**Teradata Ordered Analytical Functions**

Ordered analytical functions include ANSI SQL: 2008 compliant window functions, as well as Teradata SQL-specific functions.

**Ordered Analytical Functions**

Ordered analytical functions provide support for many common operations in analytical processing and data mining environments that require an ordered set of results rows or depend on values in a previous row.

For example, computing a seven-day running sum requires:

* First, those rows are ordered by date.
* Then, the value for the running sum must be computed,
* Adding the current row value to the value of the sum from the previous row, and
* Subtracting the value from the row eight days ago.

**Ordered Analytical Calculations at the SQL Level**

Performing ordered analytical computations at the SQL level rather than through a higher level OLAP calculation engine provides four distinct advantages.

* Reduced programming effort.
* Elimination of the need for external sort routines.
* Elimination of the need to export large data sets to external tools.
* Marked enhancement of analysis performance over the slow, single-threaded operations that external tools perform on large data sets.

**Teradata Warehouse Miner**

You need not directly code SQL queries to take advantage of ordered analytical functions. Both Teradata Database and many third-party query management and analytical tools have full access to the Teradata SQL ordered analytical functions. Teradata Warehouse Miner, for example, a tool that performs data mining preprocessing inside the database engine, relies on these features to perform functions in the database itself rather than requiring data extraction. Teradata Warehouse Miner includes approximately 40 predefined data mining functions in SQL based on the Teradata SQL-specific functions. For example, the Teradata Warehouse Miner FREQ functions uses the Teradata SQL-specific functions CSUM, RANK, and QUALIFY to determine frequencies.

**Syntax Alternatives for Ordered Analytical Functions**

**Teradata SQL supports two syntax alternatives for ordered analytical functions:**

• ANSI SQL: 2008 compliant

• Teradata

Window aggregate, rank, distribution, and row number functions are ANSI SQL: 2008 compliant, while Teradata-specific functions are not.

**The use of the Teradata-specific functions listed in the following table is strongly discouraged. These functions are retained only for backward compatibility with existing applications. Be sure to use the ANSI-compliant window functions for any new applications you develop.**

**Relationship between Teradata-Specific Functions and Window Functions**

The following table identifies equivalent ANSI SQL: 2008 window functions for Teradata specific functions:

**Teradata-Specific Functions Equivalent ANSI SQL: 2008 Window Functions**



**What is a Window functions?**

Analytical/Window functions in Teradata: A Analytical /window function is similar to an aggregate function, though **it is applied to a partition of a result set in compared to the entire table.** A window is specified by the OVER phrase, which can include the following phrases: PARTITION BY, ORDER BY, and ROWS {BETWEEN}.

**Window Feature**

The ANSI SQL: 2008 window feature provides a way to dynamically define a subset of data, or *window*, in an ordered relational database table. **A window is specified by the OVER () phrase, which can include the following clauses inside the parentheses:**

* PARTITION BY
* ORDER BY
* RESET WHEN
* ROWS

The window feature can be applied to the following functions:

* AVG
* CORR
* COUNT
* COVAR\_POP
* COVAR\_SAMP
* MAX
* MIN
* PERCENT\_RANK
* RANK
* REGR\_AVGX
* REGR\_AVGY
* REGR\_COUNT
* REGR\_INTERCEPT
* REGR\_R2
* REGR\_SLOPE
* REGR\_SXX
* REGR\_SXY
* REGR\_SYY
* ROW\_NUMBER
* STDDEV\_POP
* STDDEV\_SAMP
* SUM
* VAR\_POP
* VAR\_SAMP

**PARTITION BY Phrase**

PARTITION BY takes a column reference list and groups the rows based on the specified column reference list over which the ordered analytical function executes. Such a grouping is static.

To define a group or partition based on a condition, use the RESET WHEN phrase.

If there is no PARTITION BY phrase or RESET WHEN phrase, then the entire result set, delivered by the FROM clause, constitutes a single partition, over which the ordered analytical function executes.

