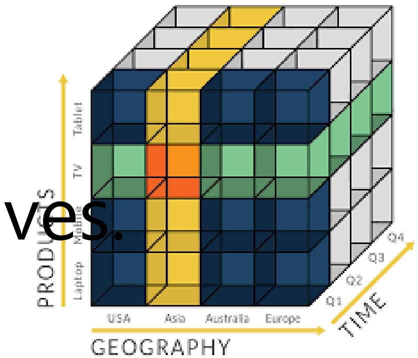
Business intelligence

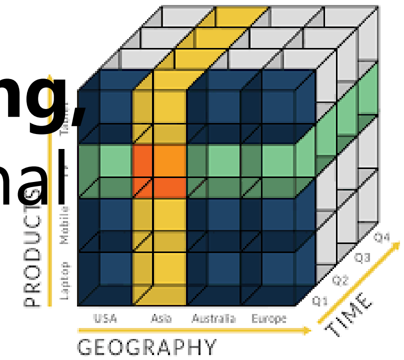
Unit 3 – Data exploitation. Query languages and visualization
S3-1 – OLAP

OLAP tools



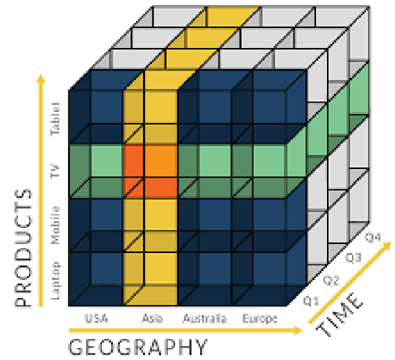
OLAP tools are used in BI for **analyzing multidimensional data** from multiple perspectives.
OLAP tools provide the user with a **multidimensional view** of data (multidimensional schema) for each activity that is being analyzed.
The user formulates queries to the OLAP tool selecting multidimensional 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 (e.g. by means a SQL SELECT statement).

OLAP tools



Multidimensional Analysis: OLAP enables **slicing, dicing, and drilling down** into data for additional insights.
Complex Calculations: OLAP tools support advanced calculations and metrics to evaluate business performance.
Fast Query Performance: MOLAP provides pre-aggregation, while HOLAP offers hybrid performance improvements.
Data Consistency: Ensures accurate, consistent analysis across BI applications.
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

Geographic — WHERE?

- IdCenter
- Cluster
- Clinic
- Area
- City
- Province
- State
- Country

Time — WHEN?

