: redundant data

Cross Tabulation of *sales* by *item-name* and *color*

A manager may want to see data laid out as number of sales for different combinations of item_info and colour
Size: all indicates that the displayed values are a summary across all values of size.

size: **all**

color

item-name	dark	pastel	white	Total
skirt	8	35	10	53
dress	20	10	5	35
shirt	14	7	28	49
pant	20	2	5	27
Total	62	54	48	164

- The table above is an example of a **cross-tabulation** (**cross-tab**)
  - Values for one of the dimension attributes form the row headers
  - Values for another dimension attribute form the column headers
  - Other dimension attributes are listed on top
  - Values in individual cells are (aggregates of) the values of the dimension attributes that specify the cell.
  - In the example, the aggregation used is the sum of the values for attribute *number* across all size
  - Extra column (row) storing the totals of the cells in the row (column)

Relational Representation of Cross-tabs

- Cross-tabs can be represented as relations
  - We use the value **all** is used to represent aggregates (summary rows or columns)
  - The SQL:1999 standard actually uses null values in place of **all** despite confusion with regular null values

item-name	*color*	*number*
skirt	dark	8
skirt	pastel	35
skirt	white	10
skirt	**all**	53
dress	dark	20
dress	pastel	10
dress	white	5
dress	**all**	35
shirt	dark	14
shirt	pastel	7
shirt	white	28
shirt	**all**	49
pant	dark	20
pant	pastel	2
pant	white	5
pant	**all**	27
all	dark	62
all	pastel	54
all	white	48
all	**all**	164

tuples with value all for colour and size

sum (number)

