Data warehouse
Data analysis

Elena Baralis
Politecnico di Torino

Data analysis

- OLAP analysis: complex aggregate function computation
  - support to different types of aggregate functions (e.g., moving average, top ten)
- Comparison operations, exploited to compare business trends (example: sale figure comparison for different time periods)
  - difficult by exploiting plain SQL
- Data analysis by means of data mining techniques

User interface

Users may query the data warehouse by means of various tools:

- controlled query environments
- query and report generation tools
- data mining tools

Controlled query environment

- It encompasses
  - complex queries with predefined structure (usually parametric)
  - ad hoc analysis procedures
  - predefined reports
- Techniques and knowledge of a specific economic area may be exploited
- It requires ad hoc code development
  - stored procedures, application packages, predefined joins and aggregations
  - flexible tools for report management are available, which allow defining
    - report layout
    - publication periodicity
    - distribution list

Ad hoc query environment

- Arbitrary OLAP queries may be defined
- Queries are designed on demand by users
  - query is defined by point and click techniques, which automatically generate SQL instructions
  - (typically) complex queries may be defined
  - spreadsheet is the user interface paradigm
- An OLAP session allows successive refinements of the same query
- Used when predefined reports are not enough

OLAP analysis

Elena Baralis
Politecnico di Torino

OLAP analysis

- Available query operations
  - roll up, drill down
  - slice and dice
  - (table) pivot
  - sorting
- Operations may be
  - used together in the same query
  - exploited in sequence to refine the same query which builds up the OLAP session

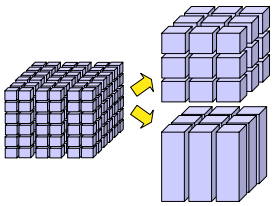
Roll up

- Data detail reduction by
  - decreasing detail in a dimension, by climbing up a hierarchy
    - example

      group by store, month → group by city, month
  - dropping a whole dimension
    - example

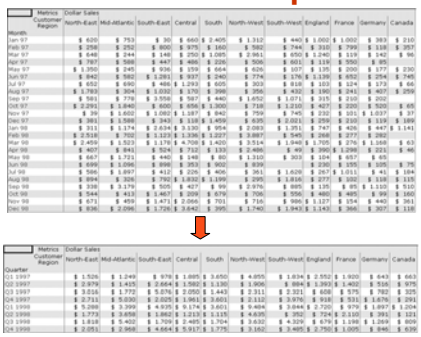
      group by product, city → group by product

Roll up



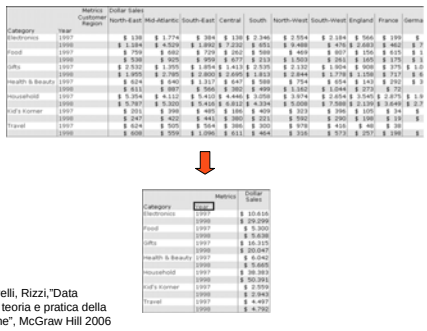
From Golfarelli, Rizzi,"Data warehouse, teoria e pratica della progettazione", McGraw Hill 2006

Roll up



From Golfarelli, Rizzi,"Data warehouse, teoria e pratica della progettazione", McGraw Hill 2006

Roll up



From Golfarelli, Rizzi,"Data warehouse, teoria e pratica della progettazione", McGraw Hill 2006

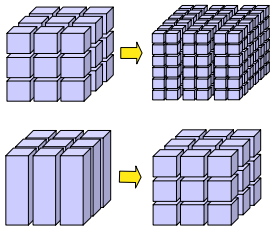
Drill down

- Data detail increase by
  - increasing detail in a dimension, by walking down a hierarchy
    - example

      group by city, month → group by store, month
  - adding a whole dimension
    - example

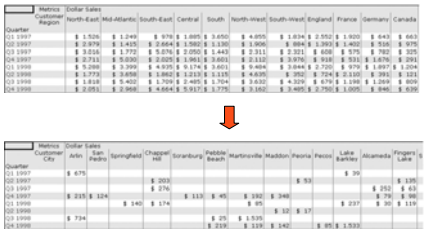
      group by product → group by product, city
- Frequently drill down operates on a subset of data produced by the initial query