- idTime
- Hour
- Day
- Week
- Month
- Semester
- Year
- Decade

idPlace [FK]
idTime [FK]
idPatient [FK]
idDiagnostic[FK]
Probability
Cost

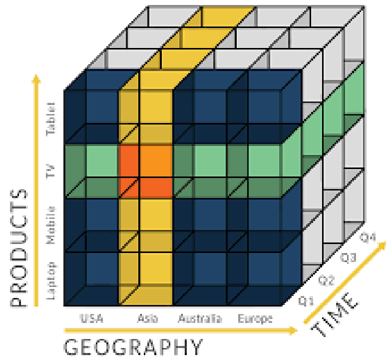
WHO? — **Patient**

- 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

↔

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

Province

P2

Diag\Sex	M	F
D1	100	200
D2	150	75

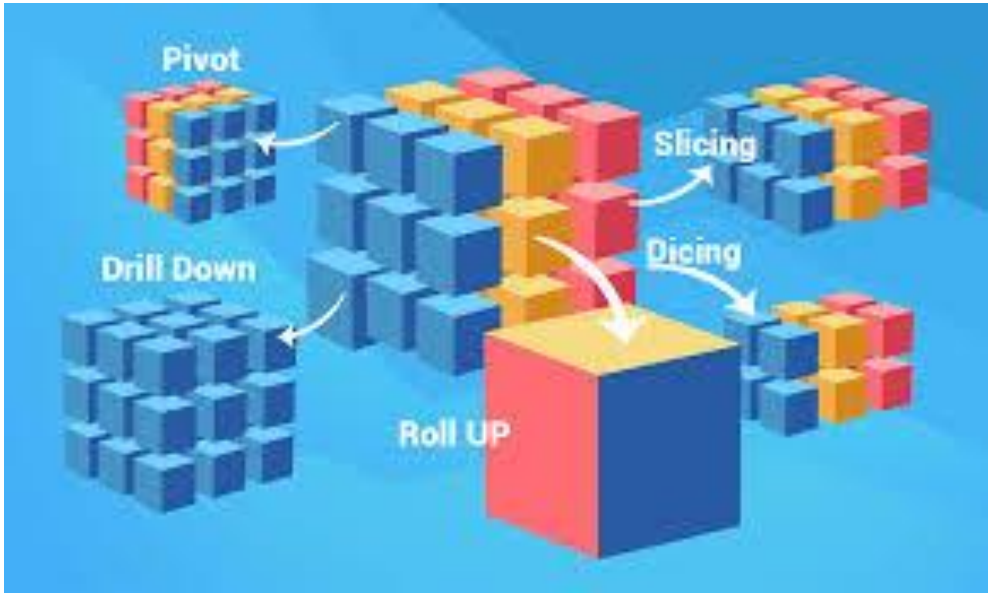
P1

Diag\Sex	M	F
D1	100	200
D2	150	75

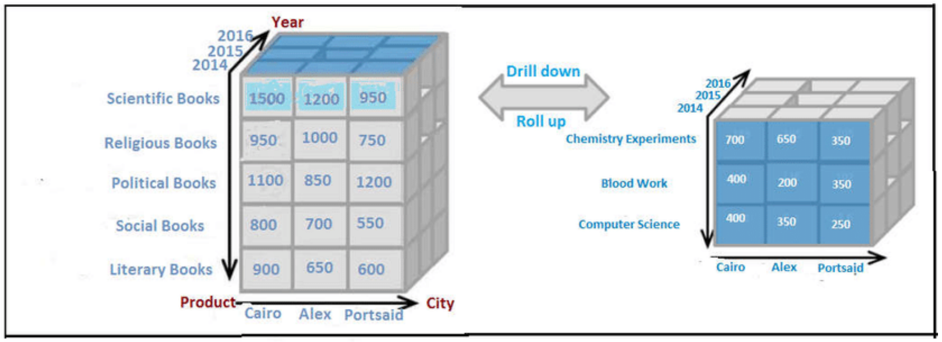
OLAP tools

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

roll →

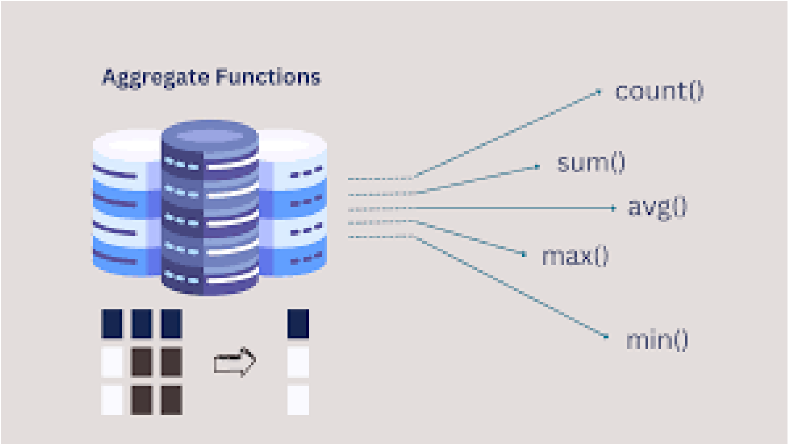
← **drill**

ROLL-DRILL

Aggregate (consolidate) and **disintegrate** (division):

- aggregation (***roll***): delete a grouping criterion in the analysis, aggregating the current groups. The granularity of one or more dimensions is aggregated.
- 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
    - departament – category - product (Product)
    - year – semester – month – day (Time)
  - ROLL-UP: lower to upper aggregation level.