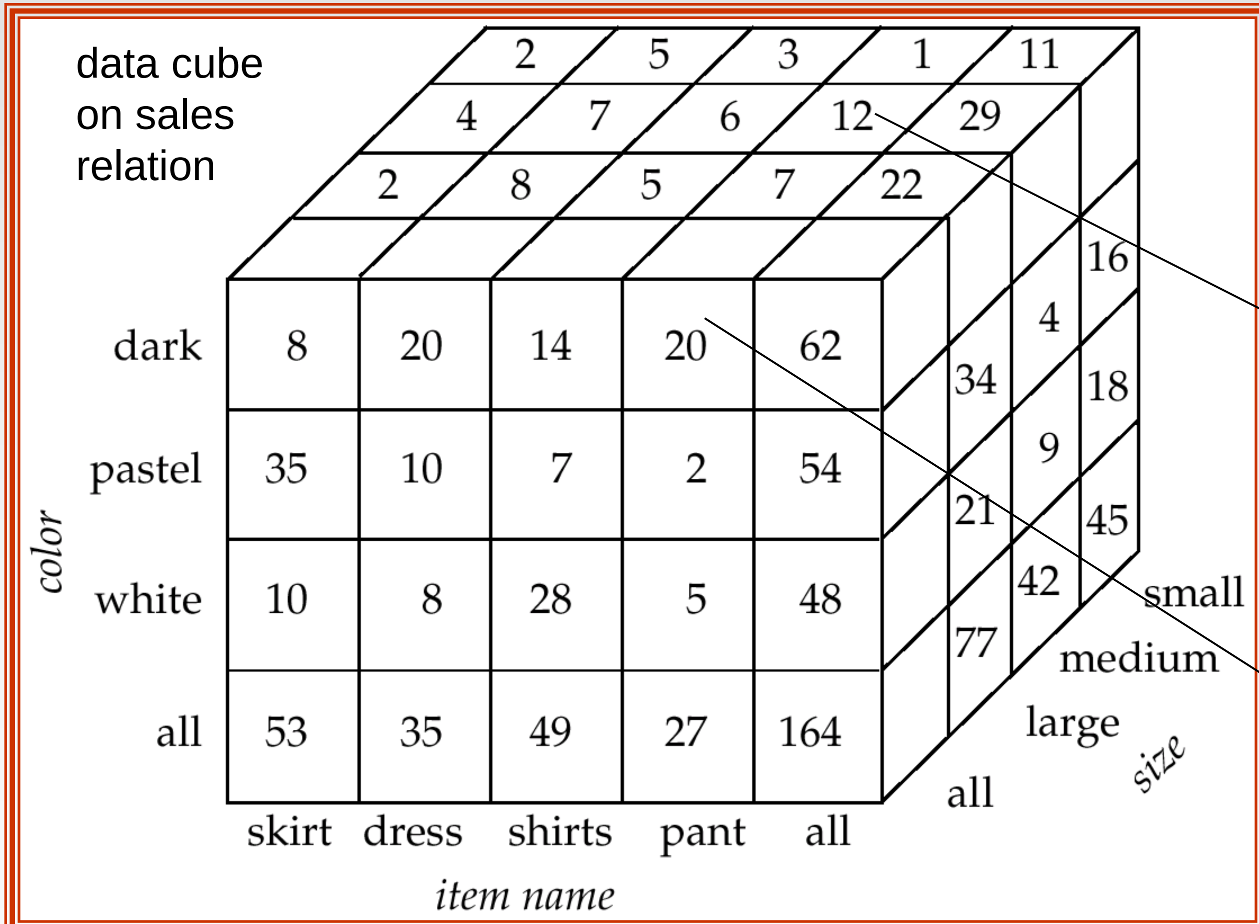
from sales
group by item_name

tuples with value all for item_name, colour and size

sum (number)
from sales

Data Cube

- A **data cube** is a multidimensional generalization of a cross-tab (which is two-dimensional)
- Can have *n* dimensions; we show 3 below (*time_name*, *colour*, *size*)
- Measure attribute is *number*
- Cross-tabs can be used as views on a data cube



- each cell identified by values of the three dimensions

(pant, dark, medium)

- each cell contains a value shown on one of the faces

(pant, dark, all)

summary of all values on that dimension

Online Analytical Processing

- An OLAP system is an interactive system that permits an analyst to view different summaries of multidimensional data

- Online indicates that an analyst must be able to request new summaries and get responses on line, and should not be forced to wait for a long time.