Consider the following table named sales.

DROP TABLE sales;

CREATE TABLE sales (

StoreID INTEGER,

SMonth SMALLINT,

ProdID VARCHAR (3),

Sales INTEGER)

NO PRIMARY INDEX;

INSERT INTO sales VALUES (1001, 1, 'C', 35000.00);

INSERT INTO sales VALUES (1001, 2, 'C', 25000.00);

INSERT INTO sales VALUES (1001, 3, 'C', 40000.00);

INSERT INTO sales VALUES (1001, 4, 'C', 25000.00);

INSERT INTO sales VALUES (1001, 5, 'C', 30000.00);

INSERT INTO sales VALUES (1001, 6, 'C', 30000.00);

INSERT INTO sales VALUES (1002, 1, 'M', 50000.00);

INSERT INTO sales VALUES (1002, 2, 'M', 65000.00);

INSERT INTO sales VALUES (1002, 3, 'M', 310000.00);

INSERT INTO sales VALUES (1002, 4, 'M', 20000.00);

INSERT INTO sales VALUES (1002, 5, 'M', 35000.00);

INSERT INTO sales VALUES (1002, 6, 'M', 200000.00);

INSERT INTO sales VALUES (1003, 1, 'I', 60000.00);

INSERT INTO sales VALUES (1003, 2, 'I', 45000.00);

INSERT INTO sales VALUES (1003, 3, 'I', 310000.00);

INSERT INTO sales VALUES (1003, 4, 'I', 60000.00);

INSERT INTO sales VALUES (1003, 5, 'I', 75000.00);

INSERT INTO sales VALUES (1003, 6, 'I', 800000.00);

INSERT INTO sales VALUES (1004, 1, 'A', 30000.00);

INSERT INTO sales VALUES (1004, 2, 'A', 45000.00);

INSERT INTO sales VALUES (1004, 3, 'A', 610000.00);

INSERT INTO sales VALUES (1004, 4, 'A', 60000.00);

INSERT INTO sales VALUES (1004, 5, 'A', 75000.00);

INSERT INTO sales VALUES (1004, 6, 'A', 800000.00);

The following SELECT statement, which does not include PARTITION BY, computes the average sales for all the stores in the table:

SELECT StoreID, SMonth, ProdID, Sales, AVG(Sales) OVER ()FROM sales;

\*\*\* Query completed. 24 rows found. 5 columns returned.

\*\*\* Total elapsed time was 1 second.

StoreID SMonth ProdID Sales Group Avg(Sales)

----------- ------ ------ ----------- ----------------

1001 1 C 35000 159792

1001 2 C 25000 159792

1001 3 C 40000 159792

1001 4 C 25000 159792

1001 5 C 30000 159792

1001 6 C 30000 159792

1002 1 M 50000 159792

1002 2 M 65000 159792

1002 3 M 310000 159792

1002 4 M 20000 159792

1002 5 M 35000 159792

1002 6 M 200000 159792

1003 1 I 60000 159792

1003 2 I 45000 159792

1003 3 I 310000 159792

1003 4 I 60000 159792

1003 5 I 75000 159792

1003 6 I 800000 159792

1004 1 A 30000 159792

1004 2 A 45000 159792

1004 3 A 610000 159792

1004 4 A 60000 159792

1004 5 A 75000 159792

1004 6 A 800000 159792

BTEQ -- Enter your DBC/SQL request or BTEQ command:

To compute the average sales for each store, partition the data in sales table by StoreID:

SELECT StoreID, SMonth, ProdID, Sales, AVG(Sales) OVER (PARTITION BY StoreID)

FROM sales;

SELECT StoreID, SMonth, ProdID, Sales, AVG(Sales) OVER (PARTITION BY StoreID)

FROM sales;

\*\*\* Query completed. 24 rows found. 5 columns returned.

\*\*\* Total elapsed time was 1 second.

