MTN.BI.03

SQL FOR ANALYSIS

Windowing Aggregate Functions

Author: Aliaksei Belablotski
Senior Software Engineer
Aliaksei_Belablotski@epam.com

Objectives

1. Analytic Functions Essential Concepts
2. Windowing Aggregate Functions
3. Logical and Physical Offset
4. Varying Window Size for Each Row
5. LISTAGG Function
6. NTH_VALUE Function

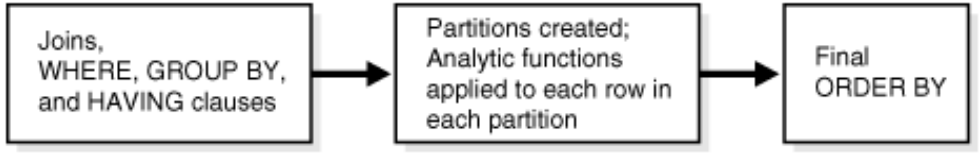
WINDOWING AGGREGATE FUNCTIONS

Analytic Functions

Type	Used For
Ranking	Calculating ranks, percentiles, and n-tiles of the values in a result set.
Windowing	**Calculating cumulative and moving aggregates. Works with these functions: SUM, AVG, MIN, MAX, COUNT, VARIANCE, STDDEV, FIRST_VALUE, LAST_VALUE, and statistical functions. Note that the DISTINCT keyword is not supported in windowing functions except for MAX and MIN.**
Reporting	Works with: SUM, AVG, MIN, MAX, COUNT (with/without DISTINCT), VARIANCE, STDDEV, RATIO_TO_REPORT, and statistical functions. Note that the DISTINCT keyword may be used in those reporting functions that support DISTINCT in aggregate mode.
LAG/LEAD	Finding a value in a row a specified number of rows from a current row.
FIRST/LAST	First or last value in an ordered group.
Linear Regression	Calculating linear regression and other statistics.
Inverse Percentile	The value in a data set that corresponds to a specified percentile.
Hypothetical Rank and Distribution	The rank or percentile that a row would have if inserted into a specified data set.

Analytic Functions Essential Concepts

1. **Processing order**



2. **Result set partitions**

   Partitions are created after the groups defined with GROUP BY clauses, so they are available to any aggregate results such as sums and averages. Partition divisions may be based upon any desired columns or expressions.

3. **Window**

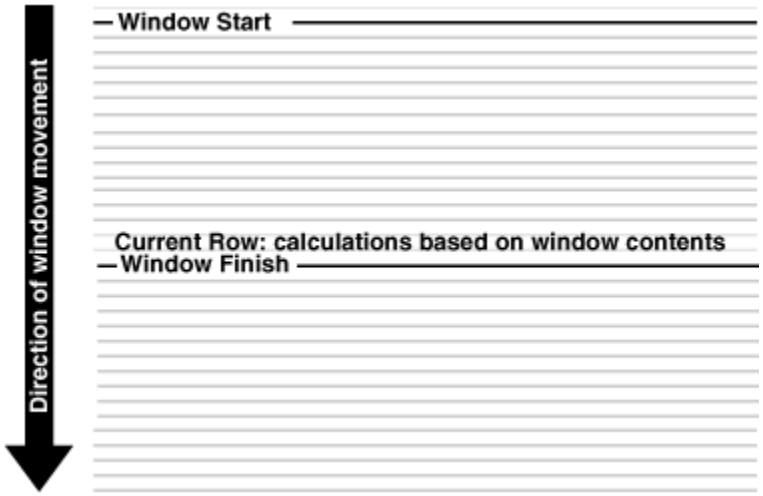
   For each row in a partition, you can define a sliding window of data. This window determines the range of rows used to perform the calculations for the current row.

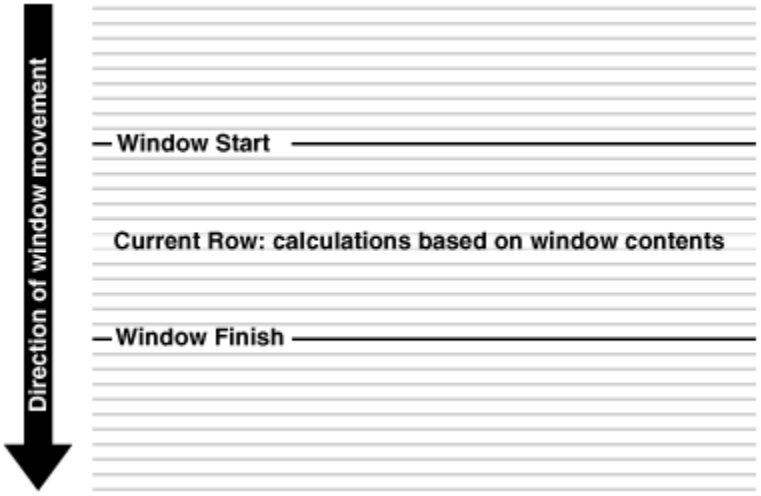
4. **Current row**

   Each calculation performed with an analytic function is based on a current row within a partition.