Other “drill”:

- **DRILL-ACROSS**: join several fact tables.
- **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

rollup →

← **drill-down**

OLAP operators

SLICE & DICE: select and project

- **SLICE**: Filter a specific dimension by selecting a single value.
- **DICE**: Filter data by applying multiple conditions across dimensions.

PIVOT: Rotate, reorientate the data view to analyze different perspectives.

Store	Product	Sales
A	TV	2
A	TV	4
B	TV	6
B	DVD	8

→

Store	Avg(Sales) for TV	Avg(Sales) for DVD
A	3	(Empty)
B	6	8

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

Slice by removing Prov dimension

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

```
SELECT diagnostic, sex,
SUM(total)
FROM table
WHERE province = 'P1'
GROUP BY diagnostic, sex;
```

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

Pivot →

	Diagnóstico	Prov incia	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

PIVOT

Example

```
CREATE EXTENSION IF NOT EXISTS tablefunc;
CREATE TABLE to_pivot (
ID serial,
Name TEXT, -- Name student
Course TEXT, -- Course
Grade INT,
primary KEY(ID)
);
INSERT INTO to_pivot(Name,Course,Grade) VALUES
('Pepe', 'BDII', 9),
('Jose', 'BDII', 7),
('Pepe', 'BI', 8),
('Jose', 'BI', 5);
```

	id	name	course	grade
1	1	Pepe	BDII	9
2	2	Jose	BDII	7
3	3	Pepe	BI	8
4	4	Jose	BI	5

	name	BI	BDII
1	Jose	7	5
2	Pepe	9	8

```
SELECT * FROM to_pivot;
SELECT * FROM crosstab ('Select Name,Course,Grade from to_pivot order by 1,2') as Pivoted (Name text, "BI" INT ,"BDII" INT);
```

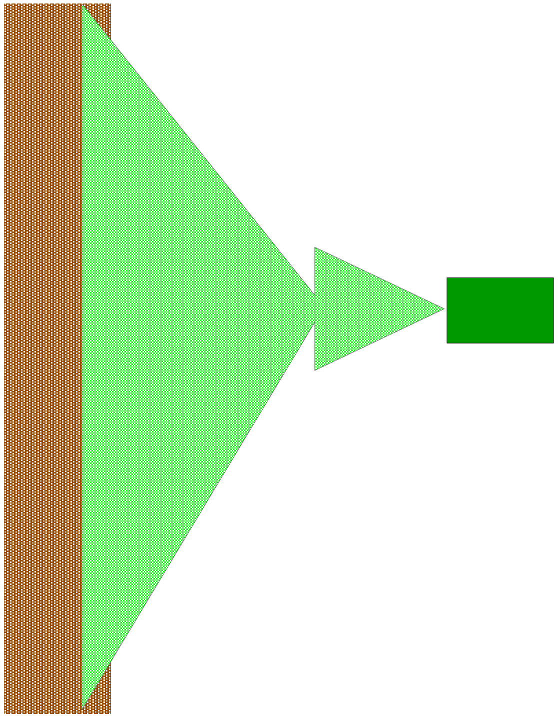
•

SQL aggregation

- sum(), count(), avg(), min(), max()

Basic idea:

- Combine values in one column
- Into only one value



Syntax:

```sql
SELECT sum(cost) FROM diagnostic;
```

DISTINCT

- Allows the aggregation only of different values

```sql
SELECT COUNT(DISTINCT cost) FROM diagnostic;
```

