Business intelligence

Unit 3 – Data exploitation. Query languages and visualization
S3-1 – OLAP

OLAP tools

- OLAP tools provide the user with a multidimensional view of data (multidimensional schema) for each activity that is being analyzed.
- The user formulates queries to the OLAP tool selecting multidimensional attributes of this scheme without knowing the internal structure (physical schema) of the data warehouse.
- The tool generates a corresponding OLAP query and sends it to the query management system (eg by means a SQL SELECT statement).

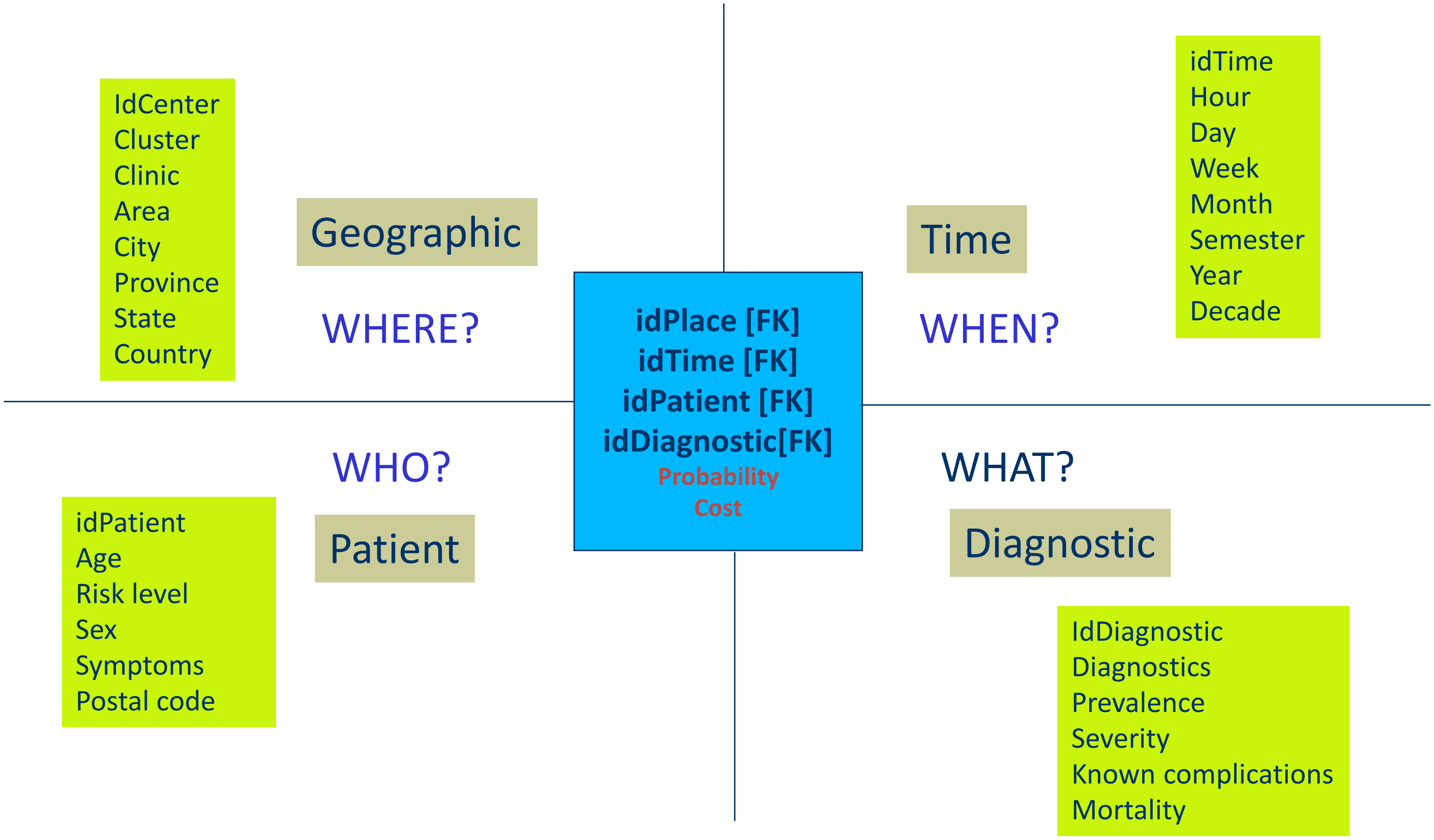
OLAP tools

- Query resolution procedure:
  - Build the query
  - Extract aggregated data
  - Visualize results
  - Analyze

OLAP query

- An OLAP query consists of
  - Retrieve measures or indicators
  - About the facts
  - parametrized by attributes in the dimensions
  - Constrained by conditions imposed on the dimensions
  - Eg: What is the total cost per diagnostic with low mortality rate in the last year for each province and sex?