Sliding Window

1. For each row in a partition, you can define a sliding window of data.
2. Window determines the range of rows used to perform the calculations for the current row.
3. The window has a starting row and an ending row.
4. Window may move at one or both ends.
5. When a window is near a border, the function returns results for only the available rows.
6. When using window functions, the current row is included during calculations.





Windowing Aggregate Functions

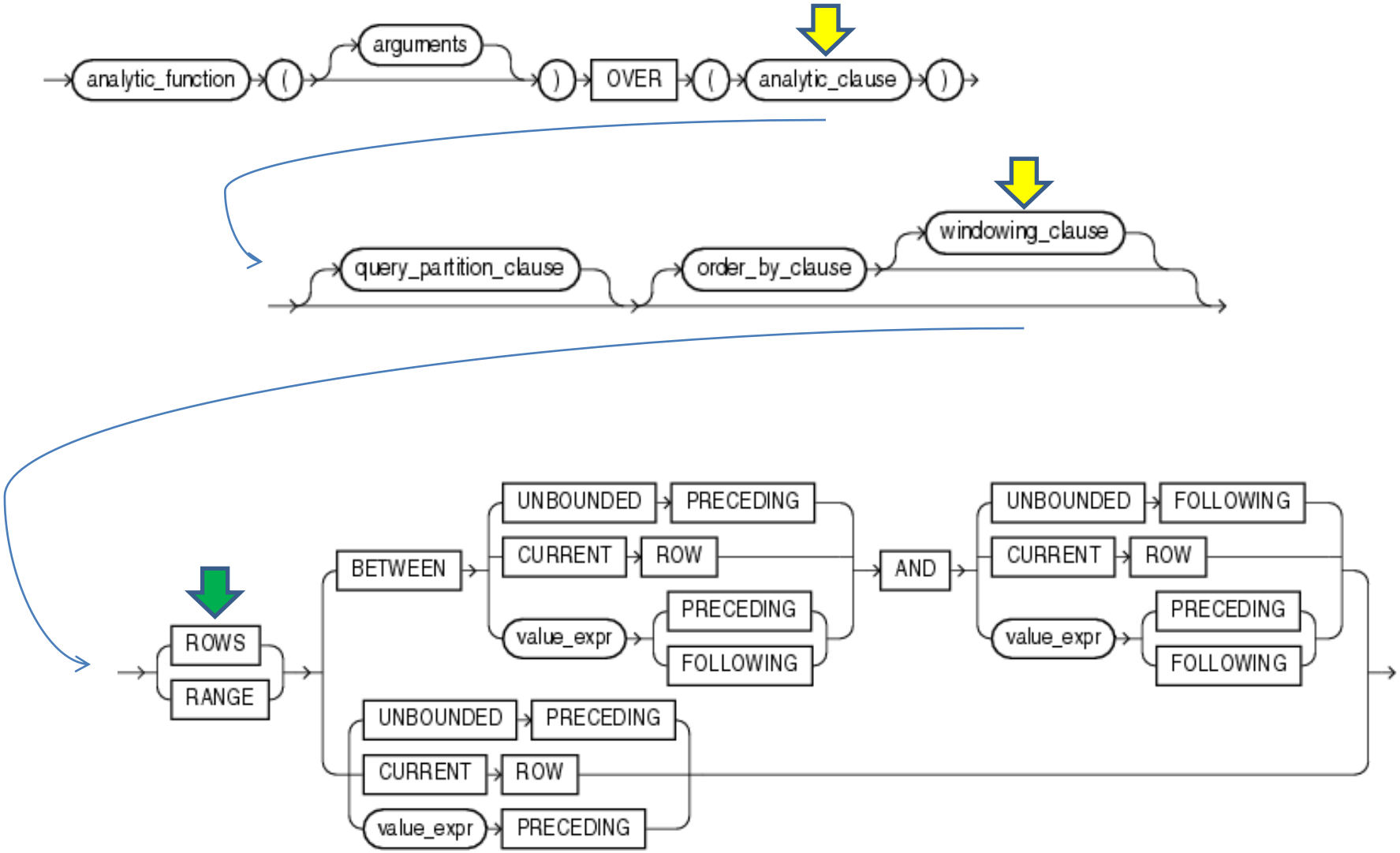
1. Windowing functions can be used to compute cumulative and centered aggregates. They return a value for each row in the table, which depends on other rows in the corresponding window.
2. With windowing aggregate functions, you can calculate moving and cumulative versions of **SUM**, **AVERAGE**, **COUNT**, **MAX**, **MIN**, and other aggregate functions.
3. They can be used only in the **SELECT** and **ORDER BY** clauses of the query.
4. Windowing aggregate functions include the convenient **FIRST_VALUE**, which returns the first value in the window; and **LAST_VALUE**, which returns the last value in the window.
5. These functions provide access to more than one row of a table without a self-join.

```
analytic_function([ arguments ])
   OVER (analytic_clause)

 where analytic_clause =
     [ query_partition_clause ]
     [ order_by_clause [ windowing_clause ] ]

and query_partition_clause =
    PARTITION BY
      { value_expr[, value_expr ]...
      | ( value_expr[, value_expr ]... )
      }
 and windowing_clause =
     { ROWS | RANGE }
     { BETWEEN
       { UNBOUNDED PRECEDING
     | CURRENT ROW
     | value_expr { PRECEDING | FOLLOWING }
     }
     AND
     { UNBOUNDED FOLLOWING
     | CURRENT ROW
     | value_expr { PRECEDING | FOLLOWING }
     }
   | { UNBOUNDED PRECEDING
     | CURRENT ROW
     | value_expr PRECEDING
     }
   }
```

