EXERCISE AT HOME: MODERATELY DIFFICULT REPORTS WITH COMPARISON ACROSS AGGREGATION LEVELS

Sales(Customer, Product, Brand, Date, City, Region, Area, Quantity, Revenue, Margin)

Revenue by Brand and Product January 2008				
Brand	**Product**	**Revenue (€)**	**Percent of Brand Revenue**	**Percent of Total Revenue**
M1	P1	175,000	45%	21%
	P2	96,000	25%	12%
	P3	114,000	30%	14%
M1	**All products**	**385,000**	**100%**	**47%**
M2	P4	102,400	23%	12%
	P5	96,200	22%	12%
	P6	124,000	28%	15%
	P7	120,000	27%	14%
M2	**All products**	**442,600**	100%	**53%**
All brands		**827,000**		100%

SOLUTION

GROUPING(A) = 1 for GROUP BY's where A is not included

```
WITH temp AS
 ( SELECT Brand, Product,SUM(Revenue) AS prodRevenue,
    SUM(CASE WHEN GROUPING(Product)=1 THEN 0 ELSE SUM(Revenue) END)
        OVER(PARTITION BY Brand) AS TotalBrand,
    SUM(CASE WHEN GROUPING(Product)=1 THEN 0 ELSE SUM(Revenue) END)
        OVER() AS Total
FROM sales
GROUP BY ROLLUP(Brand, Product)
)
SELECT Brand, Product, prodRevenue,
       CASE WHEN TotalBrand>0 THEN 100*prodRevenue/TotalBrand ELSE 0 END as pctBrand,
       100*prodRevenue/Total ELSE 0 END as pctTotal
FROM temp
ORDER BY Brand, Product
```

TotalBrand = 0 when Brand = NULL, coming from grouping on ()

VERY DIFFICULT REPORTS WITHOUT ANALYTIC SQL: EXERCISE AT HOME!

Sales(Customer, Product, Brand, Date, City, Region, Area, Quantity, Revenue, Margin)

We want to partition the customers into four groups:

- **Top5%**, with 5% of customers with the highest amount of revenues.
- **Next15%**, with 15% of other customers with the highest amount of revenues.
- **Middle30%**, with 30% of other customers with the highest amount of revenues.
- **Bottom50%**, with 50 % of the customers with the lowest amount of revenues.

For each customer group we want to know their number, and the percentage of the sum of their revenues compared to total revenue of all sales.

Group	Number of customers	Percent of total revenue
Top5%	1	20
Next15%	3	50
Middle30%	6	20
Bottom50%	10	10

SOLUTION

```
WITH temp AS
 ( SELECT Customer, SUM(Revenue) AS CustRevenue,
          CUME_DIST() OVER (ORDER BY SUM(Revenue) DESC) AS Cum
FROM sales
GROUP BY Customer
), temp2 AS
 ( SELECT Customer, CustRevenue,
          CASE WHEN Cum <= 0.05 THEN 'Top5%'
               WHEN Cum <= 0.20 THEN 'Next15%'
               WHEN Cum <= 0.50 THEN 'Middle30%'
               ELSE 'Bottom50%'
           END AS Gr
FROM temp
)
SELECT Gr, COUNT(*) AS NCustomers,
          100.0*SUM(CustRevenue)/SUM(SUM(CustRevenue)) OVER ()  AS PctRevenue
FROM temp2
GROUP BY Gr
ORDER BY Gr DESC
```

EXERCISE AT HOME: SOLUTION USING LAG-LEAD (and NO JOIN)

Sales(Customer, Product, Brand, Date, City, Region, Area, Quantity, Revenue, Margin)

Comparison between Revenue by Brand and by Product 2009 – 2008				
Brand	Product	Revenue (€) 2009	Revenue (€) 2008	Delta (%)
B1	P1	2 100	13 560	–546
	P2	3 720	23 640	–535
	P3	15 300	20 340	–33
B2	P4	12 600	1 440	89
	P5	22 500	2 100	91
	P6	48 300		100

Delta = 100 x (Revenue2009 - Revenue2008)/Revenue2009

A product may have been sold in one year, but not in the other !