OLAP extensions to SQL

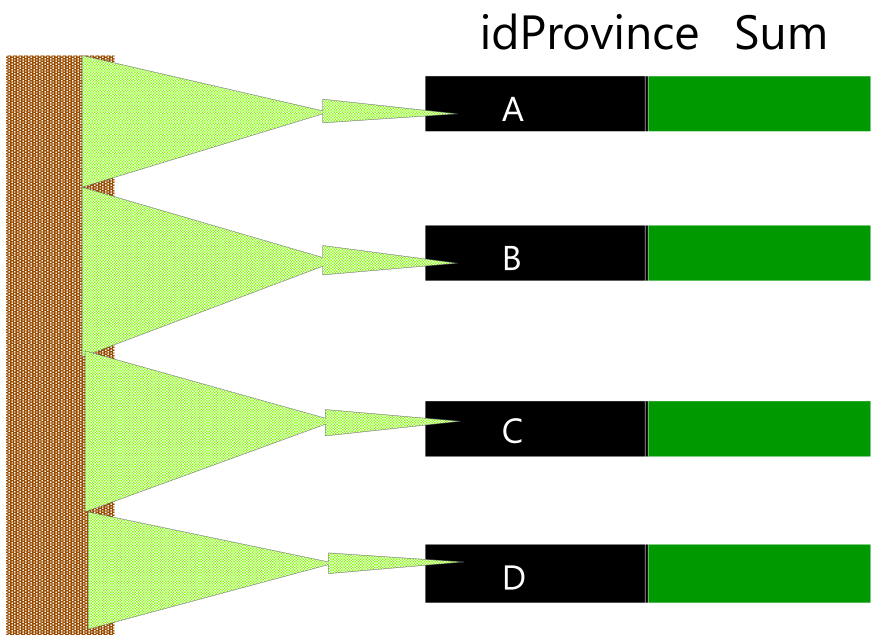
GROUP BY: Groups rows that have same values in specified columns. Aggregation functions are usually associated to it: `select idprov,sum(b) from table group by idprov`

GROUP BY + HAVING

Aggregating in subgroups of the table that fulfill some condition applied after the grouping.

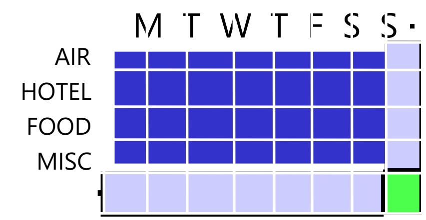
Syntax

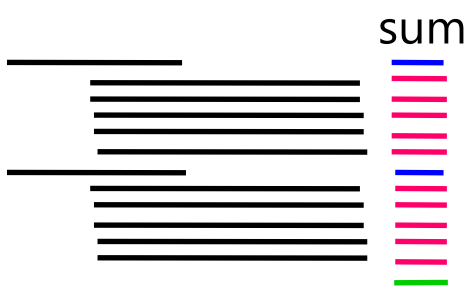
```
SELECT idProvinc, sum(cost)
FROM diagnostic
GROUP BY idProvinc
HAVING population > 2000;
```



Limitations without them

- Useful aggregations are difficult to calculate
  - Data cube
  - Complex: median, variance
  - Moving average
  - Rankings
- Marginals or crosstabs
  - SQL requires additional functionality
- 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 (multiple GROUP BY and UNION operations).

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 aggregated rows (ALL) and “usual” null (missing values).

When we have an OLAP query, how to know?

- 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 (or CASE) we can return the desired value

```
SELECT DECODE(GROUPING(Year), 1, 'Total', Year) AS Year, DECODE(GROUPING(Region), 1, 'Total', Region) AS Region, SUM(SalesAmount) AS TotalSales FROM Sales GROUP BY CUBE (Year, Region);
```

GROUPING SETS

GROUPING SETS allows us to specify multiple groupings in a single query.

We can define the subsets of columns for grouping.

No need to have separate queries or UNION ALL.

Provides better efficiency.

```
select Name, Course, AVG(Grade) from
to_pivot group by
grouping sets
((Name,Course),(Course),());
```

	name	course	avg
1	[NULL]	[NULL]	7.25
2	Pepe	BI	8
3	Jose	BI	5
4	Jose	BDII	7
5	Pepe	BDII	9
6	[NULL]	BDII	8
7	[NULL]	BI	6.5

WINDOW FUNCTIONS

WINDOW clause defines **ordered** and **overlapping** groups of rows to calculate aggregates based on a defined “window”, while retaining the original rows.