Windowing Aggregate Functions Syntax



Windowing Functions with Logical / Physical Offset

If you specified ROWS:

- **value_expr is a physical offset. It must be a constant or expression and must evaluate to a positive numeric value.**
- **If value_expr is part of the start point, then it must evaluate to a row before the end point.**

If you specified RANGE:

- **value_expr is a logical offset. It must be a constant or expression that evaluates to a positive numeric value or an interval literal.**
- **You can specify only one expression in the order_by_clause**
- **If value_expr evaluates to a numeric value, then the ORDER BY expr must be a numeric or DATE datatype.**
- **If value_expr evaluates to an interval value, then the ORDER BY expr must be a DATE datatype.**

If you omit the windowing clause entirely, then the default is RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW.

Windowing Functions with Logical Offset

A logical offset can be specified with constants such as **RANGE 10 PRECEDING**, or an expression that evaluates to a constant, or by an interval specification like **RANGE INTERVAL N DAY/MONTH/YEAR PRECEDING** or an expression that evaluates to an interval.

1. With logical offset, there can **only be one expression in the ORDER BY** expression list in the function, with type compatible to NUMERIC if offset is numeric, or DATE if an interval is specified.
2. An analytic function that uses the RANGE keyword can use **multiple sort keys in its ORDER BY** clause if it specifies either of these two windows:
   - RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW. (The short form of this is RANGE UNBOUNDED PRECEDING, which can also be used.)
   - RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING.
3. Window boundaries that do not meet these conditions can have only one sort key in the analytic function's ORDER BY clause.

Cumulative amount_sold by customer ID by quarter in 1999

```
SELECT C.CUST_ID, T.CALENDAR_QUARTER_DESC,
  TO_CHAR (SUM(AMOUNT_SOLD), '9,999,999,999.99') AS Q_SALES,
  TO_CHAR(SUM(SUM(AMOUNT_SOLD))
    OVER (PARTITION BY C.CUST_ID ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
          ROWS UNBOUNDED PRECEDING), '9,999,999,999.99') AS CUM_SALES
  FROM SALES S, TIMES T, CUSTOMERS C
  WHERE S.TIME_ID=T.TIME_ID AND S.CUST_ID=C.CUST_ID AND T.CALENDAR_YEAR=2000
    AND C.CUST_ID IN (2595, 9646, 11111)
  GROUP BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
  ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC;
```

	CUST_ID	CALENDAR_QUARTER_DESC	Q_SALES	CUM_SALES
1	2595	2000-01	659.92	659.92
2	2595	2000-02	224.79	884.71
3	2595	2000-03	313.90	1,198.61
4	2595	2000-04	6,015.08	7,213.69
5	9646	2000-01	1,337.09	1,337.09
6	9646	2000-02	185.67	1,522.76
7	9646	2000-03	203.86	1,726.62
8	9646	2000-04	458.29	2,184.91
9	11111	2000-01	43.18	43.18
10	11111	2000-02	33.33	76.51
11	11111	2000-03	579.73	656.24
12	11111	2000-04	307.58	963.82

Quarter Report

```
SELECT C.CUST_ID, T.CALENDAR_QUARTER_DESC,
  TO_CHAR (SUM(AMOUNT_SOLD), '9,999,999,999.99') AS Q_SALES,
  TO_CHAR(SUM(SUM(AMOUNT_SOLD))
    OVER (PARTITION BY C.CUST_ID ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
          ROWS BETWEEN 1 PRECEDING AND CURRENT ROW) - SUM(AMOUNT_SOLD), '9,999,999,990.99') AS PREV_Q,
  TO_CHAR(SUM(AMOUNT_SOLD) * 2 - SUM(SUM(AMOUNT_SOLD))
    OVER (PARTITION BY C.CUST_ID ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
          ROWS BETWEEN 1 PRECEDING AND CURRENT ROW), '9,999,999,990.99') AS DELTA_Q
  FROM SALES S, TIMES T, CUSTOMERS C
  WHERE S.TIME_ID=T.TIME_ID AND S.CUST_ID=C.CUST_ID AND T.CALENDAR_YEAR=2000
    AND C.CUST_ID IN (2595, 9646, 11111)
  GROUP BY C.CUST_ID, T.CALENDAR_QUARTER_DESC, T.CALENDAR_QUARTER_NUMBER
  ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC;
```