SOLUTION USING LAG-LEAD (and NO JOIN)

```
WITH temp AS (
      SELECT Brand, Product, Year(Date) AS Year, SUM(Revenue) AS Revenue
      FROM Sales
      WHERE Year(Date) IN (2008, 2009)
      GROUP BY Brand, Product, Year(Date)
),
laglead AS (
      SELECT Brand, Product, Year, Revenue,
            LAG(Revenue, 1, 0) OVER(PARTITION BY Brand, Product ORDER BY Year) AS PrevR,
            LEAD(Revenue) OVER(PARTITION BY Brand, Product ORDER BY Year) AS NextR
      FROM temp
)
SELECT Brand, Product, Revenue AS Revenue2009, PrevR AS Revenue2008,
            CASE WHEN Year = 2008 THEN -100
                  ELSE Round(100*(Revenue-PrevR)/Revenue) END AS Delta
FROM laglead
WHERE Year = 2009 OR (Year = 2008 AND NextR IS NULL)
ORDER BY Brand, Product
```

VERY DIFFICULT REPORTS WITHOUT ANALYTIC SQL: RUNNING TOTALS

Sales(Customer, Product, Brand, Date, City, Region, Area, Quantity, Revenue, Margin)

Product P1 Revenue by Quarter and Month
Year 2009

Quarter	Month	Revenue (€)	Revenue QtoD (€)	Revenue YtoD (€)
Q1	January	16 500	16 500	16 500
Q1	February	14 220	30 720	30 720
Q1	March	27 480	58 200	58 200
Q2	April	7 920	7 920	66 120
Q2	May	1 200	9 120	67 320
Q2	June	1 260	10 380	68 580
Q3	July	5 400	5 400	73 980
Q3	August	11 730	17 130	85 710
Q3	September	10 860	27 990	96 570
Q4	October	5 850	5 850	102 420
Q4	November	2 100	7 950	104 520
Q4	December			

WINDOWING

```
<AggregateFunction>(<expr>)
OVER(
      [PARTITION BY <attribute list>]
      [ORDER BY <sort attribute list>
       [<ROWS or RANGE> <window size specification>]]
     ) [ AS Ide ]
```

Windowing functions are used to compute cumulative, moving and centered aggregates.

Window functions add a value to each row that depends on the other rows in the window, based on distance (ROWS) or value (RANGE)

Examples of window specifications (together with **ORDER BY Date**):

- **ROWS UNBOUNDED PRECEDING.** The window begin with the first record of the partition and ends with the current record.
- **ROWS BETWEEN 5 PRECEDING AND 5 FOLLOWING.** The window include all records that fall within the given offset of preceding and following number of rows.
- **RANGE BETWEEN INTERVAL 5 DAYS PRECEDING AND CURRENT ROW.** The window include all records that fall within 5 days from current date.

WINDOWING EXAMPLE

Sales(Customer, Product, Brand, Date, City, Region, Area, Quantity, Revenue, Margin)

Product P1 Revenue by Quarter and Month Year 2009				
Quarter	**Month**	**Revenue (€)**	**Revenue QtoD (€)**	**Revenue YtoD (€)**
Q1	January	16 500	16 500	16 500
Q1	February	14 220	30 720	30 720
Q1	March	27 480	58 200	58 200
Q2	April	7 920	7 920	66 120
Q2	May	1 200	9 120	67 320
Q2	June	1 260	10 380	68 580
Q3	July	5 400	5 400	73 980
Q3	August	11 730	17 130	85 710
Q3	September	10 860	27 990	96 570
Q4	October	5 850	5 850	102 420
Q4	November	2 100	7 950	104 520
Q4	December			

WINDOWING EXAMPLE

Sales(Customer, Product, Brand, Date, City, Region, Area, Quantity, Revenue, Margin)

```
SELECT    Quarter_Name(QUARTER(Date)) AS Quarter
          , Month_Name(MONTH(Date)) AS Month
          , SUM(Revenue) AS Revenue
          , SUM(SUM(Revenue)) OVER
              (PARTITION BY QUARTER(Date)
                  ORDER BY MONTH(Date)
                      ROWS UNBOUNDED PRECEDING) AS RevenueQToD
          , SUM(SUM(Revenue)) OVER
                  (ORDER BY MONTH(Date)
                      ROWS UNBOUNDED PRECEDING) AS RevenueYToD
FROM      Sales
WHERE     YEAR(Date) = 2009
GROUP BY  QUARTER(Date), MONTH(Date)
ORDER BY  Quarter, Month;
```

Product P1 Revenue by Quarter and Month
Year 2009

Quarter	Month	Revenue (€)	Revenue QtoD (€)	Revenue YtoD (€)
Q1	January	16 500	16 500	16 500
Q1	February	14 220	30 720	30 720
Q1	March	27 480	58 200	58 200
Q2	April	7 920	7 920	66 120
Q2	May	1 200	9 120	67 320
Q2	June	1 260	10 380	68 580
Q3	July	5 400	5 400	73 980
Q3	August	11 730	17 130	85 710
Q3	September	10 860	27 990	96 570
Q4	October	5 850	5 850	102 420
Q4	November	2 100	7 950	104 520
Q4	December			

EXAMPLE

Sales(Customer, Product, Brand, Date, City, Region, Area, Quantity, Revenue, Margin)

A moving average of total revenue, with a moving window of 3 months, by month.

```
SELECT    MONTH(Date) AS Month

FROM      Sales
GROUP BY  MONTH(Date)
ORDER BY  Month;
```

