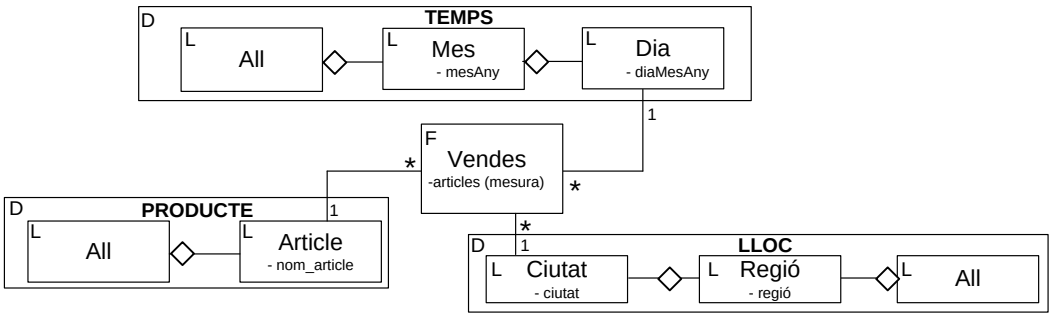
INTRODUCTION

Computing aggregate data is a basic feature for OLAP (as well as for other tools). For this reason, the standard SQL-99 added 3 new modifiers to the GROUP BY clause in order to support and enhance advanced aggregate computation. In this document we discuss these modifiers as well as their usefulness to implement star-join (i.e., ROLAP) implementations. In short, and as you will realize after reading this document, these modifiers tune up the GROUP BY clause syntax (without changing its behaviour).

The new GROUP BY modifiers are the following: GROUPING SETS, ROLLUP and CUBE. Although their names may ring a bell to multidimensional concepts already introduced in this course, do not get mislead by their names, which are not really appropriate according to the multidimensional theory.

Let us first introduce a traditional case study, upon which we will introduce these new keywords. Consider the following star-schema:



It is a simple star-schema, with 3 dimensions (producte –product-, temps –time- and lloc –place-). Suppose now a star-join logical implementation and assume we aim at showing data (i.e., a data cube) at article, mes and region granularity. As presented earlier in this course, the cube-query pattern is intended to retrieve data from a ROLAP star-schema implementation. To do so, the cube-query must retrieve the Vendes atomic data and perform two roll-ups (one over the time dimension from dia to mes and the other one over the lloc dimension from ciutat to regió). The result would be the following:

```
SELECT d1.nom_article, d2.regió, d3.mesAny, SUM(f.articles)

FROM Vendes f, Producte d1, Lloc d2, Temps d3

WHERE f.IDProduct=d1.ID AND f.IDPlace=d2.ID AND f.IDTime=d3.ID

AND d1.nom_article IN ('Bolígrafs','Gomes') AND d2.regió='Catalunya'

AND d3.mesAny IN ('Gener2002','Febrer2002')

GROUP BY d1.nom_article, d2.regió, d3.mesAny

ORDER BY d1.nom_article, d2.regió, d3.mesAny
```

For the sake of comprehension, the reader will note three slicers over each dimension. The reason is to restrict the amount of values to be shown in the upcoming figures.

GROUPING SETS

Typically, the users do not want to just visualize data at a specific aggregation level (i.e., at a certain granularity), but *totals* are of their interest. These aggregations are depicted in a kind of table known as *cross-tab*. For example:

Vendes	Catalunya		
	Gener02	Febrer02	Total
Bolígrafs	275827	290918	566745
Gomes	784172	918012	1702184
Total	1059999	1208930	2268929

This table is not a proper cube because it mixes data cells from different granularities or cubes (each granularity represented by a different colour). Note, however, that this table can be seen as the union of four different cubes (i.e., four different granularity levels). For example, the yellow cells represent the data cube obtained at the {article, region and mes} granularity. In other words, they represent the Vendes fact at its atomic granularity. The other cells are produced by aggregating data to the All level of some dimensions. Specifically, red cells represent the cube at the {mes, regió, All} granularity level (i.e., a roll-up over the article dimension has been performed to the All level), the green ones at {All, regió, article} granularity and the blue ones at (All, regió, All).