	CUST_ID	CALENDAR_QUARTER_DESC	Q_SALES	PREV_Q	DELTA_Q
1	2595	2000-01	659.92	0.00	659.92
2	2595	2000-02	224.79	659.92	-435.13
3	2595	2000-03	313.90	224.79	89.11
4	2595	2000-04	6,015.08	313.90	5,701.18
5	9646	2000-01	1,337.09	0.00	1,337.09
6	9646	2000-02	185.67	1,337.09	-1,151.42
7	9646	2000-03	203.86	185.67	18.19
8	9646	2000-04	458.29	203.86	254.43
9	11111	2000-01	43.18	0.00	43.18
10	11111	2000-02	33.33	43.18	-9.85
11	11111	2000-03	579.73	33.33	546.40
12	11111	2000-04	307.58	579.73	-272.15

Quarter Report

```
SELECT C.CUST_ID, T.CALENDAR_QUARTER_DESC,
  TO_CHAR (SUM(AMOUNT_SOLD), '9,999,999,999.99') AS Q_SALES,
  TO_CHAR(SUM(SUM(AMOUNT_SOLD))
    OVER (PARTITION BY C.CUST_ID ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
          ROWS BETWEEN 1 PRECEDING AND CURRENT ROW) - SUM(AMOUNT_SOLD), '9,999,999,990.99') AS PREV_Q,
  TO_CHAR(SUM(AMOUNT_SOLD) * 2 - SUM(SUM(AMOUNT_SOLD))
    OVER (PARTITION BY C.CUST_ID ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
          ROWS BETWEEN 1 PRECEDING AND CURRENT ROW), '9,999,999,990.99') AS DELTA_Q,
  CASE WHEN (SUM(SUM(AMOUNT_SOLD))
             OVER (PARTITION BY C.CUST_ID ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
                   ROWS BETWEEN 1 PRECEDING AND CURRENT ROW) = SUM(AMOUNT_SOLD)) THEN 'N/A'
    ELSE TO_CHAR(((SUM(AMOUNT_SOLD) * 2 - SUM(SUM(AMOUNT_SOLD))
      OVER (PARTITION BY C.CUST_ID ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
            ROWS BETWEEN 1 PRECEDING AND CURRENT ROW)) / (SUM(SUM(AMOUNT_SOLD))
      OVER (PARTITION BY C.CUST_ID ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
            ROWS BETWEEN 1 PRECEDING AND CURRENT ROW)- SUM(AMOUNT_SOLD)))*100, '9,999,999,990.99')|| '%'
    END AS DELTA_Q_PRC
  FROM SALES S, TIMES T, CUSTOMERS C
  WHERE S.TIME_ID=T.TIME_ID AND S.CUST_ID=C.CUST_ID AND T.CALENDAR_YEAR=2000
    AND C.CUST_ID IN (2595, 9646)
  GROUP BY C.CUST_ID, T.CALENDAR_QUARTER_DESC, T.CALENDAR_QUARTER_NUMBER
  ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC;
```

	CUST_ID	CALENDAR_QUARTER_DESC	Q_SALES	PREV_Q	DELTA_Q	DELTA_Q_PRC
1	2595	2000-01	659.92	0.00	659.92	N/A
2	2595	2000-02	224.79	659.92	-435.13	-65.94%
3	2595	2000-03	313.90	224.79	89.11	39.64%
4	2595	2000-04	6,015.08	313.90	5,701.18	1,816.24%
5	9646	2000-01	1,337.09	0.00	1,337.09	N/A
6	9646	2000-02	185.67	1,337.09	-1,151.42	-86.11%
7	9646	2000-03	203.86	185.67	18.19	9.80%
8	9646	2000-04	458.29	203.86	254.43	124.81%

Quarter Report

```
SELECT C.CUST_ID, T.CALENDAR_QUARTER_DESC,
  TO_CHAR (SUM(AMOUNT_SOLD), '9,999,999,999.99') AS Q_SALES,
  TO_CHAR(FIRST_VALUE(SUM(AMOUNT_SOLD))
    OVER (PARTITION BY C.CUST_ID ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
          ROWS BETWEEN 1 PRECEDING AND CURRENT ROW), '9,999,999,990.99') AS PREV_Q,
  TO_CHAR(SUM(AMOUNT_SOLD) - FIRST_VALUE(SUM(AMOUNT_SOLD))
    OVER (PARTITION BY C.CUST_ID ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
          ROWS BETWEEN 1 PRECEDING AND CURRENT ROW), '9,999,999,990.99') AS DELTA_Q
  FROM SALES S, TIMES T, CUSTOMERS C
  WHERE S.TIME_ID=T.TIME_ID AND S.CUST_ID=C.CUST_ID AND T.CALENDAR_YEAR=2000
    AND C.CUST_ID IN (2595, 9646, 11111)
  GROUP BY C.CUST_ID, T.CALENDAR_QUARTER_DESC, T.CALENDAR_QUARTER_NUMBER
  ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC;
```