StoreID SMonth ProdID Sales Group Avg(Sales)

----------- ------ ------ ----------- ----------------

1001 3 C 40000 30833

1001 5 C 30000 30833

1001 6 C 30000 30833

1001 2 C 25000 30833

1001 4 C 25000 30833

1001 1 C 35000 30833

1002 6 M 200000 113333

1002 1 M 50000 113333

1002 2 M 65000 113333

1002 4 M 20000 113333

1002 3 M 310000 113333

1002 5 M 35000 113333

1003 4 I 60000 225000

1003 3 I 310000 225000

1003 2 I 45000 225000

1003 6 I 800000 225000

1003 5 I 75000 225000

1003 1 I 60000 225000

1004 1 A 30000 270000

1004 6 A 800000 270000

1004 5 A 75000 270000

1004 4 A 60000 270000

1004 3 A 610000 270000

1004 2 A 45000 270000

BTEQ -- Enter your DBC/SQL request or BTEQ command:

select StoreID, AVG(sales) FROM sales GROUP BY StoreID ORDER BY 1;

select StoreID, AVG(sales) FROM sales GROUP BY StoreID ORDER BY 1;

\*\*\* Query completed. 4 rows found. 2 columns returned.

\*\*\* Total elapsed time was 1 second.

StoreID Average(Sales)

----------- --------------

1001 30833

1002 113333

1003 225000

1004 270000

BTEQ -- Enter your DBC/SQL request or BTEQ command:

**ORDER BY Phrase**

ORDER BY specifies how the rows are ordered in a partition, which determines the sort order of the rows over which the function is applied.

SELECT StoreID, SMonth, ProdID, Sales, AVG(Sales) OVER (PARTITION BY StoreID ORDER BY SMonth) FROM sales;

\*\*\* Query completed. 24 rows found. 5 columns returned.

\*\*\* Total elapsed time was 1 second.

StoreID SMonth ProdID Sales Group Avg(Sales)

----------- ------ ------ ----------- ----------------

1001 1 C 35000 30833

1001 2 C 25000 30833

1001 3 C 40000 30833

1001 4 C 25000 30833

1001 5 C 30000 30833

1001 6 C 30000 30833

1002 1 M 50000 113333

1002 2 M 65000 113333

1002 3 M 310000 113333

1002 4 M 20000 113333

1002 5 M 35000 113333

1002 6 M 200000 113333

1003 1 I 60000 225000

1003 2 I 45000 225000

1003 3 I 310000 225000

1003 4 I 60000 225000

1003 5 I 75000 225000

1003 6 I 800000 225000

1004 1 A 30000 270000

1004 2 A 45000 270000

1004 3 A 610000 270000

1004 4 A 60000 270000

1004 5 A 75000 270000

1004 6 A 800000 270000

BTEQ -- Enter your DBC/SQL request or BTEQ command:

To add the monthly sales for a store in the sales table to the sales for previous months, compute the cumulative sales and order the rows in each partition by SMonth:

UNBOUNDED PRECEDING, all rows before the current row -> fixed

UNBOUNDED FOLLOWING, all rows after the current row -> fixed

SELECT StoreID, SMonth, ProdID, Sales, SUM(Sales) OVER (PARTITION BY StoreID ORDER BY SMonth ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)

FROM sales;

\*\*\* Query completed. 24 rows found. 5 columns returned.

\*\*\* Total elapsed time was 1 second.