Problem


IdCenter
Cluster
Clinic
Area
City
Province
State
Country
Geographic
WHERE?
Time
WHEN?
idTime
Hour
Day
Week
Month
Semester
Year
Decade
idPlace [FK]
idTime [FK]
idPatient [FK]
idDiagnostic[FK]
Probability
Cost
WHO?
Patient
idPatient
Age
Risk level
Sex
Symptoms
Postal code
WHAT?
Diagnostic
IdDiagnostic
Diagnostics
Prevalence
Severity
Known complications
Mortality

OLAP Cube

Fact tables

Diagnostic	**Sex**	**Total**
D1	M	100
D1	F	200
D2	M	150
D2	F	75

$\longleftrightarrow$

2D view

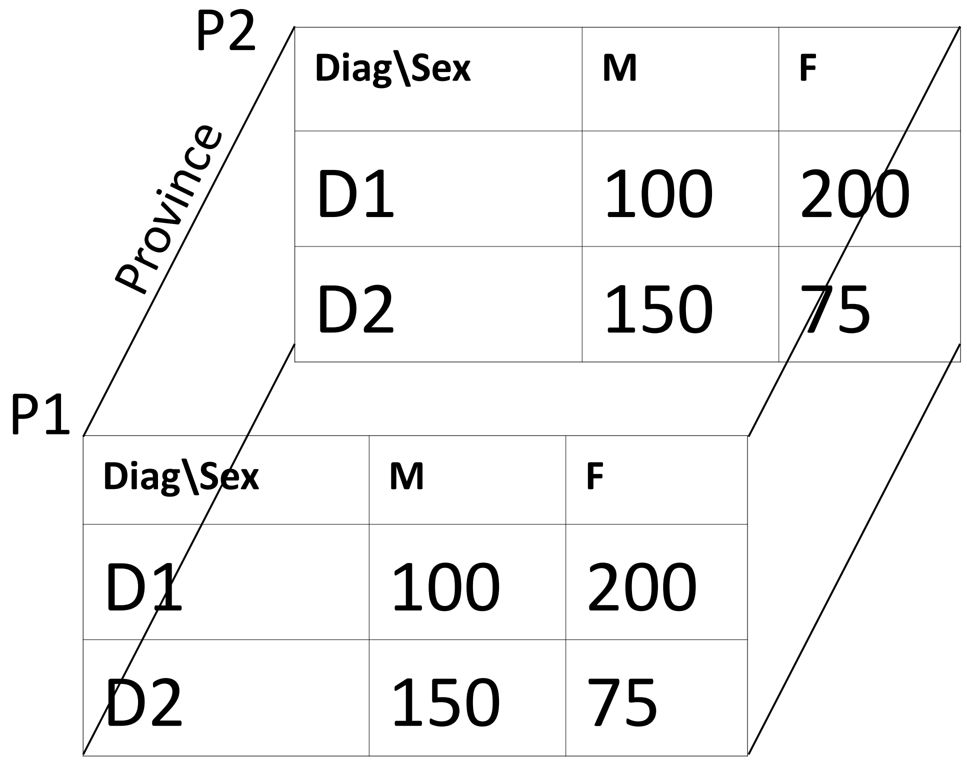
Diag\Sex	**M**	**F**
D1	100	200
D2	150	75

OLAP Cube

Fact table

Diagnostic	Sex	Province	Total
D1	M	P1	100
D1	F	P1	200
D1	M	P1	100
D1	F	P1	200
D2	M	P2	150
D2	F	P2	75
D2	M	P2	150
D2	F	P2	75

3D view



P2

Diag\Sex	M	F
D1	100	200
D2	150	75

P1

Diag\Sex	M	F
D1	100	200
D2	150	75

OLAP tools

- The interesting thing is NOT ONLY to be able to query, in a way, something you can do with selections, projections, concatenation and traditional groupings.
- What is really interesting OLAP tools are its refinement operators for handling queries.
  - DRILL
  - ROLL
  - SLICE & DICE
  - PIVOT
  - ROLLUP
  - CUBE