- An analyst can look at different cross-tabs on the same data by selecting the attributes in the cross-tab.
- Each cross-tab is a two-dimensional view of a multidimensional data cube

  - Pivoting

  - Slicing

  - Rollup

  - Drill down

Online Analytical Processing

- **Pivoting:** the operation of changing the dimensions used in a cross-tab

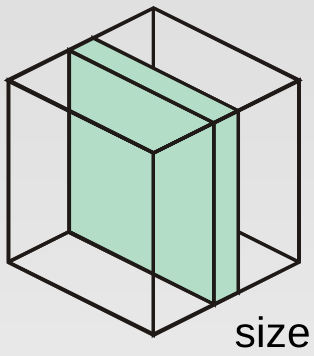
size: **all**

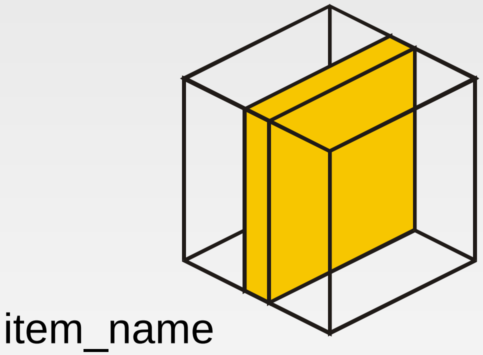
color

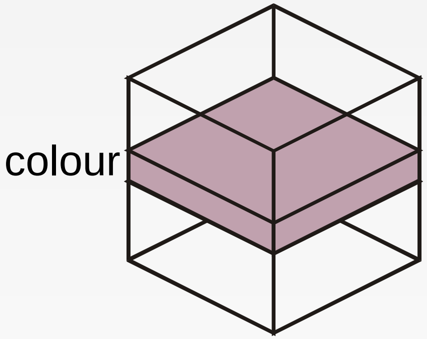
item-name		dark	pastel	white	Total
	skirt	8	35	10	53
	dress	20	10	5	35
	shirt	14	7	28	49
	pant	20	2	5	27
	Total	62	54	48	164

Online Analytical Processing

- **Slicing:** creating a cross-tab for fixed values only
  - cross-tab for *item_name* and *colour* for a fixed value of *size*
  - cross-tab for *colour* and *size* for a fixed value of *item_name*
  - cross-tab for *item_name* and *size* for a fixed value of *colour*