Drill down



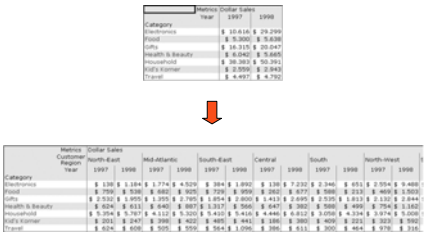
From Golfarelli, Rizzi,"Data warehouse, teoria e pratica della progettazione", McGraw Hill 2006

Drill down



From Golfarelli, Rizzi,"Data warehouse, teoria e pratica della progettazione", McGraw Hill 2006

Drill down

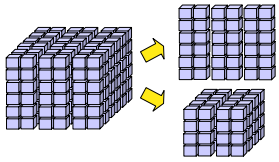


From Golfarelli, Rizzi,"Data warehouse, teoria e pratica della progettazione", McGraw Hill 2006

Slice and dice

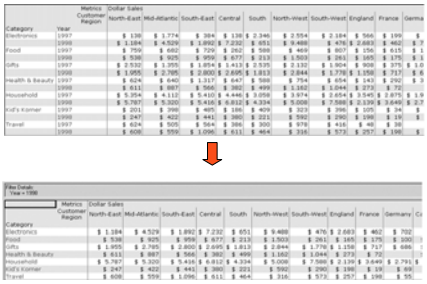
- Selection of a data subset by means of selection predicates
  - slice: equality predicate selecting a “slice”
    - example: Year=2005
  - dice: predicate expression selecting a “dice”
    - example: Category=‘Food’ and City=‘Torino’

Slice and dice



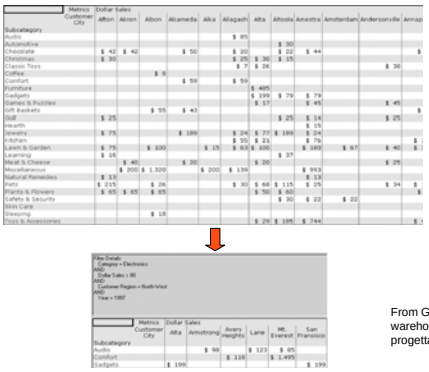
From Golfarelli, Rizzi,"Data warehouse, teoria e pratica della progettazione", McGraw Hill 2006

Slice and dice



From Golfarelli, Rizzi,"Data warehouse, teoria e pratica della progettazione", McGraw Hill 2006

Slice and dice

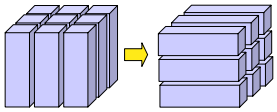


From Golfarelli, Rizzi,"Data warehouse, teoria e pratica della progettazione", McGraw Hill 2006

Pivot

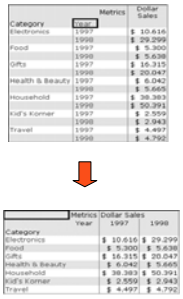
- Reorganization of the multidimensional structure without varying the detail level
  - increases readability of the same information
  - multidimensional representation is always based on a "grid" (hierarchical spreadsheet)
    - two dimensions are the main grid axes
    - position of dimensions in the grid are changed

Pivot



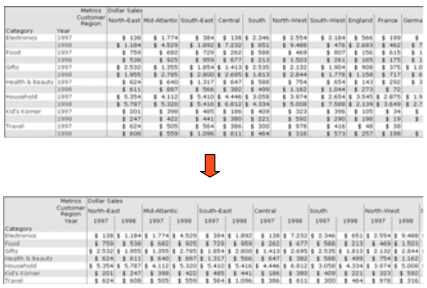
From Golfarelli, Rizzi,"Data warehouse, teoria e pratica della progettazione", McGraw Hill 2006

Pivot



From Golfarelli, Rizzi,"Data warehouse, teoria e pratica della progettazione", McGraw Hill 2006

Pivot



From Golfarelli, Rizzi,"Data warehouse, teoria e pratica della progettazione", McGraw Hill 2006

Extensions of the SQL language

Elena Baralis
Politecnico di Torino

Extensions of the SQL language

- Interface tools require
  - new aggregate functions
    - aggregate functions exploited for economic analysis (moving average, median, ...)
    - position in the sort order (i.e., rank)
  - functions for report generation
    - partial and cumulative totals
- New OLAP functions in the ANSI standard
  - implemented starting from DB2 UDB 7.1, Oracle 8i v2

Extensions of the SQL language

- Interface tools require
  - operators for the computation of different group bys at the same time
- The SQL-99 (SQL3) standard has extended the SQL group by clause

Example data base

Sales(City,Month,Amount)