ROLL-DRILL

Diagnostic	Sex	Province	Total
D1	M	P1	100
D1	F	P1	200
D1	M	P1	100
D1	F	P1	200
D2	M	P2	150
D2	F	P2	75
D2	M	P2	150
D2	F	P2	75

Diagnostic	Sex	Total
D1	M	200
D1	F	400
D2	M	300
D2	F	150

$\xrightarrow{\text{roll}}$

$\xleftarrow{\text{drill}}$

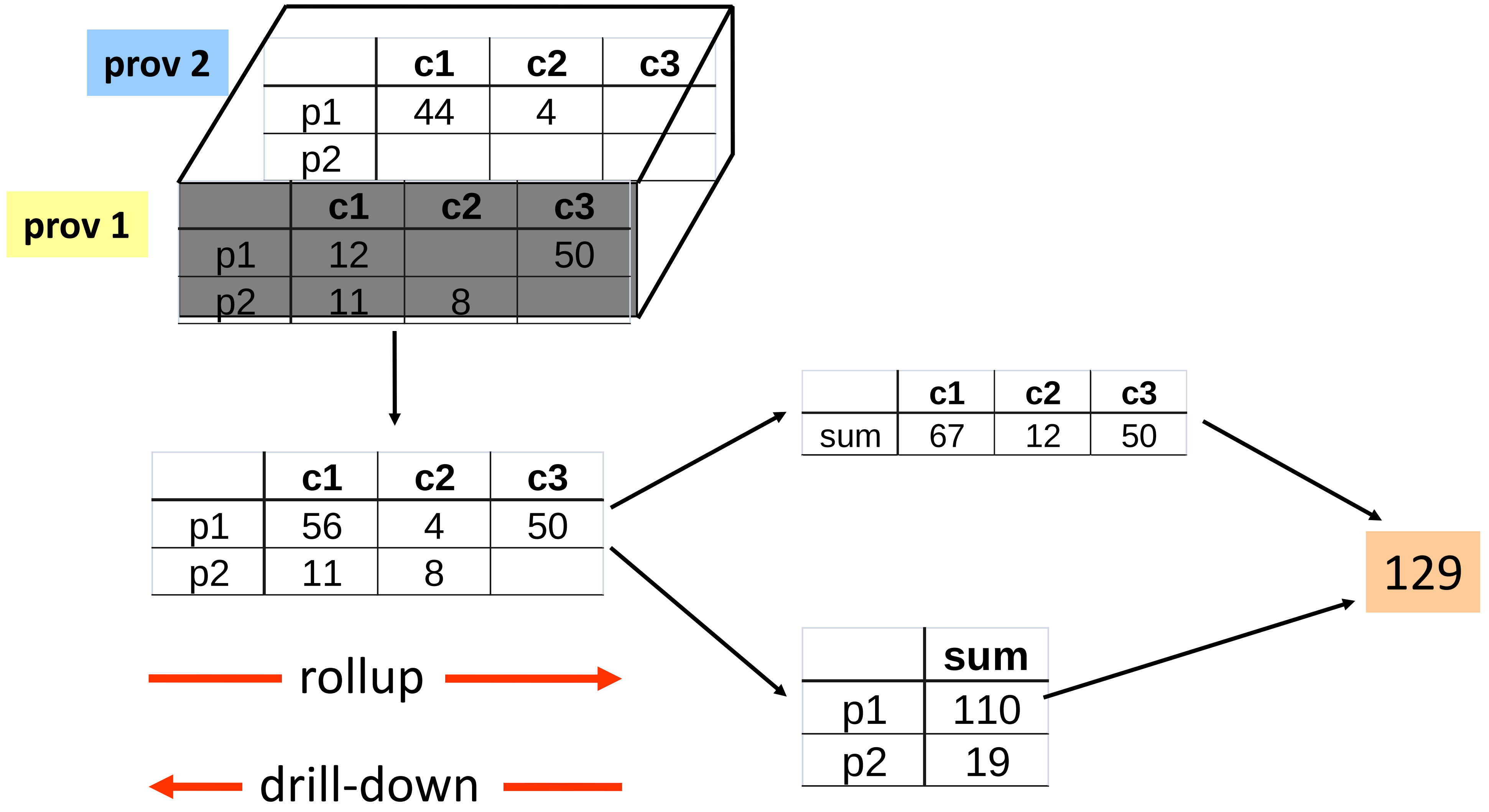
ROLL-DRILL

- Aggregate (consolidate) and disintegrate (division):
  - aggregation (**roll**): delete a grouping criterion in the analysis, aggregating the current groups.
  - disintegrate (**drill**): enter a new grouping criterion in the analysis, breaking existing groups.
- Aggregation in SQL: sum, count, max, min, average, …