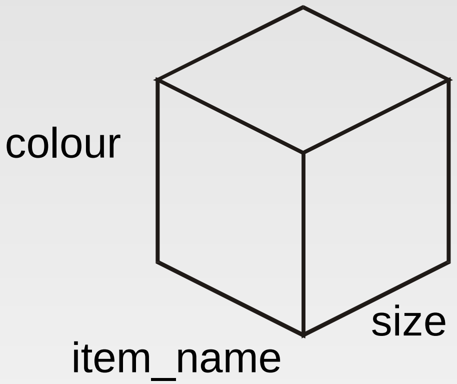


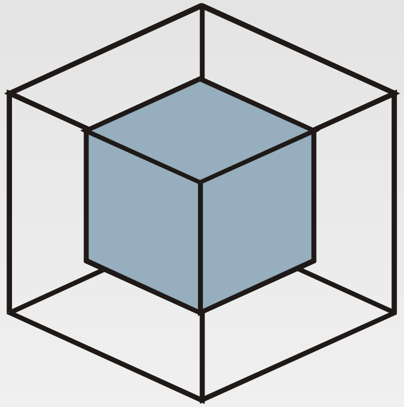




Online Analytical Processing

- **Dicing :** slicing, when values for multiple dimensions are fixed.





subset of item_names

subset of colours

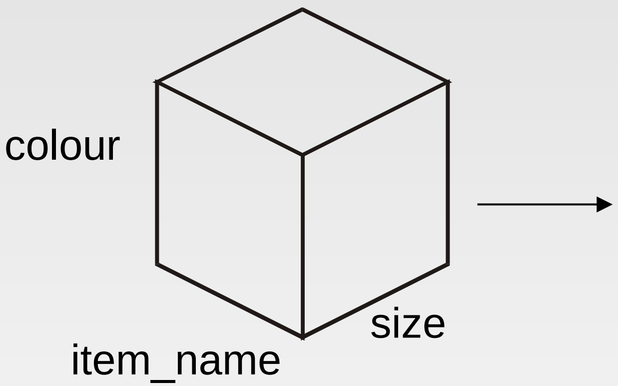
subset of size

Online Analytical Processing

OLAP systems permit users to view data at any desired level of granularity

- **Rollup:** moving from finer-granularity data to a coarser granularity by means of aggregation

  Starting from the data cube on the sales table, we got cross-tab by rolling up on size



size: **all**

color

item-name

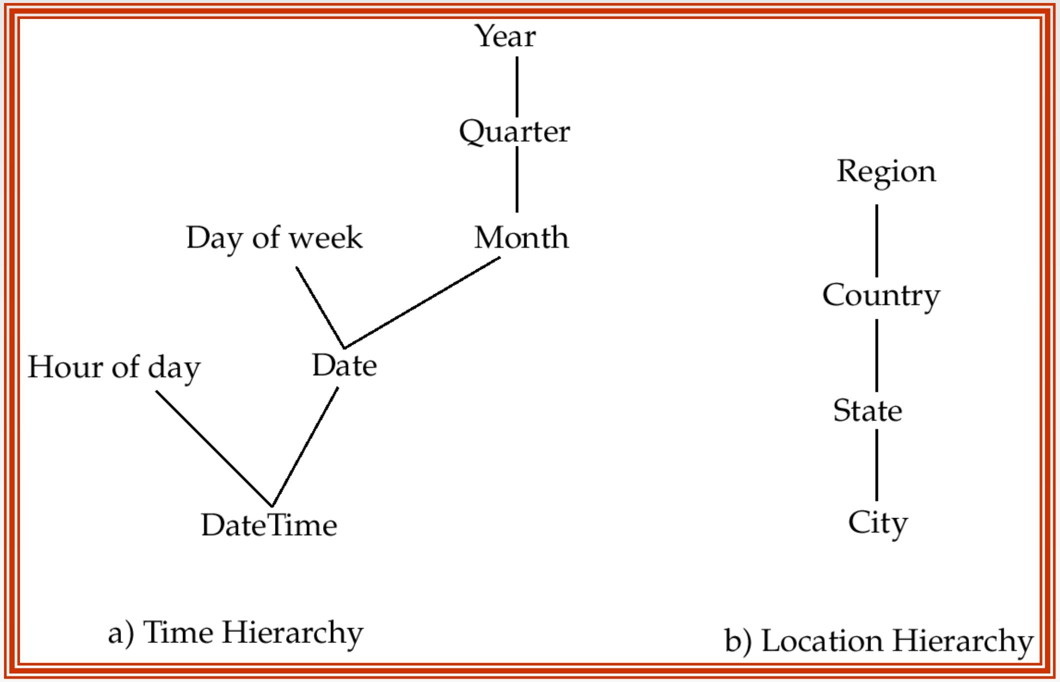
	dark	pastel	white	Total
skirt	8	35	10	53
dress	20	10	5	35
shirt	14	7	28	49
pant	20	2	5	27
Total	62	54	48	164