City	Month	Amount
Milano	7	110
Milano	8	10
Milano	9	70
Milano	10	90
Milano	11	35
Milano	12	135
Torino	7	70
Torino	8	35
Torino	9	80
Torino	10	95
Torino	11	50
Torino	12	120

SQL OLAP functions

- New class of aggregate functions (OLAP functions) characterized by
  - computation window, inside which the computation of aggregate functions is performed
    - cumulative totals and moving average can be computed
  - new aggregate functions to compute the rank in a given sort order

Computation window

- New **`window`** clause, characterized by
  - *partitioning*: Rows are grouped without collapsing them (different from **`group by`**)
    - no partitioning: a single group is defined
  - *row ordering*, separately in each partition (similar to **`order by`**)
  - *aggregation window*: For each row in the partition, it defines the row group on which the aggregate function is computed

Example

- Show, for each city and month
  - sale amount
  - average on the current month and the two previous months, separately for each city

Example

- Partitioning on city
  - average computation is reset when the city changes
- Ordering by month, to compute the moving average on the current month and the two preceding months
  - without ordering the computation is meaningless
- Aggregation window size: the current row and the two preceding rows

Example

```
SELECT City, Month, Amount,
       AVG(Amount) OVER Wavg AS MovingAvg
FROM Sales
WINDOW Wavg AS (PARTITION BY City
                ORDER BY Month
                ROWS 2 PRECEDING)
```


Example

```
SELECT City, Month, Amount,
       AVG(Amount) OVER (PARTITION BY City
                         ORDER BY Month
                         ROWS 2 PRECEDING)
       AS MovingAvg
FROM Sales
```


Result

City	Month	Amount	MovingAvg	
Milano	7	110	110	Partition 1
Milano	8	10	60	
Milano	9	90	70	
Milano	10	80	60	
Milano	11	40	60	
Milano	12	140	90	
Torino	7	70	70	Partition 2
Torino	8	30	50	
Torino	9	80	60	
Torino	10	100	70	
Torino	11	50	60	
Torino	12	150	100	

Observations

- Sort order is required, because the computation of the moving average considers rows in an ordered fashion
  - the window sort order does not enforce a predefined output sort order
- When the window is not complete, the computation takes place on the available rows
  - it is possible to require a **NULL** result for each incomplete window
- Several different computation windows may be specified

Aggregation window

- The moving window on which the aggregate function is computed may be defined
  - at the *physical level*: It builds the group by counting rows
    - example: the current row and the two preceding rows
  - at the *logical level*: It builds the group by defining an interval on the sort key
    - example: the current month and the two preceding months

Physical interval definition

- Between a lower bound and the current row

```
ROWS 2 PRECEDING
```

- Between lower and upper bounds

```
ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING
ROWS BETWEEN 3 PRECEDING AND 1 PRECEDING
```

- Between the beginning (or the end) of a partition and the current row

```
ROWS UNBOUNDED PRECEDING (o FOLLOWING)
```


Physical interval

- Appropriate for sequence data with no gaps
  - example: no month is missing in the sequence
  - more than a sort key can be specified
    - computation ignores breaks due to change in any sort key value
    - example: order by month and year
  - no mathematical expressions are needed to compute the window

Logical interval definition

- The **range** clause is used, with the same syntax as the physical interval
- A distance on the sort key between the interval bounds and the current value should be defined
- Example

```
RANGE 2 MONTH PRECEDING
```


Logical interval

- Appropriate for “sparse” data, with gaps in the sequence
  - example: a month is missing in the sequence
  - only a single sort key can be specified
  - the sort key can only be alphanumeric or date type (arithmetic expressions are allowed)

Applications

- Moving aggregate computations
  - computations on a window which moves over data
  - examples: moving average, moving sum
- Cumulative total computations
  - the (cumulative) total is incremented by adding an instance at a time
- Comparison between detailed data and aggregated data

Computation of a cumulative total

- Show, for each city and month
  - sale amount
  - cumulative sale amount for increasing months, separately for each city

Computation of a cumulative total

- Partition by city
  - the cumulative total is reset when the city changes
- Order by (ascending) month to compute the sum for increasing months
  - without sorting, the computation would be meaningless
- Size of the aggregation window
  - from the starting row of the partition to the current row

Computation of a cumulative total

```
SELECT City, Month, Amount,
       SUM(Amount) OVER (PARTITION BY City
                         ORDER BY Month
                         ROWS UNBOUNDED PRECEDING)
       AS CumeTot
FROM Sales
```