	CUST_ID	CALENDAR_QUARTER_DESC	Q_SALES	PREV_Q	DELTA_Q
1	2595	2000-01	659.92	659.92	0.00
2	2595	2000-02	224.79	659.92	-435.13
3	2595	2000-03	313.90	224.79	89.11
4	2595	2000-04	6,015.08	313.90	5,701.18
5	9646	2000-01	1,337.09	1,337.09	0.00
6	9646	2000-02	185.67	1,337.09	-1,151.42
7	9646	2000-03	203.86	185.67	18.19
8	9646	2000-04	458.29	203.86	254.43
9	11111	2000-01	43.18	43.18	0.00
10	11111	2000-02	33.33	43.18	-9.85
11	11111	2000-03	579.73	33.33	546.40
12	11111	2000-04	307.58	579.73	-272.15

Quarter Report

```
SELECT CUST_ID, CALENDAR_QUARTER_DESC, Q_SALES,
  CASE WHEN SUBSTR(CALENDAR_QUARTER_DESC,-1,1)='1' THEN 'N/A'
    ELSE TO_CHAR(PREV_Q, '9,999,999,990.99')
  END AS PREV_Q,
  CASE WHEN SUBSTR(CALENDAR_QUARTER_DESC,-1,1)='1' THEN 'N/A'
    ELSE TO_CHAR(Q_SALES - PREV_Q, '9,999,999,990.99')
  END as DELTA_Q,
  CASE WHEN SUBSTR(CALENDAR_QUARTER_DESC,-1,1)='1' THEN 'N/A'
    ELSE TO_CHAR((Q_SALES - PREV_Q) / PREV_Q * 100, '9,999,999,990.99') || '%'
  END as DELTA_Q_PRC
FROM (
  SELECT C.CUST_ID, T.CALENDAR_QUARTER_DESC,
    SUM(AMOUNT_SOLD) AS Q_SALES,
    FIRST_VALUE(SUM(AMOUNT_SOLD))
      OVER (PARTITION BY C.CUST_ID ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
            ROWS BETWEEN 1 PRECEDING AND CURRENT ROW) AS PREV_Q
    FROM SALES S, TIMES T, CUSTOMERS C
    WHERE S.TIME_ID=T.TIME_ID AND S.CUST_ID=C.CUST_ID AND T.CALENDAR_YEAR=2000
      AND C.CUST_ID IN (2595, 9646, 11111)
    GROUP BY C.CUST_ID, T.CALENDAR_QUARTER_DESC, T.CALENDAR_QUARTER_NUMBER
    ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
);
```

	CUST_ID	CALENDAR_QUARTER_DESC	Q_SALES	PREV_Q	DELTA_Q	DELTA_Q_PRC
1	2595	2000-01	659.92	N/A	N/A	N/A
2	2595	2000-02	224.79	659.92	-435.13	-65.94%
3	2595	2000-03	313.9	224.79	89.11	39.64%
4	2595	2000-04	6015.08	313.90	5,701.18	1,816.24%
5	9646	2000-01	1337.09	N/A	N/A	N/A
6	9646	2000-02	185.67	1,337.09	-1,151.42	-86.11%
7	9646	2000-03	203.86	185.67	18.19	9.80%
8	9646	2000-04	458.29	203.86	254.43	124.81%
9	11111	2000-01	43.18	N/A	N/A	N/A
10	11111	2000-02	33.33	43.18	-9.85	-22.81%
11	11111	2000-03	579.73	33.33	546.40	1,639.36%
12	11111	2000-04	307.58	579.73	-272.15	-46.94%

Quarter Report

```
SELECT CUST_ID, CALENDAR_QUARTER_DESC, Q_SALES,
  CASE WHEN SUBSTR(CALENDAR_QUARTER_DESC,-1,1)='1' THEN 'N/A'
    ELSE TO_CHAR(PREV_Q, '9,999,999,990.99')
  END AS PREV_Q,
  CASE WHEN SUBSTR(CALENDAR_QUARTER_DESC,-1,1)='1' THEN 'N/A'
    ELSE TO_CHAR(Q_SALES - PREV_Q, '9,999,999,990.99')
  END as DELTA_Q,
  CASE WHEN SUBSTR(CALENDAR_QUARTER_DESC,-1,1)='1' THEN 'N/A'
    ELSE TO_CHAR((Q_SALES - PREV_Q) / PREV_Q * 100, '9,999,999,990.99') || '%'
  END as DELTA_Q_PRC
FROM (
  SELECT C.CUST_ID, T.CALENDAR_QUARTER_DESC,
    SUM(AMOUNT_SOLD) AS Q_SALES,
    LAG(SUM(AMOUNT_SOLD))
      OVER (PARTITION BY C.CUST_ID ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC) AS PREV_Q
    FROM SALES S, TIMES T, CUSTOMERS C
    WHERE S.TIME_ID=T.TIME_ID AND S.CUST_ID=C.CUST_ID AND T.CALENDAR_YEAR=2000
      AND C.CUST_ID IN (2595, 9646, 11111)
    GROUP BY C.CUST_ID, T.CALENDAR_QUARTER_DESC, T.CALENDAR_QUARTER_NUMBER
    ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
);
```