EXAMPLE

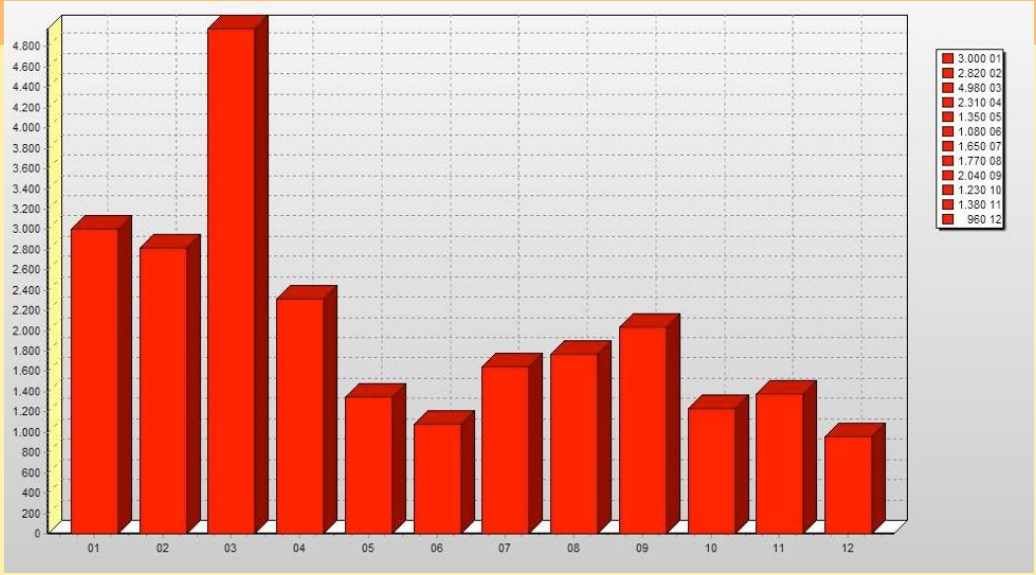
Sales(Customer, Product, Brand, Date, City, Region, Area, Quantity, Revenue, Margin)

A moving average of total revenue, with a moving window of 3 months, by month.

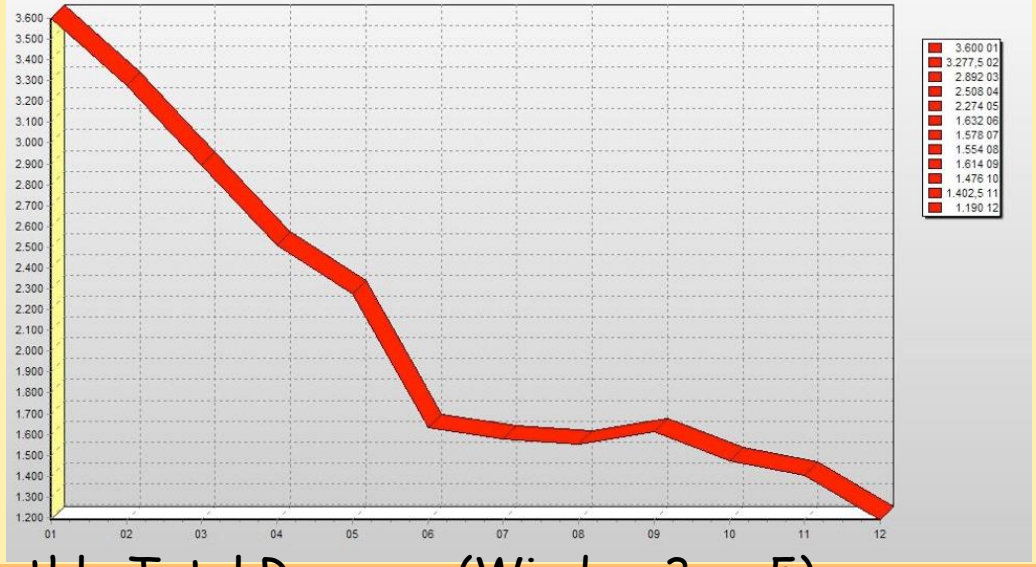
```
SELECT      MONTH(Date) AS Month
            , ROUND(AVG(SUM(Revenue))
                OVER (ORDER BY MONTH(Date)
                    ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING), 2)
                AS MovingAverageRevenue
FROM        Sales
GROUP BY    MONTH(Date)
ORDER BY    Month;
```

Result visualization ...

MOVING AVERAGE OF TOTAL REVENUE BY MONTHS OF A YEAR



Monthly Total Revenue



Moving Average Monthly Total Revenue (Window 3 or 5)

SUMMARY

SQL is not select-from-where **only**.

Grouping and **aggregation** is a major part of SQL.

SQL has been extended for OLAP operations, because of intensive data warehouse applications during the last decade.

Make sure you understand SQL. It is much more than syntax.