DRILLing (ROLLing)

- DRILL (ROLL) can be done on:
  - attributes of one dimension on which a hierarchy has been defined:
    - DRILL-DOWN: upper to lower aggregation level
    - ROLL-UP: lower to upper aggregation level.
      - departament – category - product (Product)
      - year – semester – month – day (Time)
- Other "drill"
  - DRILL-ACROSS: join several fact tables:
    - Careful implementation.
    - Sometimes this name is also used to change the dimension
  - DRILL-THROUGH: Use SQL to explore up to the relational back-end tables

OLAP. ROLL-DRILL



prov 2

	c1	c2	c3
p1	44	4	
p2			

prov 1

	c1	c2	c3
p1	12		50
p2	11	8	

	c1	c2	c3
p1	56	4	50
p2	11	8	

	c1	c2	c3
sum	67	12	50

	sum
p1	110
p2	19

129

OLAP operators

- SLICE & DICE: select and project
  - SLICE: Delete a dimension on the analysis
  - DICE: Define a condition on some attribute of the dimension

- PIVOT: Rotate, reorientate the 2D- table view.

SLICE & DICE

Diagnóstico	Sexo	Provincia	Total	Núm
D1	H	P1	100	6
D1	M	P1	200	5
D1	H	P2	100	6
D1	M	P2	200	11
D2	H	P1	150	7
D2	M	P1	75	7
D2	H	P2	150	2
D2	M	P2	75	1

$\xrightarrow{\text{Slice (Num) \& dice (P1)}}$

Diagnostic	Sex	Total
D1	M	100
D1	F	200
D2	M	150
D2	F	70

PIVOT

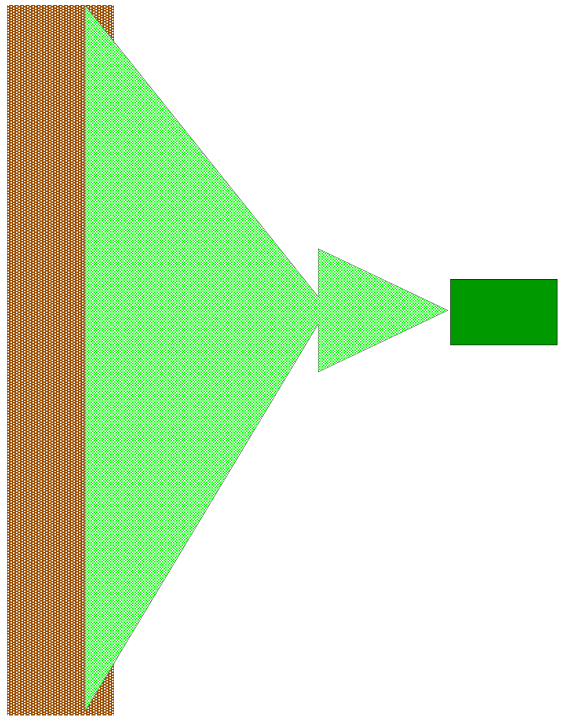
	Diagnóstico	Sexo	Total
P1	D1	H	100
	D1	M	200
	D2	H	150
	D2	M	75
P2	D1	H	100
	D1	M	200
	D2	H	150
	D2	M	75

$\xrightarrow{\text{Pivot}}$

	Diagnóstico	Provincia	Total
H	D1	P1	100
	D1	P2	100
	D2	P1	150
	D2	P2	150
M	D1	P1	200
	D1	P2	200
	D2	P1	75
	D2	P2	75