	CUST_ID	CALENDAR_QUARTER_DESC	Q_SALES	PREV_Q	DELTA_Q	DELTA_Q_PRC
1	2595	2000-01	659.92	N/A	N/A	N/A
2	2595	2000-02	224.79	659.92	-435.13	-65.94%
3	2595	2000-03	313.9	224.79	89.11	39.64%
4	2595	2000-04	6015.08	313.90	5,701.18	1,816.24%
5	9646	2000-01	1337.09	N/A	N/A	N/A
6	9646	2000-02	185.67	1,337.09	-1,151.42	-86.11%
7	9646	2000-03	203.86	185.67	18.19	9.80%
8	9646	2000-04	458.29	203.86	254.43	124.81%
9	11111	2000-01	43.18	N/A	N/A	N/A
10	11111	2000-02	33.33	43.18	-9.85	-22.81%
11	11111	2000-03	579.73	33.33	546.40	1,639.36%
12	11111	2000-04	307.58	579.73	-272.15	-46.94%

All customers a centered moving average of sales for one week in late December 1999

```
SELECT T.TIME_ID, TO_CHAR (SUM(AMOUNT_SOLD), '9,999,999,999') AS SALES,
 TO_CHAR(AVG(SUM(AMOUNT_SOLD)) OVER (ORDER BY T.TIME_ID
  RANGE BETWEEN INTERVAL '1' DAY PRECEDING AND
  INTERVAL '1' DAY FOLLOWING), '9,999,999,999') AS CENTERED_3_DAY_AVG
FROM SALES S, TIMES T
WHERE S.TIME_ID=T.TIME_ID AND T.CALENDAR_WEEK_NUMBER IN (51)
  AND CALENDAR_YEAR=1999
GROUP BY T.TIME_ID
ORDER BY T.TIME_ID;
```

	TIME_ID	SALES	CENTERED_3_DAY_AVG
1	20-DEC-99 00:00:00	134,337	106,676
2	21-DEC-99 00:00:00	79,015	102,539
3	22-DEC-99 00:00:00	94,264	85,342
4	23-DEC-99 00:00:00	82,746	93,322
5	24-DEC-99 00:00:00	102,957	82,937
6	25-DEC-99 00:00:00	63,107	87,062
7	26-DEC-99 00:00:00	95,123	79,115

Varying Window Size for Each Row

```
SELECT T.TIME_ID, T.DAY_NAME, TO_CHAR (SUM(AMOUNT_SOLD), '9,999,999,999') AS SALES,
  TO_CHAR(AVG(SUM(AMOUNT_SOLD)) OVER (ORDER BY T.TIME_ID
   RANGE BETWEEN INTERVAL '1' DAY PRECEDING AND
   INTERVAL '1' DAY FOLLOWING), '9,999,999,999') AS CENTERED_3_DAY_AVG,
  TO_CHAR(AVG(SUM(AMOUNT_SOLD)) OVER (ORDER BY T.TIME_ID
   RANGE BETWEEN
    (CASE WHEN T.DAY_NAME = 'Monday' THEN (INTERVAL '2' DAY) ELSE (INTERVAL '1' DAY) END) PRECEDING
   AND
   INTERVAL '1' DAY FOLLOWING), '9,999,999,999') AS CENTERED_3_DAY_AVG
FROM SALES S, TIMES T
WHERE S.TIME_ID=T.TIME_ID AND T.CALENDAR_WEEK_NUMBER IN (50, 51)
  AND CALENDAR_YEAR=1999
GROUP BY T.TIME_ID, T.DAY_NAME
ORDER BY T.TIME_ID;
```

	TIME_ID	DAY_NAME	SALES	CENTERED_3_DAY_AVG	CENTERED_3_DAY_AVG_1
1	13-DEC-99 00:00:00	Monday	48,755	48,374	48,374
2	14-DEC-99 00:00:00	Tuesday	47,992	43,065	43,065
3	15-DEC-99 00:00:00	Wednesday	32,448	51,972	51,972
4	16-DEC-99 00:00:00	Thursday	75,476	83,664	83,664
5	17-DEC-99 00:00:00	Friday	143,069	120,295	120,295
6	18-DEC-99 00:00:00	Saturday	142,341	102,789	102,789
7	19-DEC-99 00:00:00	Sunday	22,959	99,879	99,879
8	20-DEC-99 00:00:00	Monday	134,337	78,770	94,663
9	21-DEC-99 00:00:00	Tuesday	79,015	102,539	102,539
10	22-DEC-99 00:00:00	Wednesday	94,264	85,342	85,342
11	23-DEC-99 00:00:00	Thursday	82,746	93,322	93,322
12	24-DEC-99 00:00:00	Friday	102,957	82,937	82,937
13	25-DEC-99 00:00:00	Saturday	63,107	87,062	87,062
14	26-DEC-99 00:00:00	Sunday	95,123	79,115	79,115