StoreID SMonth ProdID Sales Cumulative Sum(Sales)

----------- ------ ------ ----------- ---------------------

1001 1 C 35000 35000

1001 2 C 25000 60000

1001 3 C 40000 100000

1001 4 C 25000 125000

1001 5 C 30000 155000

1001 6 C 30000 185000

1002 1 M 50000 50000

1002 2 M 65000 115000

1002 3 M 310000 425000

1002 4 M 20000 445000

1002 5 M 35000 480000

1002 6 M 200000 680000

1003 1 I 60000 60000

1003 2 I 45000 105000

1003 3 I 310000 415000

1003 4 I 60000 475000

1003 5 I 75000 550000

1003 6 I 800000 1350000

1004 1 A 30000 30000

1004 2 A 45000 75000

1004 3 A 610000 685000

1004 4 A 60000 745000

1004 5 A 75000 820000

1004 6 A 800000 1620000

BTEQ -- Enter your DBC/SQL request or BTEQ command:

OR

SELECT StoreID, SMonth, ProdID, Sales, SUM(sales) OVER (PARTITION BY StoreID ORDER BY SMonth ROWS UNBOUNDED PRECEDING)

FROM sales;

SELECT StoreID, SMonth, ProdID, Sales, SUM(Sales) OVER (PARTITION BY StoreID ORDER BY SMonth ROWS BETWEEN 1 PRECEDING AND CURRENT ROW)

FROM sales;

\*\*\* Query completed. 24 rows found. 5 columns returned.

\*\*\* Total elapsed time was 1 second.

StoreID SMonth ProdID Sales Moving Sum(Sales)

----------- ------ ------ ----------- -----------------

1001 1 C 35000 35000

1001 2 C 25000 60000

1001 3 C 40000 65000

1001 4 C 25000 65000

1001 5 C 30000 55000

1001 6 C 30000 60000

1002 1 M 50000 50000

1002 2 M 65000 115000

1002 3 M 310000 375000

1002 4 M 20000 330000

1002 5 M 35000 55000

1002 6 M 200000 235000

1003 1 I 60000 60000

1003 2 I 45000 105000

1003 3 I 310000 355000

1003 4 I 60000 370000

1003 5 I 75000 135000

1003 6 I 800000 875000

1004 1 A 30000 30000

1004 2 A 45000 75000

1004 3 A 610000 655000

1004 4 A 60000 670000

1004 5 A 75000 135000

1004 6 A 800000 875000

BTEQ -- Enter your DBC/SQL request or BTEQ command:

SELECT StoreID, SMonth, ProdID, Sales, SUM(Sales) OVER (PARTITION BY StoreID ORDER BY SMonth ROWS BETWEEN 1 PRECEDING AND 1 PRECEDING)

FROM sales;

\*\*\* Query completed. 24 rows found. 5 columns returned.

\*\*\* Total elapsed time was 1 second.

StoreID SMonth ProdID Sales Moving Sum(Sales)

----------- ------ ------ ----------- -----------------

1001 1 C 35000 ?

1001 2 C 25000 35000

1001 3 C 40000 25000

1001 4 C 25000 40000

1001 5 C 30000 25000

1001 6 C 30000 30000

1002 1 M 50000 ?

1002 2 M 65000 50000

1002 3 M 310000 65000

1002 4 M 20000 310000

1002 5 M 35000 20000

1002 6 M 200000 35000

1003 1 I 60000 ?

1003 2 I 45000 60000

1003 3 I 310000 45000

1003 4 I 60000 310000

1003 5 I 75000 60000

1003 6 I 800000 75000

1004 1 A 30000 ?

1004 2 A 45000 30000

1004 3 A 610000 45000

1004 4 A 60000 610000

1004 5 A 75000 60000

1004 6 A 800000 75000

BTEQ -- Enter your DBC/SQL request or BTEQ command:

**Another Example.**

DROP TABLE test;

CREATE TABLE test (A DECIMAL(5,2), B DECIMAL(5,2));

INSERT INTO test VALUES(1, 1);

INSERT INTO test VALUES(2, 5);

INSERT INTO test VALUES(3, 3);

INSERT INTO test VALUES(4, 5);

INSERT INTO test VALUES(5, 4);

INSERT INTO test VALUES(6, 11);

SELECT \* FROM test;

\*\*\* Query completed. 6 rows found. 2 columns returned.

\*\*\* Total elapsed time was 1 second.