Indeed, in the same way we can define a function piecewise, we can do it with a cross-tab by uniting cube pieces. For example:

$$\text{Vendes} \oplus \textit{Roll-up}_{\text{Temps::All}}(\text{Vendes}) \oplus \textit{Roll-up}_{\text{Articles::All}}(\text{Vendes}) \oplus \textit{Roll-up}_{\text{Articles::All,Temps::All}}(\text{Vendes})$$

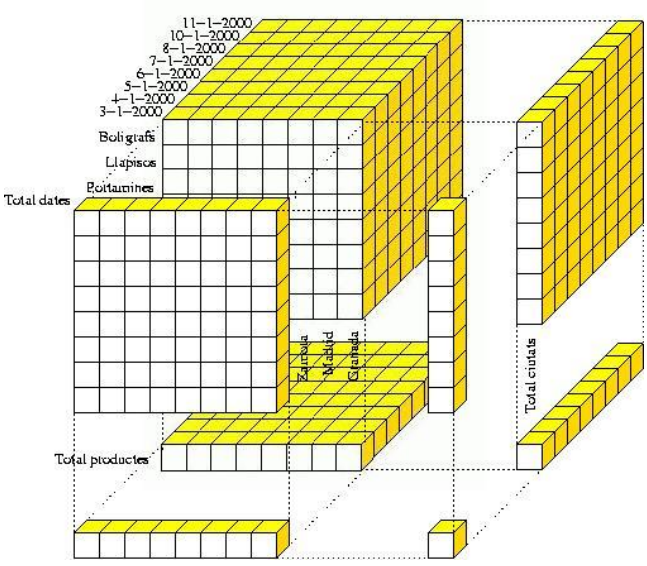
Where $\oplus$ denotes a cube composition tablewise. Now go back to the previous cube-query introduced and realize that that query only retrieves the four yellow cells. To obtain the other cells we need to union four queries as shown below:

```
SELECT d1.nom_article, d2.regió, d3.mesAny, SUM(fet.articles)
FROM Vendes fet, Producte d1, Lloc d2, Temps d3
WHERE fet.IDProducte=d1.ID AND fet.IDLloc=d2.ID AND fet.IDTemps=d3.ID
   AND d1.nom_article IN ('Bolígrafs','Gomes') AND d2.regió='Catalunya'
   AND d3.mesAny IN ('Gener2002','Febrer2002')
GROUP BY d1.nom_article, d2.regió, d3.mesAny
UNION
SELECT d1.nom_article, d2.regió, 'Total', SUM(fet.articles)
FROM Vendes fet, Producte d1, Lloc d2, Temps d3
WHERE fet.IDProducte=d1.ID AND fet.IDLloc=d2.ID AND fet.IDTemps=d3.ID
   AND d1.nom_article IN ('Bolígrafs','Gomes') AND d2.regió='Catalunya'
   AND d3.mesAny IN ('Gener2002','Febrer2002')
GROUP BY d1.nom_article, d2.regió
UNION
SELECT 'Total', d2.regió, d3.mesAny, SUM(fet.articles)
FROM Vendes fet, Producte d1, Lloc d2, Temps d3
```

```
WHERE fet.IDProducte=d1.ID AND fet.IDLloc=d2.ID AND fet.IDTemps=d3.ID
   AND d1.nom_article IN ('Bolígrafs','Gomes') AND d2.regió='Catalunya'
   AND d3.mesAny IN ('Gener2002','Febrer2002')
GROUP BY d2.regió, d3.mesAny
UNION
SELECT 'Total', d2.regió, 'Total', SUM(fet.articles)
FROM Vendes fet, Producte d1, Lloc d2, Temps d3
WHERE fet.IDProducte=d1.ID AND fet.IDLloc=d2.ID AND fet.IDTemps=d3.ID
   AND d1.nom_article IN ('Bolígrafs','Gomes') AND d2.regió='Catalunya'
   AND d3.mesAny IN ('Gener2002','Febrer2002')
GROUP BY d2.regió
ORDER BY d1.nom_article, d2.regió, d3.mesAny;
```