LISTAGG Function

```
SELECT CALENDAR_QUARTER_DESC, Q_SALES,
  CASE WHEN SUBSTR(CALENDAR_QUARTER_DESC,-1,1)='1' THEN 'N/A'
    ELSE TO_CHAR(PREV_Q, '9,999,999,990.99')
  END AS PREV_Q,
  CASE WHEN SUBSTR(CALENDAR_QUARTER_DESC,-1,1)='1' THEN 'N/A'
    ELSE TO_CHAR(Q_SALES - PREV_Q, '9,999,999,990.99')
  END as DELTA_Q,
  CASE WHEN SUBSTR(CALENDAR_QUARTER_DESC,-1,1)='1' THEN 'N/A'
    ELSE TO_CHAR((Q_SALES - PREV_Q) / PREV_Q * 100, '9,999,999,990.99') || '%'
  END as DELTA_Q_PRC
FROM (
  SELECT T.CALENDAR_QUARTER_DESC,
    SUM(AMOUNT_SOLD) AS Q_SALES,
    FIRST_VALUE(SUM(AMOUNT_SOLD))
      OVER (ORDER BY T.CALENDAR_QUARTER_DESC
            ROWS BETWEEN 1 PRECEDING AND CURRENT ROW) AS PREV_Q
    FROM SALES S, TIMES T, CUSTOMERS C
    WHERE S.TIME_ID=T.TIME_ID AND S.CUST_ID=C.CUST_ID AND T.CALENDAR_YEAR=2000
      AND C.CUST_ID IN (2595, 9646, 11111)
    GROUP BY T.CALENDAR_QUARTER_DESC
)
ORDER BY 1;
```

	CALENDAR_QUARTER_DESC	Q_SALES	PREV_Q	DELTA_Q	DELTA_Q_PRC
1	2000-01	2040.19	N/A	N/A	N/A
2	2000-02	443.79	2,040.19	-1,596.40	-78.25%
3	2000-03	1097.49	443.79	653.70	147.30%
4	2000-04	6780.95	1,097.49	5,683.46	517.86%

LISTAGG Function

```
SELECT T.CALENDAR_QUARTER_DESC,
  LISTAGG(C.CUST_ID, ',') WITHIN GROUP (ORDER BY C.CUST_ID) as LIST_AGG,
  SUM(AMOUNT_SOLD) AS Q_SALES,
  FIRST_VALUE(SUM(AMOUNT_SOLD))
    OVER (ORDER BY T.CALENDAR_QUARTER_DESC
          ROWS BETWEEN 1 PRECEDING AND CURRENT ROW) AS PREV_Q
FROM SALES S, TIMES T, CUSTOMERS C
WHERE S.TIME_ID=T.TIME_ID AND S.CUST_ID=C.CUST_ID AND T.CALENDAR_YEAR=2000
  AND C.CUST_ID IN (2595, 9646, 11111)
GROUP BY T.CALENDAR_QUARTER_DESC
ORDER BY T.CALENDAR_QUARTER_DESC;
```

	CALENDAR_QUARTER_DESC	LIST_AGG	Q_SALES	PREV_Q
1	2000-01	2595,2595,2595,2595,2595,2595,2595,2595,2595,2595,2595,2595,2595,2595,2595,2595,2595,2595,2595,2595,	2040.19	2040.19
2	2000-02	2595,2595,2595,2595,2595,2595,2595,9646,9646,9646,9646,9646,11111,11111,11111,11111	443.79	2040.19
3	2000-03	2595,2595,2595,2595,2595,2595,2595,2595,9646,9646,9646,9646,9646,9646,11111,11111,11111,11111,11111,	1097.49	443.79
4	2000-04	2595,2595,2595,2595,2595,2595,2595,2595,2595,2595,2595,2595,2595,2595,2595,2595,2595,2595,2595,2595,	6780.95	1097.49

NTH_VALUE Function

```
SELECT C.CUST_ID, T.CALENDAR_QUARTER_DESC,
  SUM(AMOUNT_SOLD) AS Q_SALES,
  FIRST_VALUE(SUM(AMOUNT_SOLD))
    OVER (PARTITION BY C.CUST_ID ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
          ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS Q1,
  NTH_VALUE(SUM(AMOUNT_SOLD), 2)
    OVER (PARTITION BY C.CUST_ID ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
          ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS Q2,
  NTH_VALUE(SUM(AMOUNT_SOLD), 3)
    OVER (PARTITION BY C.CUST_ID ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
          ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS Q3,
  LAST_VALUE(SUM(AMOUNT_SOLD))
    OVER (PARTITION BY C.CUST_ID ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
          ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS Q4
FROM SALES S, TIMES T, CUSTOMERS C
WHERE S.TIME_ID=T.TIME_ID AND S.CUST_ID=C.CUST_ID AND T.CALENDAR_YEAR=2000
  AND C.CUST_ID IN (2595, 9646, 11111)
GROUP BY C.CUST_ID, T.CALENDAR_QUARTER_DESC, T.CALENDAR_QUARTER_NUMBER
ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC;
```