Computation of a cumulative total

City	Month	Amount	CumeTot	
Milano	7	110	110	Partition 1
Milano	8	10	120	
Milano	9	90	210	
Milano	10	80	290	
Milano	11	40	330	
Milano	12	140	470	
Torino	7	70	70	Partition 2
Torino	8	30	100	
Torino	9	80	180	
Torino	10	100	280	
Torino	11	50	330	
Torino	12	150	480	

Comparison between detailed data and total data

- Show, for each city and month
  - sale amount
  - total sale amount on the whole time period for the current city

Comparison between detailed data and total data

- Partition by city
  - the total amount is reset when the city changes
- Sorting is not needed
  - the total amount is computed independently of the sort order of tuples
- The aggregation window is not needed
  - it is the whole partition

Comparison between detailed data and total data

```
SELECT City, Month, Amount,
       SUM(Amount) OVER (PARTITION BY City)
       AS TotalAmount
FROM Sales
```

Comparison between detailed data and total data

City	Month	Amount	TotalAmount
Milano	7	110	470
Milano	8	10	470
Milano	9	90	470
Milano	10	80	470
Milano	11	40	470
Milano	12	140	470
Torino	7	70	480
Torino	8	30	480
Torino	9	80	480
Torino	10	100	480
Torino	11	50	480
Torino	12	150	480

Partition 1

Partition 2

Comparison between detailed data and total data

- Show, for each city and month
  - sale amount
  - ratio between current row amount and grand total
  - ratio between current row amount and total amount by city
  - ratio between current row amount and total amount by month

Comparison between detailed data and total data

- Three different computation windows
  - grand total: no partitioning
  - total by city: partition by city
  - total by month: partition by month
- No sort is needed in any window
  - totals are independent of the sort order of tuples
- The aggregation window is always the whole partition

Comparison between detailed data and total data

```
SELECT City, Month, Amount
  Amount/SUM(Amount) OVER ()
  AS TotalFract
  Amount/SUM(Amount) OVER (PARTITION BY City)
  AS CityFract
  Amount/SUM(Amount) OVER (PARTITION BY Month)
  AS MonthFract
FROM Sales
```


Comparison between detailed data and total data

City	Month	Amount	TotalFract	CityFract	MonthFrct
Milano	7	110	110/950	110/470	110/180
Milano	8	10	10/950	10/470	10/40
Milano	9	90	90/950	90/470	90/170
Milano	10	80	80/950	80/470	80/180
Milano	11	40	40/950	40/470	40/90
Milano	12	140	140/950	140/470	140/290
Torino	7	70	70/950	70/480	70/180
Torino	8	30	30/950	30/480	30/40
Torino	9	80	80/950	80/480	80/170
Torino	10	100	100/950	100/480	100/180
Torino	11	50	50/950	50/480	50/90
Torino	12	150	150/950	150/480	150/290

`Group by` and `window`

- Windows can be used together with grouping performed by **`group by`**
- The “temporary table” generated by the execution of the **`group by`** clause (possibly with aggregate function computation) becomes the operand to which the computations in the **`window`** clause are applied

Example

- Assume that the **Sales** table contains information on sales with daily granularity
- Show, for each city and month
  - sale amount
  - average sale with respect to the current month and the two preceding months, separately for each city

Example

- Grouping by month is needed to compute the total amount by month before computing the moving average
  - the group by clause is used for computing the monthly total
- The temporary table generated by the group by computation is the operand on which the computation window is defined

Example

```
SELECT City, Month, SUM(Amount) AS TotMonth,
       AVG(SUM(Amount)) OVER (PARTITION BY City
                              ORDER BY Month
                              ROWS 2 PRECEDING)
       AS MovingAvg
FROM Sales
WHERE <join conditions>
GROUP BY City, Month
```


Ranking functions

- Functions computing the rank of a value inside a partition
  - **rank()** function: computes the rank by leaving an empty slot after a tie
    - example: after 2 first, the next rank is third
  - **denserank()** function: computes the rank by leaving an empty slot after a tie
    - example: after 2 first, the next rank is second

Example

- Show, for each city in december
  - sale amount
  - rank on amount

Example

- Partitioning is not needed
  - a single partition including all cities
- Order by amount to perform ranking
  - without sorting, the computation would be meaningless
- The aggregation window is the whole partition

Example

```
SELECT City, Amount,
       RANK() OVER (ORDER BY Amount DESC)
       AS Ranking
FROM Sales
WHERE Month = 12
```