Check carefully each united piece. They are almost identical. Indeed, formally, we are only changing the GROUP BY clause to obtain the desired granularity. Furthermore, if we were interested in computing the totals for all three dimensions instead of two, we would need 7 unions; i.e., 8 different SQL queries at different granularities only changing their GROUP BY clause. In other words, in addition to those granularities computed in the example introduced we should compute the {mes, All, article}, {mes, All, All} and {All, All, article} granularities.

Next figure sketches the eight cubes to union (vendes per dia, producte and ciutat; per dia and producte; per dia and ciutat; per producte and ciutat; per dia; per producte; per ciutat; and the overall total).



It is easy to realize that the amount of unions to perform grows at an exponential rate with regard to the number of dimensions. Fortunately, the SQL'99 standard provides specific syntax to save us time. As shown in the next query, we can use the GROUPING SETS keyword in the GROUP BY clause to produce the desired granularities (in brackets after the keyword). In this way the other query clauses are written only once:

```
SELECT d1.nom_article, d2.regió, d3.mesAny, SUM(fet.articles)
```

```
FROM Vendes fet, Producte d1, Lloc d2, Temps d3
WHERE fet.IDProdcute=d1.ID AND fet.IDLloc=d2.ID AND fet.IDTemps=d3.ID
   AND d1.nom_article IN ('Bolígrafs', 'Gomes') AND d2.regió='Catalunya'
   AND d3.mesAny IN ('Gener2002','Febrer2002)
GROUP BY GROUPING SETS ((d1.nom_article, d2.regió, d3.mesAny),
                        (d1.nom_article, d2.regió),
                        (d2.regió, d3.mesAny),
                        (d2.regió))
ORDER BY d1.nom_article, d2.regió, d3.mesAny;
```

The query output would be the following:

nom_article	Regió	mesAny	Articles
Bolígrafs	Catalunya	Gener02	275827
Bolígrafs	Catalunya	Febrer02	290918
Bolígrafs	Catalunya	NULL	566745
Gomes	Catalunya	Gener02	784172
Gomes	Catalunya	Febrer02	918012
Gomes	Catalunya	NULL	1702184
NULL	Catalunya	Gener02	1059999
NULL	Catalunya	Febrer02	1208930
NULL	Catalunya	NULL	2268929

Besides the visualization problem (which the corresponding exploitation tool is responsible for), pay attention to the NULL values. What is their meaning in this context? Is it 'unknown', maybe 'not applicable'? Well, none of them. Indeed, it is a new meaning introduced in the SQL'99 standard which means 'total' (i.e., it represents the All value for that dimension).

But as you might have thought, how can we distinguish the meaning between different NULL values? In other words, given a NULL value, how can we know if this is the result of aggregating data or an unknown value? To answer this question, the standard introduces the GROUPING function. This function takes as parameter an attribute and it returns '1' if it represents a total (i.e., produced by an aggregation). Otherwise, it returns '0'. Thus, you might need to use this function for presentation issues and, for example, rewrite the previous query to replace the NULL values by the total word (note that d2.regió appears in all four aggregations and thus, it never holds for the total meaning):

```
SELECT
   CASE WHEN GROUPING(d1.nom_article)=1 THEN 'TotalDeGenerIFebrer'
        ELSE d1.nom_article,
   d2.regió,
   CASE WHEN GROUPING(d3.mesAny)=1 THEN 'TotalDeBoligrafsIGomes'
        ELSE d3.mesAny,
   SUM(fet.articles)
FROM Vendes fet, Producte d1, Lloc d2, Temps d3
WHERE fet.IDProdcute=d1.ID AND fet.IDLloc=d2.ID AND fet.IDTemps=d3.ID
```

```
   AND d1.nom_article IN ('Bolígrafs', 'Gomes') AND d2.regió='Catalunya'
   AND d3.mesAny IN ('Gener2002','Febrer2002)
GROUP BY GROUPING SETS ((d1.nom_article, d2.regió, d3.mesAny),
                        (d1.nom_article, d2.regió),
                        (d2.regió, d3.mesAny),
                        (d2.regió))
ORDER BY d1.nom_article, d2.regió, d3.mesAny;
```