	CUST_ID	CALENDAR_QUARTER_DESC	Q_SALES	Q1	Q2	Q3	Q4
1	2595	2000-01	659.92	659.92	224.79	313.9	6015.08
2	2595	2000-02	224.79	659.92	224.79	313.9	6015.08
3	2595	2000-03	313.9	659.92	224.79	313.9	6015.08
4	2595	2000-04	6015.08	659.92	224.79	313.9	6015.08
5	9646	2000-01	1337.09	1337.09	185.67	203.86	458.29
6	9646	2000-02	185.67	1337.09	185.67	203.86	458.29
7	9646	2000-03	203.86	1337.09	185.67	203.86	458.29
8	9646	2000-04	458.29	1337.09	185.67	203.86	458.29
9	11111	2000-01	43.18	43.18	33.33	579.73	307.58
10	11111	2000-02	33.33	43.18	33.33	579.73	307.58
11	11111	2000-03	579.73	43.18	33.33	579.73	307.58
12	11111	2000-04	307.58	43.18	33.33	579.73	307.58

NTH_VALUE Function

```
SELECT CUST_ID, MAX(Q1) as Q1, MAX(Q2) as Q2, MAX(Q3) as Q3, MAX(Q4) as Q4
FROM (
  SELECT C.CUST_ID CUST_ID, T.CALENDAR_QUARTER_DESC,
    SUM(AMOUNT_SOLD) AS Q_SALES,
    FIRST_VALUE(SUM(AMOUNT_SOLD))
      OVER (PARTITION BY C.CUST_ID ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
            ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS Q1,
    NTH_VALUE(SUM(AMOUNT_SOLD), 2)
      OVER (PARTITION BY C.CUST_ID ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
            ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS Q2,
    NTH_VALUE(SUM(AMOUNT_SOLD), 3)
      OVER (PARTITION BY C.CUST_ID ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
            ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS Q3,
    LAST_VALUE(SUM(AMOUNT_SOLD))
      OVER (PARTITION BY C.CUST_ID ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
            ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS Q4
  FROM SALES S, TIMES T, CUSTOMERS C
  WHERE S.TIME_ID=T.TIME_ID AND S.CUST_ID=C.CUST_ID AND T.CALENDAR_YEAR=2000
    AND C.CUST_ID IN (2595, 9646, 11111)
  GROUP BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
)
GROUP BY CUST_ID
ORDER BY 1;
```

	CUST_ID	Q1	Q2	Q3	Q4
1	2595	659.92	224.79	313.9	6015.08
2	9646	1337.09	185.67	203.86	458.29
3	11111	43.18	33.33	579.73	307.58

NTH_VALUE Function

```
SELECT C.CUST_ID,
  SUM(AMOUNT_SOLD) AS Q_SALES,
  FIRST_VALUE(SUM(AMOUNT_SOLD))
    OVER (PARTITION BY C.CUST_ID ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
          ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS Q1,
  NTH_VALUE(SUM(AMOUNT_SOLD), 2)
    OVER (PARTITION BY C.CUST_ID ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
          ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS Q2,
  NTH_VALUE(SUM(AMOUNT_SOLD), 3)
    OVER (PARTITION BY C.CUST_ID ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
          ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS Q3,
  LAST_VALUE(SUM(AMOUNT_SOLD))
    OVER (PARTITION BY C.CUST_ID ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
          ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS Q4
FROM SALES S, TIMES T, CUSTOMERS C
WHERE S.TIME_ID=T.TIME_ID AND S.CUST_ID=C.CUST_ID AND T.CALENDAR_YEAR=2000
  AND C.CUST_ID IN (2595, 9646, 11111)
GROUP BY C.CUST_ID, T.CALENDAR_QUARTER_DESC
HAVING T.CALENDAR_QUARTER_DESC = '2000-01'
ORDER BY C.CUST_ID, T.CALENDAR_QUARTER_DESC;
```

	CUST_ID	Q_SALES	Q1	Q2	Q3	Q4
1	2595	659.92	659.92	(null)	(null)	659.92
2	9646	1337.09	1337.09	(null)	(null)	1337.09
3	11111	43.18	43.18	(null)	(null)	43.18

MTN.BI.03

SQL FOR ANALYSIS

Questions and Answers

Windowing Aggregate Functions

Author: Aliaksei Belablotski
Senior Software Engineer
Aliaksei_Belablotski@epam.com