Analytical Functions

Analytic functions enhance both database performance and developer productivity. They are valuable for all types of processing, ranging from interactive decision support to batch report jobs.

We use Analytical functions for 4 reasons

- Improve Query Speed
- Enhanced Developer Productivity
- Minimized Learning Effort
- Standardized Syntax

Oracle has created four families of Analytic functions

Ranking Family

This family supports business questions like “show the top 10 and bottom 10 salesperson per each region” or “show, for each region, salespersons that make up 25% of the sales”.

Examples: RANK, DENSE_RANK, PERCENT_RANK, CUME_DIST and NTILE functions.

Window Aggregate Family

This family addresses questions like “show the 13-week moving average of a stock price” or “show cumulative sum of sales per each region.” The new features provide moving and cumulative processing for all the SQL aggregate functions including AVG, SUM, MIN, MAX, COUNT, VARIANCE and STDDEV

Reporting Aggregate Family

One of the most common types of calculations is the comparison of a non-aggregate value to an aggregate value. All percent-of-total and market share calculations require this processing. The new family provides reporting aggregate processing for all SQL functions including AVG, SUM, MIN,MAX, COUNT, VARIANCE and STDDEV

LAG/LEAD Family

Studying change and variation is at the heart of analysis. Necessarily, this involves comparing the values of different rows in a table. While this has been possible in SQL, usually through self-joins, it has not been efficient or easy to formulate. The LAG/LEAD family enables queries to compare different rows of a table simply by specifying an offset from the current row.

Examples:

Show the Sales as a percentage of total Sales

```
SELECT SP.FIRST_NAME,
          SUM (TOTAL_AMOUNT) AS TOTAL_AMOUNT,
          RATIO_TO_REPORT (SUM (TOTAL_AMOUNT)) OVER () * 100 AS RATIO_PERC
    FROM SALES S, SALESPERSON SP
   WHERE S.SALESPERSON_ID = SP.SALESPERSON_ID
GROUP BY SP.FIRST_NAME
```

Combine detail data with Aggregate data

```
SELECT S.SALES_DATE,
          S.ORDER_ID,
          S.PRODUCT_ID,
          S.CUSTOMER_ID,
          S.TOTAL_AMOUNT,
         AVG (TOTAL_AMOUNT) OVER (PARTITION BY SALES_DATE) AS AVG_TOTAL_AMOUNT
  FROM SALES S
```

Dividing data into Groups/Bands

```
SELECT SP.FIRST_NAME,
          SUM (TOTAL_AMOUNT) AS TOTAL_AMOUNT,
          NTILE (3) OVER (ORDER BY SUM (TOTAL_AMOUNT) DESC) AS BUCKET_LIST
    FROM SALES S, SALESPERSON SP
   WHERE S.SALESPERSON_ID = SP.SALESPERSON_ID
GROUP BY SP.FIRST_NAME
```

Show the top 3 and bottom 3 sales person by Month

```
SELECT * FROM
(
  SELECT TRUNC (S.SALES_DATE, 'mon') AS SALES_MONTH,
           SP.FIRST_NAME,
          SUM (TOTAL_AMOUNT) AS TOTAL_AMOUNT,
          RANK () OVER (PARTITION BY TRUNC (S.SALES_DATE, 'mon')
                        ORDER BY SUM (TOTAL_AMOUNT) DESC) AS SALESPERSON_RANK_TOP,
          RANK ()  OVER (PARTITION BY TRUNC (S.SALES_DATE, 'mon')
                         ORDER BY SUM (TOTAL_AMOUNT) ASC) AS SALESPERSON_RANK_BOTTOM
    FROM SALES S, SALESPERSON SP
   WHERE S.SALESPERSON_ID = SP.SALESPERSON_ID
GROUP BY TRUNC (S.SALES_DATE, 'mon'), S.SALESPERSON_ID, SP.FIRST_NAME
)
WHERE SALESPERSON_RANK_TOP <= 3 OR SALESPERSON_RANK_BOTTOM <= 3
```

Show the sales growth across time

```
SELECT SALES_MONTH,
        TOTAL_AMOUNT AS CURRENT_MONTH,
        PREVIOUS_MONTH,
        ROUND(((TOTAL_AMOUNT - PREVIOUS_MONTH)/PREVIOUS_MONTH) * 100,2) AS GROWTH_PERC
FROM
(
SELECT TRUNC (S.SALES_DATE, 'mon') AS SALES_MONTH,
        SUM (TOTAL_AMOUNT) AS TOTAL_AMOUNT,
        LAG(SUM(TOTAL_AMOUNT),1) OVER(ORDER BY TRUNC(S.SALES_DATE, 'mon')) AS
                                                                    PREVIOUS_MONTH,
        LEAD(SUM(TOTAL_AMOUNT),1) OVER(ORDER BY TRUNC(S.SALES_DATE, 'mon')) AS
                                                                    NEXT_MONTH
   FROM SALES S, SALESPERSON SP
  WHERE S.SALESPERSON_ID = SP.SALESPERSON_ID
GROUP BY TRUNC (S.SALES_DATE, 'mon')
)
```