In this case, the query output would be the following:

nom_article	Regió	mesAny	articles
Bolígrafs	Catalunya	Gener02	275827
Bolígrafs	Catalunya	Febrer02	290918
Bolígrafs	Catalunya	TotalDeGenerIFebrer	566745
Gomes	Catalunya	Gener02	784172
Gomes	Catalunya	Febrer02	918012
Gomes	Catalunya	TotalDeGenerIFebrer	1702184
TotalDeBoligrafsIGomes	Catalunya	Gener02	1059999
TotalDeBoligrafsIGomes	Catalunya	Febrer02	1208930
TotalDeBoligrafsIGomes	Catalunya	TotalDeGenerIFebrer	2268929

5.2.1. ROLLUP

As previously discussed, the number of totals grows at an exponential rate regarding the number of dimensions in the cube. As a consequence, even if we can save writing the other query clauses by using the GROUPING SETS modifier, it can still happen to be unbearable in some cases. In other words, the GROUPING SETS avoids writing the same query *n* times, but we still have to write all the attribute combinations needed to produce the desired granularities.

To facilitate even more computing aggregations, the SQL'99 standard introduced the ROLLUP keyword. Given an attribute set, it computes all the aggregations by disregarding, on each grouping, the right-most attribute in the set. Thus, it is not a set in the mathematical sense, as order does matter. Consider the previous query that used the GROUPING SETS modifier and how it can be rewritten using the ROLLUP keyword (pay attention to the attribute order in the GROUP BY and ORDER BY clauses):

```
SELECT d1.nom_article, d2.regió, d3.mesAny, SUM(fet.articles)
FROM Vendes fet, Producte d1, Lloc d2, Temps d3
WHERE fet.IDProdcute=d1.ID AND fet.IDLloc=d2.ID AND fet.IDTemps=d3.ID
   AND d1.nom_article IN ('Bolígrafs', 'Gomes') AND d2.regió='Catalunya'
   AND d3.mesAny IN ('Gener2002','Febrer2002')
GROUP BY ROLLUP (d2.regió, d1.nom_article, d3.mesAny);
ORDER BY d2.regió, d3.mesAny, d1.nom_article;
```

The query output would be:

nom_article	regió	mesAny	Articles
Bolígrafs	Catalunya	Gener02	275827
Gomes	Catalunya	Gener02	784172
Bolígrafs	Catalunya	Febrer02	290918
Gomes	Catalunya	Febrer02	918012
Bolígrafs	Catalunya	NULL	566745
Gomes	Catalunya	NULL	1702184
NULL	Catalunya	NULL	2268929
NULL	NULL	NULL	2268929

The first four rows correspond to the "GROUP BY d2.regió, d1.nom_article, d3.mesAny"; next two to the "GROUP BY d2.regió, d1.nom_article"; next one to the "GROUP BY d2.regió"; and the last one to the "GROUP BY ()" (note that from the SQL'99 standard on it is allowed to write GROUP BY(); i.e., with the empty list). If we happen to have a single value for `d2.regió` (e.g., Catalunya), the two last rows happen to be identical.