A B

------- -------

2.00 5.00

1.00 1.00

4.00 5.00

3.00 3.00

6.00 11.00

5.00 4.00

As you can see, each average is calculated "over" an ordered frame consisting of the range of rows between the previous row (1 preceding) and the subsequent row (1 following).

SELECT a, b, avg(b) OVER (ORDER BY a ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)

FROM test

ORDER BY a;

\*\*\* Query completed. 6 rows found. 3 columns returned.

\*\*\* Total elapsed time was 1 second.

A B Moving Avg(B)

------- ------- -------------

1.00 1.00 3.00

2.00 5.00 3.00

3.00 3.00 4.33

4.00 5.00 4.00

5.00 4.00 6.67

6.00 11.00 7.50

BTEQ -- Enter your DBC/SQL request or BTEQ command:

SELECT a, b, avg(b) OVER (ORDER BY a ROWS BETWEEN 1 PRECEDING AND 1 PRECEDING)

FROM test

ORDER BY a;

\*\*\* Query completed. 6 rows found. 3 columns returned.

\*\*\* Total elapsed time was 1 second.

A B Moving Avg(B)

------- ------- -------------

1.00 1.00 ?

2.00 5.00 1.00

3.00 3.00 5.00

4.00 5.00 3.00

5.00 4.00 5.00

6.00 11.00 4.00

SELECT a, b, avg(b) OVER (ORDER BY a ROWS BETWEEN 2 PRECEDING AND 1 PRECEDING)

FROM test

ORDER BY a;

\*\*\* Query completed. 6 rows found. 3 columns returned.

\*\*\* Total elapsed time was 1 second.

A B Moving Avg(B)

------- ------- -------------

1.00 1.00 ?

2.00 5.00 1.00

3.00 3.00 3.00

4.00 5.00 4.00

5.00 4.00 4.00

6.00 11.00 4.50

# ROWS UNBOUNDED PRECEDING / UNBOUNDED FOLLOWING

Starting and ending rows might be fixed or relative to the current row based on the following keywords:

UNBOUNDED PRECEDING, all rows before the current row -> fixed

UNBOUNDED FOLLOWING, all rows after the current row -> fixed

x PRECEDING, x rows before the current row -> relative

y FOLLOWING, y rows after the current row -> relative

SELECT a, b, sum(b) OVER (ORDER BY a ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)

FROM test

ORDER BY a;

\*\*\* Query completed. 6 rows found. 3 columns returned.

\*\*\* Total elapsed time was 1 second.

A B Cumulative Sum(B)

------- ------- -----------------

1.00 1.00 1.00

2.00 5.00 6.00

3.00 3.00 9.00

4.00 5.00 14.00

5.00 4.00 18.00

6.00 11.00 29.00

BTEQ -- Enter your DBC/SQL request or BTEQ command:

**OR**

SELECT a, b, sum(b) OVER (ORDER BY a ROWS UNBOUNDED PRECEDING)

FROM test

ORDER BY a;

\*\*\* Query completed. 6 rows found. 3 columns returned.

\*\*\* Total elapsed time was 1 second.

A B Cumulative Sum(B)

------- ------- -----------------

1.00 1.00 1.00

2.00 5.00 6.00

3.00 3.00 9.00

4.00 5.00 14.00

5.00 4.00 18.00

6.00 11.00 29.00

**UNBOUNDED FOLLOWING, all rows after the current row -> fixed**

BTEQ -- Enter your DBC/SQL request or BTEQ command:

SELECT a, b, sum(b) OVER (ORDER BY a ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING)

FROM test

ORDER BY a;

\*\*\* Query completed. 6 rows found. 3 columns returned.

\*\*\* Total elapsed time was 1 second.