- **Drill down:** The opposite operation - that of moving from coarser-granularity data to finer-granularity data
  - Finer granularity cannot be generated by coarser granularity data; Must be generated either from the original data or from ever finer granularity summary data

Hierarchies on Dimensions

Analysts may wish to view a dimension at different levels of details

- **Hierarchy** on dimension attributes: lets dimensions to be viewed at different levels of detail
  - E.g. the dimension DateTime can be used to aggregate by hour of day, date, day of week, month, quarter or year



Cross Tabulation With Hierarchy on item-name

- Cross-tabs can be easily extended to deal with hierarchies
  - Can drill down or roll up on a hierarchy

Assume Clothes are grouped by *category* (menswear and womenswear): category lies above item_name

category	*item-name*					
		dark	pastel	white	total	
womenswear	skirt	8	8	10	53	
	dress	20	20	5	35	
	subtotal	28	28	15		88
menswear	pants	14	14	28	49	
	shirt	20	20	5	27	
	subtotal	34	34	33		76
total		62	62	48		164

Different levels are shown in the same cross-tab

OLAP Implementation

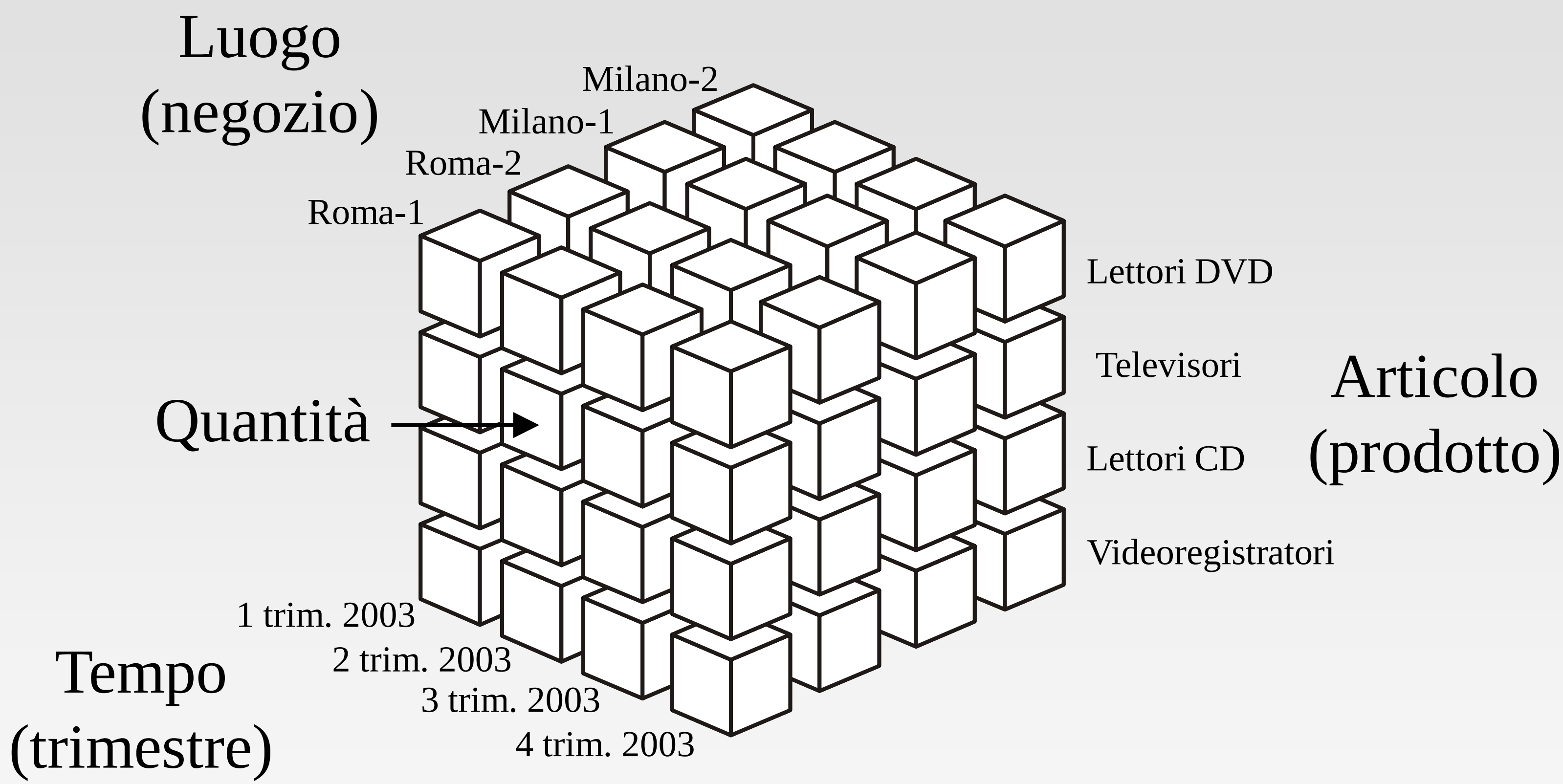
- The earliest OLAP systems used multidimensional arrays in memory to store data cubes, and are referred to as **multidimensional OLAP (MOLAP)** systems.
- OLAP implementations using only relational database features are called **relational OLAP (ROLAP)** systems
- Hybrid systems, which store some summaries in memory and store the base data and other summaries in a relational database, are called **hybrid OLAP (HOLAP)** systems.