OLAP extensions to SQL

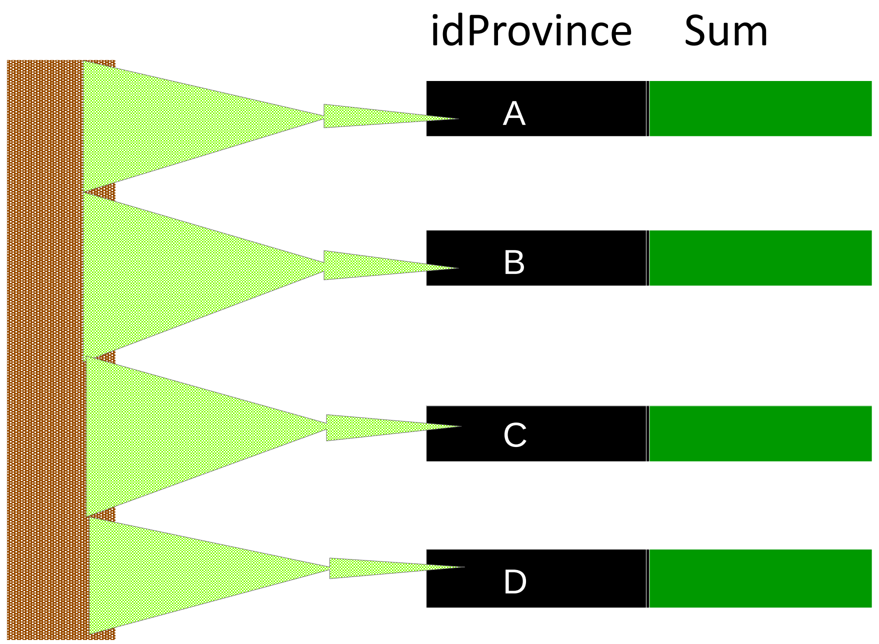
- SQL aggregation
  - sum(), count(), avg(), min(), max()
- Basic idea:
  - Combine values in one column
  - Into only one value
- Syntax:
  - SELECT sum(cost) FROM diagnostic;
- DISTINCT
  - Allows the aggregation only of different values
  - SELECT COUNT(DISTINCT cost) FROM diagnostic



OLAP extensions to SQL

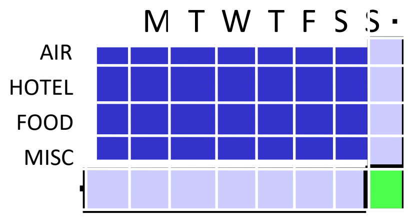
- GROUP BY + HAVING
- Aggregating in subgroups of the table
- That fulfill some condition
- Syntax

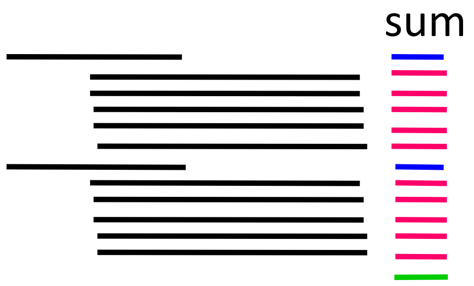
```
SELECT idProvinc, sum(cost)
FROM  diagnostic
GROUP BY idProvinc
HAVING population > 2000;
```



OLAP extensions to SQL

- Limitations
  - Useful aggregations are difficult to calculate
    - Data cube
    - Complex: median, variance
    - Moving average
    - Rankings
  - Marginals or crosstabs
    - GROUP BY limited to 0-D and 1-D
  - Include sum and partial sums
    - drill-down & roll-up





ROLLUP

- ROLLUP: performs the aggregation for the set of prefix of the attributes given
- Example:

```
SELECT item-name, color, size, SUM(number)
  FROM sales
GROUP BY ROLLUP(item-name, color, size)
```

- Calculates SUM for the n+1 prefixes:
- { (item-name, color, size), (item-name, color), (item-name), ( ) }
- Very useful for aggregating in hierarchies defined on dimensions
- It can be done in SQL without OLAP extensions, but very inefficiently.
- To improve efficiency: calculate the higher level aggregations using partial results of the more detailed levels

CUBE

- CUBE: generalization of GROUP BY to n-dimensions.
- Calculates the aggregation function for all the subsets of the attributes given instead for only the prefixes (ROLLUP)
- Example:

```
SELECT item-name, color, size, SUM(number)
 FROM sales
 GROUP BY CUBE (item-name, color, size)
```

- Calculates the aggregate for the set of 2^n combinations:
- {(item-name, color, size),
  (item-name, color), (item-name, size), (color, size),
  (item-name), (color), (size),
  () }
- For each combination, the result is null for attributes that are not present in the combination.