Result

City	Amount	Ranking
Torino	150	1
Milano	140	2

Sorting the result

- A sorted result is obtained by means of the **order by** clause
  - may be different from the sort order in the computation window
- Example: sort the result in the former example by increasing city

Example

```
SELECT City, Amount,
       RANK() OVER (ORDER BY Amount DESC)
       AS Ranking
FROM Sales
WHERE Month = 12
ORDER BY City
```

City	Amount	Ranking
Milano	140	2
Torino	150	1

group by clause extensions

- Multidimensional spreadsheets compute several partial totals “in one shot”
  - total sale amount by month and city
  - total sale amount by month
  - total sale amount by city
- For the sake of efficiency avoid
  - multiple data reads
  - redundant data sorts

group by clause extensions

- SQL-99 standard extended the syntax of the **group by** clause
  - **rollup** computes aggregations on all groups obtained by removing one by one the columns in the grouping clause
  - **cube** computes aggregations on all combinations of the columns in the grouping clause
  - **grouping sets** computes aggregations on the group list in the grouping clause (grouping sets different from the previous clauses may be specified)
    - **()** for grand totals (no grouping)

Rollup: example

- Consider the following tables
  - **Time(Tkey,Day,Month,Year,…)**
  - **Shop(Skey,City,Region,…)**
  - **Product(Pkey,PName,Brand,…)**
  - **Sales(Skey,Tkey,Pkey,Amount)**
- Compute total sales in the year 2000 for the following attribute combinations
  - product, month, city
  - month, city
  - city

Rollup: example

```
SELECT City, Month, Pkey,
       SUM(Amount) AS TotSales
FROM Time T, Shop S, Sales V
WHERE T.Tkey = V.Tkey
  AND S.Skey = V.Skey
  AND Year = 2000
GROUP BY ROLLUP (City,Month,Pkey)
```

- The column sort order in **rollup** determines which aggregates are computed

Rollup: result

City	Month	Pkey	TotSales
Milano	7	145	110
Milano	7	150	10
Milano	…	…	…
Milano	7	**NULL**	8500
Milano	8	…	…
Milano	**NULL**	**NULL**	150000
Torino	…	…	150
Torino	…	**NULL**	2500
Torino	**NULL**	**NULL**	135000
…	…	…	…
NULL	**NULL**	**NULL**	25005000

- "Superaggregates" are represented by **NULL**

Cube: example

- Compute total sales in the year 2000 for *all* combinations of the following attributes
  - product, month, city
- The following aggregations should be computed
  - product, month, city
  - product, month
  - month, city
  - product, city
  - product
  - month
  - city
  - no grouping

Cube: example

```
SELECT City, Month, Pkey,
       SUM(Amount) AS TotSales
FROM Time T, Shop S, Sales V
WHERE T.Tkey = V.Tkey
  AND S.Skey = V.Skey
  AND Year = 2000
GROUP BY CUBE (City,Month,Pkey)
```

- The sort order of columns in **cube** is irrelevant

Cube computation

- Consider distributive and algebraic properties of aggregate functions
  - *distributive* aggregate functions (**min**, **max**, **sum**, **count**) may be computed from aggregations on a larger set of attributes (i.e., with larger granularity)
    - Example: from total sales by product and month, total sales by month may be computed
  - algebraic aggregate functions (**avg**, …) may be computed from aggregations on a larger set of attributes (i.e., with larger granularity), if appropriate support aggregations are stored
    - Example: average requires
      - the average value in the group
      - the cardinality of the group

Cube computation

- To increase the efficiency of cube computation, the distributive/algebraic properties of the aggregate functions are exploited
  - previously computed **`group by`** are exploited
  - **`rollup`** requires a single sort operation
  - the cube is a combination of several **`rollup`** operations (in the appropriate order)
  - previously executed sort operations are exploited (also partially)
    - it is possible to exploit sort on (A,B) to sort by (A,C)

Grouping Set: example

- Compute total sales in the year 2000 for the following groups
  - month
  - month, city, product
- A roll up would perform the computation of unnecessary groupings and aggregations

Grouping Set: example

```
SELECT City, Month, Pkey,
       SUM(Amount) AS TotSales
FROM Time T, Shop S, Sales S
WHERE T.Tkey = S.Tkey
  AND S.Skey = S.Skey
  AND Year = 2000
GROUP BY GROUPING SETS
      (Month, (City,Month,Pkey))
```