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. It eliminates rows-level granularity

- 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 ROWS BETWEEN 1
preceding and 1 following), sum(cost) OVER (order by date ROWS
BETWEEN 7 preceding and CURRENT ROW)) FROM diagnostics;
```

WINDOW FUNCTIONS

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**)

Frame declaration is opcional. By default, RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

WINDOW FUNCTIONS

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)

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
```

RANK 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 comunity, prov-comunity-rank
```

si particionamos por provinicia, daria que todas las provincias tendrian ranking 1

- Several RANK can be included in the same query.

Other rank functions

- **percent_rank**: it displays each row as a percentage of all the other rows up to 100% in a rank. = (rank-1)/(Number_rows_partition-1).

```
SELECT Student, Score,
PERCENT_RANK() OVER
(ORDER BY Score DESC) AS
Percent_Rank
FROM Students;
```

	student	score	percent_rank
1	Elena	92	0
2	Jose	90	0.25
3	Pepe	85	0.5
4	Marco	85	0.5
5	Manolo	78	1

- **cume_dist**: cummulative distribution: It displays the number of values in the set preceding and including in the specified order divided by the number of rows.

```
SELECT Student, Score,
CUME_DIST() OVER (ORDER BY Score
DESC) AS Cume_Dist
FROM Students;
```

	student	score	cume_dist
1	Elena	92	0.2
2	Jose	90	0.4
3	Pepe	85	0.8
4	Marco	85	0.8
5	Manolo	78	1

Other functions

Other rank functions

- **row_number:**

```
SELECT Student, Score,
ROW_NUMBER() OVER (ORDER BY Score DESC) AS Row_Number
FROM Students;
```

	student	score	row_number
1	Elena	92	1
2	Jose	90	2
3	Pepe	85	3
4	Marco	85	4
5	Manolo	78	5

- **ntile**(*x*): cuantile
  - Divides the rows in the partition in x buckets with the same number of rows

```
SELECT Student, Score,
NTILE(2) OVER (ORDER BY Score
DESC) AS ntile
FROM Students;
```

	student	score	ntile
1	Elena	92	1
2	Jose	90	1
3	Pepe	85	1
4	Marco	85	2
5	Manolo	78	2

Other functions

- Numeric functions (exp, cos, ln, …) `SELECT EXP(2);`
- Aggregated (std, var, corr, regr, …) `SELECT STDDEV(Score) FROM Students;`
- Frame functions: lag, lead, …

	student	score	previous_score	next_score
1	Elena	92	[NULL]	90
2	Jose	90	92	85
3	Pepe	85	90	85
4	Marco	85	85	78
5	Manolo	78	85	[NULL]

```
SELECT Student,Score,
LAG(Score) OVER (ORDER BY Score DESC) AS Previous_Score,
LEAD(Score) OVER (ORDER BY Score DESC) AS Next_Score
FROM Students;
```

SQL:1999 allows the use of **nulls first** and **nulls last.** It serves to define if nulls appear before or after non-null values in the sort ordering. By default, NULLS FIRST for DESC order and NULLS LAST for ASC order.

```
SELECT Student,Score,ROW_NUMBER() OVER (ORDER
BY Score ASC) AS Default_Order,ROW_NUMBER()
OVER (ORDER BY Score ASC NULLS FIRST) AS
Nulls_First_Order,ROW_NUMBER() OVER (ORDER BY
Score ASC NULLS LAST) AS Nulls_Last_Order
FROM Students;
```

	student	score	default_order	nulls_first_order	nulls_last_order
1	Manolo	78	1	2	1
2	Pepe	85	2	3	2
3	Marco	85	3	4	3
4	Jose	90	4	5	4
5	Elena	92	5	6	5
6	Pat	[NULL]	6	1	6

Codd rules

1 Multidimensional view of data
2 Transparency to support (ROLAP, MOLAP) interfaz simple para el usuario, le da igualk lo que hay detras
3 Accessibility el usuario debe acceder de forma simple a los datos, da igual donde esten
4 Coherent performance in reporting resultados rapidos y consistentes
5 Client-Server Architecture
6 Generic operations regarding the number of dimensions
7 Dynamic sparse matrix
8 Multiuser support
9 Flexibility in the definition of the dimensions: constraints, aggregations and hierarchies among them.
10 Intuitive handling of operators: drill, roll, slice-&-dice, pivot.
11 flexible report generation
12 No limit dimensions