GROUPING

- SQL:1999 uses NULL for representing both ALL and “usual” null
- In order to distinguish them we can use the GROUPING function that applied to an attribute
  - Returns 1 if NULL represents ALL
  - Returns 0 otherwise
  - Combined with DECODE we can return the desired value

WINDOW

- WINDOW clause defines **ordered** and **overlapping** groups of rows to calculate aggregates included at the end of each row.
- GROUP BY clause defines disjoint partitions of tuples in a sorted table, then calculates aggregates on those partitions, and generates a tuple with the result of the aggregate for each partition
- Example: "For each day, we want the average cost of obtaining diagnoses from the previous day, the current and the next, and cumulatively in the last 7 days":

SELECT date,

sum(cost) **OVER** (order by date **BETWEEN ROWS** 1 **preceding** and 1 **following**),

sum(sum(precio) **OVER** (order by date **ROWS** 7 **preceding**))

FROM diagnostics;

WINDOW

- Syntax:
- SELECT attribute_list_1,
  - Aggregated_function **OVER** W as windowName
- FROM table_list
- WHERE constraints
- **WINDOW** W AS (
  - **PARTITION BY** attribute_list_2
  - **ORDER BY** attribute_list_3
  - **frame declaration**)

WINDOW

- Execution:
- FROM, WHERE, GROUP and HAVING generate an intermediate table.
- PARTITION: each partition contains tuples with the same values in the attributes given in attribute_list_2
- ORDER BY: rows in each partition are sorted according to the values of the attributes in attribute_list_3
- SELECT the tuples under the constraints established in the frame declaration
  - RANGE: logical conditions (ie: 5 days)
  - ROWS: in rows (ie: 5 preceding rows)

WINDOW

- Frame examples:
  - between rows unbounded preceding and current row
  - rows unbounded preceding
  - range between 10 preceding and current row
  - range interval 10 day preceding
  - range between interval 1 month preceding and interval 1 month following
- Default frame: If the frame is not specified, all preceding and current rows are considered in the partition
  - RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

RANK

- RANK assigns to every tuple a rank based in some sorting of some attribute
- Example: given a cost-province relation rank each province by its cost.

```
SELECT province,
    rank() over (order by coste desc) as provrank
FROM diagnostic
```

- Afterwards, the result can be sorted by that field

```
SELECT province,
   rank() over (order by coste desc) as provrank
FROM diagnostic order by provrank
```

- RANKING allow gaps if there are 2 values with the same ranking.
  - Example: if the 1rst and 2nd classified have the same cost, then both will be assigned rank 1, and the next row will have rank 3
  - DENSE_RANK does not allow gaps, so the next row will have rank 2

RANK

- RANK over partitions:
  - “Rank the community and provinces by their cost”

```
SELECT province, comunity,
rank () over (partition by comunity order by cost desc) as
prov-comunity-rank
FROM diagnostic
ORDER BY by comunity, prov-comunity-rank
```

- Several RANK can be included in the same query.

Other functions

- Other rank functions
  - **percent_rank**: it displays each row as a percentage of all the other rows up to 100% in a rank
  - **cume_dist**: cummulative distribution
    - It displays the number of values in the set preceding and including x in the specified order divided by the number of rows.
  - **row_number**
  - **ntile**(*x*): cuantile
    - Divides the rows in the partition in x buckets with the same number ot rows

```
SELECT comunity, count (*),
 ntile(3) over (order by count(*) desc) as quartile
FROM diagnostic join patient
GROUP BY comunity;
```

Other functions

- Numeric functions (exp, cos, ln, …)
- Aggregated (std, var, corr, regr, …)
- Window functions:
    - Ranking: rank, dense_rank
    - Distribution: percentage_rank, cume_dist
    - Count: row_num
- Frame functions: lag, lead, …
- SQL:1999 allows the use of **nulls first** and **nulls last**

```
SELECT student-id,
rank ( ) over (order by marks desc nulls last) as s-rank
FROM student-marks
```

Codd rules

- 1 Multidimensional view of data
- 2 Transparency to support (ROLAP, MOLAP)
- 6 Generic operations regarding the number of dimenstions
- 9 Flexibility in the definition of the dimensions: constraints, aggregations and hierarchies among them.
- 10 Intuitive handling of operators: drill, roll, slice-&-dice, pivot.
- 12 No limit dimensions
- other:
  - 3 Accessibility from different data sources
  - 4 Coherent performance in reporting
  - 5 Client-Server Architecture
  - 7 Dynamic sparse matrix
  - 8 Multiuser support
  - 11 flexible report generation