OLAP Implementation (Cont.)

- Early OLAP systems precomputed *all* possible aggregates in order to provide online response
  - Space and time requirements for doing so can be very high
    - 2^n combinations of **group by**
  - It suffices to precompute some aggregates, and compute others on demand from one of the precomputed aggregates
    - Can compute aggregate on (*item-name, color*) from an aggregate on (*item-name, color, size*)
      - is cheaper than computing it from scratch
- Several optimizations available for computing multiple aggregates
  - Can compute aggregates on (*item-name, color, size*), (*item-name, color*) and (*item-name*) using a single sorting of the base data

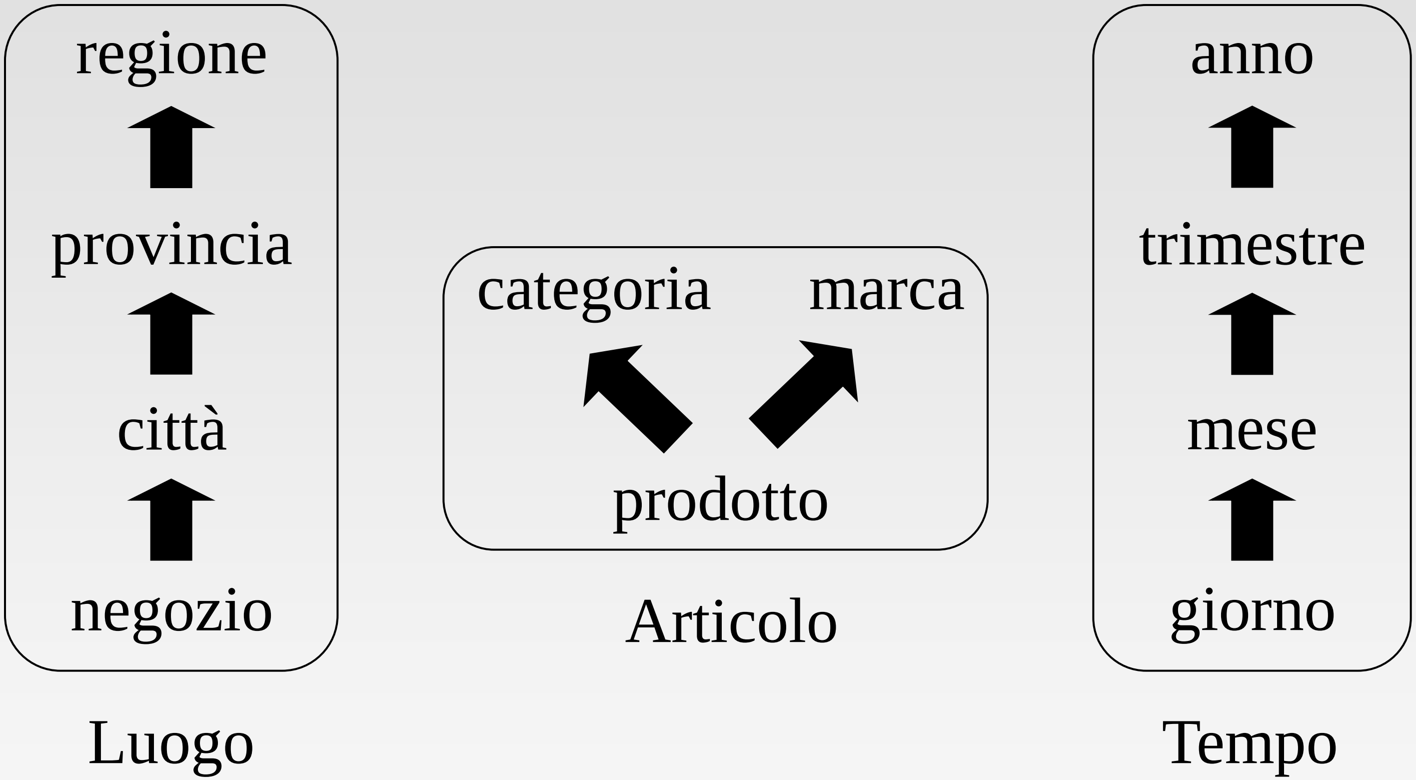
An example

Basi di dati: Architetture e linee di evoluzione
Atzeni, Ceri, Fraternali, Paraboschi, Torlone,
Mc GrawHill