A B Remaining Sum(B)

------- ------- ----------------

1.00 1.00 29.00

2.00 5.00 28.00

3.00 3.00 23.00

4.00 5.00 20.00

5.00 4.00 15.00

6.00 11.00 11.00

BTEQ -- Enter your DBC/SQL request or BTEQ command:

SELECT a, b, sum(b) OVER (ORDER BY a ROWS BETWEEN 1 FOLLOWING AND UNBOUNDED

FOLLOWING)

FROM test

ORDER BY a;

\*\*\* Query completed. 6 rows found. 3 columns returned.

\*\*\* Total elapsed time was 1 second.

A B Remaining Sum(B)

------- ------- ----------------

1.00 1.00 28.00

2.00 5.00 23.00

3.00 3.00 20.00

4.00 5.00 15.00

5.00 4.00 11.00

6.00 11.00 ?

ROWS can be specified with the ANSI SQL:2008 compliant window aggregate functions:

* + AVG
  + CORR
  + COUNT
  + COVAR\_POP
  + COVAR\_SAMP
  + MAX
  + MIN
  + REGR\_AVGX
  + REGR\_AVGY
  + REGR\_COUNT
  + REGR\_INTERCEPT
  + REGR\_R2
  + REGR\_SLOPE
  + REGR\_SXX
  + REGR\_SXY
  + REGR\_SYY
  + STDDEV\_POP
  + STDDEV\_SAMP
  + SUM
  + VAR\_POP
  + VAR\_SAMP

SELECT StoreID, SMonth, ProdID, Sales, SUM(Sales) OVER (PARTITION BY StoreID

ORDER BY SMonth ROWS BETWEEN 2 PRECEDING AND CURRENT ROW)

FROM sales;

\*\*\* Query completed. 24 rows found. 5 columns returned.

\*\*\* Total elapsed time was 1 second.

StoreID SMonth ProdID Sales Moving Sum(Sales)

----------- ------ ------ ----------- -----------------

1001 1 C 35000 35000

1001 2 C 25000 60000

1001 3 C 40000 100000

1001 4 C 25000 90000

1001 5 C 30000 95000

1001 6 C 30000 85000

1002 1 M 50000 50000

1002 2 M 65000 115000

1002 3 M 310000 425000

1002 4 M 20000 395000

1002 5 M 35000 365000

1002 6 M 200000 255000

1003 1 I 60000 60000

1003 2 I 45000 105000

1003 3 I 310000 415000

1003 4 I 60000 415000

1003 5 I 75000 445000

1003 6 I 800000 935000

1004 1 A 30000 30000

1004 2 A 45000 75000

1004 3 A 610000 685000

1004 4 A 60000 715000

1004 5 A 75000 745000

1004 6 A 800000 935000

BTEQ -- Enter your DBC/SQL request or BTEQ command:

**RESET WHEN Phrase**

RESET WHEN is a Teradata extension to the ANSI SQL standard.

Depending on the evaluation of the specified condition, RESET WHEN determines the group or partition, over which the ordered analytical function operates. If the condition evaluates to TRUE, a new dynamic partition is created inside the specified window partition.

If there is no RESET WHEN phrase or PARTITION BY phrase, then the entire result set, delivered by the FROM clause, constitutes a single partition, over which the ordered analytical function executes.

You can have different RESET WHEN clauses in the same SELECT list.

**Note:** A window specification that specifies a RESET WHEN clause must also specify an ORDER BY clause.

SELECT StoreID, SMonth, ProdID, Sales,

SUM(sales) OVER (PARTITION BY StoreID ORDER BY SMonth

RESET WHEN sales < /\* preceding row sum \*/ SUM(sales) OVER (PARTITION BY StoreID ORDER BY SMonth ROWS BETWEEN 1 PRECEDING AND 1 PRECEDING)

ROWS UNBOUNDED PRECEDING)

FROM sales;

\*\*\* Query completed. 24 rows found. 5 columns returned.

\*\*\* Total elapsed time was 1 second.