Realize, accordingly, that `GROUP BY ROLL-UP (`a_1`,..,`a_n`)` corresponds to:

```
GROUP BY GROUPING SETS ((a1,...,an),
                        (a1,...,an-1),
                           ...
                        (a1),
                        ())
```

Importantly, note these two relevant features. Firstly, the order you write the attributes in the ROLLUP list does matter (it determines the aggregations to be performed). However, the attribute order specified in the ORDER BY clause does not affect the query result (only how it is presented to the user). Secondly, we can avoid producing the last row by fixing the corresponding dimension to a single value (in our case, it is fixed to Catalunya, check the slicer over the lloc dimension in the query) and forcing this value to be present in all aggregations computed (i.e., placing it in the GROUP BY but out of the ROLLUP expression):

```
SELECT d1.nom_article, d2.regió, d3.mesAny, SUM(fet.articles)
FROM Vendes fet, Producte d1, Lloc d2, Temps d3
WHERE fet.IDProdcute=d1.ID AND fet.IDLloc=d2.ID AND fet.IDTemps=d3.ID
   AND d1.nom_article IN ('Bolígrafs', 'Gomes') AND d2.regió='Catalunya'
   AND d3.mesAny IN ('Gener2002','Febrer2002')
GROUP BY d2.regió, ROLL-UP (d1.nom_article, d3.mesAny);
ORDER BY d2.regió, d3.mesAny, d1.nom_article;
```

This query output is exactly the same as the one presented before but without the last row.

Note that we can rewrite the GROUPING SETS exemplifying query presented in previous section (which contained four aggregations) as follows:

```
SELECT d1.nom_article, d2.regió, d3.mesAny, SUM(fet.articles)
FROM Vendes fet, Producte d1, Lloc d2, Temps d3
```

```
WHERE fet.IDProdcute=d1.ID AND fet.IDLloc=d2.ID AND fet.IDTemps=d3.ID
   AND d1.nom_article IN ('Bolígrafs', 'Gomes') AND d2.regió='Catalunya'
   AND d3.mesAny IN ('Gener2002','Febrer2002)
GROUP BY GROUPING SETS ((d2.regió, ROLL-UP (d1.nom_article, d3.mesAny)),
                        (d2.regió, d3.mesAny))
ORDER BY d1.nom_article, d2.regió, d3.mesAny;
```

5.2.2. CUBE

ROLLUP saves us from writing down all attribute combinations in order to produce the desired granularities. However, we are still forced to write some of them (check the previous example). But it is possible to produce all the combinations needed for the GROUPING SETS exemplifying query by using the CUBE keyword. Check the following query; it produces the 9 cells of that example in a more concise way:

```
SELECT d1.nom_article, d2.regió, d3.mesAny, SUM(fet.articles)
FROM Vendes fet, Producte d1, Lloc d2, Temps d3
WHERE fet.IDProdcute=d1.ID AND fet.IDLloc=d2.ID AND fet.IDTemps=d3.ID
   AND d1.nom_article IN ('Bolígrafs', 'Gomes') AND d2.regió='Catalunya'
   AND d3.mesAny IN ('Gener2002','Febrer2002)
GROUP BY d2.regió, CUBE (d1.nom_article, d3.mesAny);
ORDER BY d1.nom_article, d2.regió, d3.mesAny;
```

Thus, `GROUP BY CUBE (a, b, c)` corresponds to:

```
GROUP BY GROUPING SETS ((a,b,c),
                        (a,b),
                        (a,c),
                        (b,c),
                        (a),
                        (b),
                        (c),
                        ())
```

In addition, CUBE and ROLLUP can be combined in the same GROUP BY expression to obtain the desired attribute combinations. For example:

`GROUP BY CUBE(a,b), ROLLUP(c,d)` corresponds to:

```
GROUP BY GROUPING SETS ((a,b,c,d),
                        (a,b,c),
```

```
(a,b),

(a,c,d),

(a,c),

(a),

(b,c,d),

(b,c),

(b),

(c,d),

(c),

())
```

Indeed, the SQL'99 standard allows combining CUBE, ROLLUP and GROUPING SETS to a certain extent. We suggest to practice with a toy example and realize which combinations make sense and which do not.

CONCLUSIONS

The CUBE, ROLLUP and GROUPING SETS keywords were introduced in the SQL'99 standard as modifiers for the GROUP BY clause. They were intended to facilitate aggregation computations and thus, they can be considered as syntactic sugar. However, they are something else than pure syntactic sugar, as they do improve the system performance since the query optimizer receives valuable additional information about the queries to be carried out.