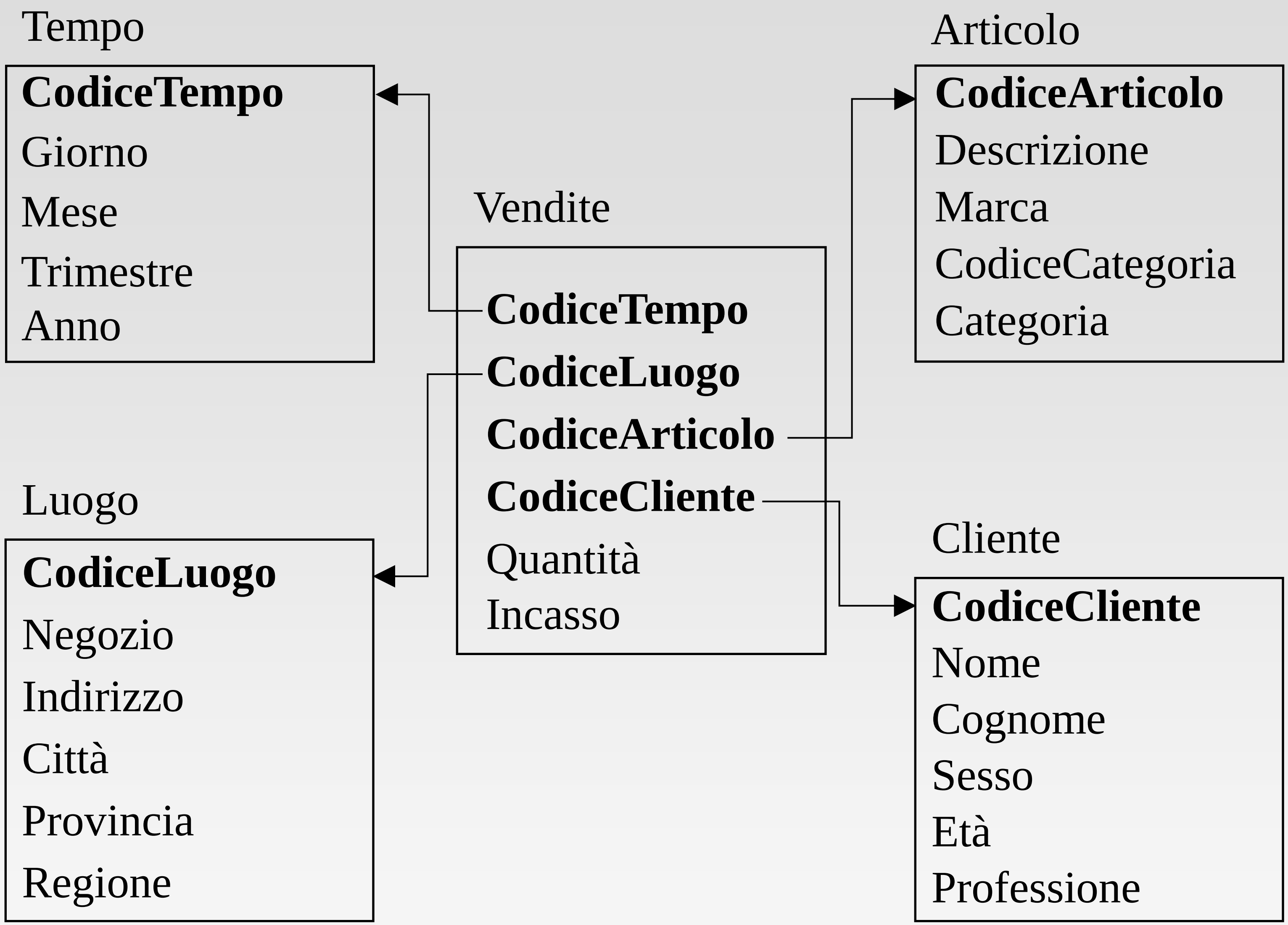


Fatto = vendita Misura = quantità Dimensione = Tempo, Articolo, Luogo

Dimensioni e gerarchie di livelli



Basi di dati: Architetture e linee di evoluzione
Atzeni, Ceri, Fraternali, Paraboschi, Torlone,
Mc GrawHill



Schema a stella

Basi di dati: Architetture e linee di evoluzione
Atzeni, Ceri, Fraternali, Paraboschi, Torlone,
Mc GrawHill

LUOGO

CodL	Negozio	Indirizzo	Città	Prov	Regione
L1	Roma1	Via Po, 3	Roma	RM	Lazio
L2	Roma2	P. Navona, 8	Roma	RM	Lazio
L3	Milano1	P. Duomo, 7	Milano	MI	Lombardia
...	...	...	...	...	...
L100	NULL	NULL	Roma	RM	Lazio
L101	NULL	NULL	Milano	MI	Lombardia
...	...	...	...	...	...
L1000	NULL	NULL	NULL	RM	Lazio
...	...	...	...	...	...

ARTICOLO

CodA	Prod	Marca	CodCat	Categoria
...	...	...	...	...
P43	KV21	Sony	TEL	Televisore
...	...	...	...	...
P377	NULL	NULL	LDV	Lettori DVD
...	...	...	...	...

VENDITE

CodA	CodT	CodL	CodC	Quant	Inc
...	...	...	...	...	...
P43	T99	L3	C79	1	2K
...	...	...	...	...	...

VENDITETRIMESTRALI

CodP	CodT	CodL	Quant
...	...	...	...
P377	T504	L2	219
...	...	...	...

TEMPO

CodT	Giorno	Mese	Trimestre	Anno
...	...	...	...	...
T99	1 Apr 03	Apr-03	2 trim. 03	2003
...	...	...	...	...
T504	NULL	NULL	2 trim. 03	2003
...	...	...	...	...

CLIENTE

CodC	Nome	Cognome	Sesso	Età	Professione
...	...	...	...	...	...
C79	Maria	Neri	F	30	Impiegata
...	...	...	...	...	...

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.

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*)

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.*

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*), ( ) }

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

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*

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*

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

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*