StoreID SMonth ProdID Sales Cumulative Sum(Sales)

----------- ------ ------ ----------- ---------------------

1001 1 C 35000 35000

1001 2 C 25000 25000

1001 3 C 40000 65000

1001 4 C 25000 25000

1001 5 C 30000 55000

1001 6 C 30000 85000

1002 1 M 50000 50000

1002 2 M 65000 115000

1002 3 M 310000 425000

1002 4 M 20000 20000

1002 5 M 35000 55000

1002 6 M 200000 255000

1003 1 I 60000 60000

1003 2 I 45000 45000

1003 3 I 310000 355000

1003 4 I 60000 60000

1003 5 I 75000 135000

1003 6 I 800000 935000

1004 1 A 30000 30000

1004 2 A 45000 75000

1004 3 A 610000 685000

1004 4 A 60000 60000

1004 5 A 75000 135000

1004 6 A 800000 935000

BTEQ -- Enter your DBC/SQL request or BTEQ command:

SELECT StoreID, SMonth, ProdID, Sales,

SUM(sales) OVER (PARTITION BY StoreID ORDER BY SMonth

RESET WHEN sales < /\* preceding row sum \*/ SUM(sales) OVER (PARTITION BY StoreID ORDER BY SMonth ROWS BETWEEN 2 PRECEDING AND 1 PRECEDING)

ROWS UNBOUNDED PRECEDING)

FROM sales;

\*\*\* Query completed. 24 rows found. 5 columns returned.

\*\*\* Total elapsed time was 1 second.

StoreID SMonth ProdID Sales Cumulative Sum(Sales)

----------- ------ ------ ----------- ---------------------

1001 1 C 35000 35000

1001 2 C 25000 25000

1001 3 C 40000 40000

1001 4 C 25000 25000

1001 5 C 30000 30000

1001 6 C 30000 30000

1002 1 M 50000 50000

1002 2 M 65000 115000

1002 3 M 310000 425000

1002 4 M 20000 20000

1002 5 M 35000 35000

1002 6 M 200000 235000

1003 1 I 60000 60000

1003 2 I 45000 45000

1003 3 I 310000 355000

1003 4 I 60000 60000

1003 5 I 75000 75000

1003 6 I 800000 875000

1004 1 A 30000 30000

1004 2 A 45000 75000

1004 3 A 610000 685000

1004 4 A 60000 60000

1004 5 A 75000 75000

1004 6 A 800000 875000

BTEQ -- Enter your SQL request or BTEQ command:

**Multiple Window Specifications (RANK ANSI SQL:2008 standard)**

**NOTE: DENSE\_RANK function is not implemented in Teradata.**

In an SQL statement using more than one window function, each window function can have a unique window specification.

SELECT StoreID, SMonth, ProdID, Sales, SUM(Sales) OVER (PARTITION BY StoreID

ORDER BY SMonth ROWS BETWEEN 2 PRECEDING AND CURRENT ROW),

RANK() OVER (PARTITION BY StoreID ORDER BY Sales DESC)

FROM sales;

\*\*\* Query completed. 24 rows found. 6 columns returned.

\*\*\* Total elapsed time was 1 second.

StoreID SMonth ProdID Sales Moving Sum(Sales) Rank(Sales)

----------- ------ ------ ----------- ----------------- -----------

1001 3 C 40000 100000 1

1001 1 C 35000 35000 2

1001 6 C 30000 85000 3

1001 5 C 30000 95000 3

1001 2 C 25000 60000 5

1001 4 C 25000 90000 5

1002 3 M 310000 425000 1

1002 6 M 200000 255000 2

1002 2 M 65000 115000 3

1002 1 M 50000 50000 4

1002 5 M 35000 365000 5

1002 4 M 20000 395000 6

1003 6 I 800000 935000 1

1003 3 I 310000 415000 2

1003 5 I 75000 445000 3

1003 1 I 60000 60000 4

1003 4 I 60000 415000 4

1003 2 I 45000 105000 6

1004 6 A 800000 935000 1

1004 3 A 610000 685000 2

1004 5 A 75000 745000 3

1004 4 A 60000 715000 4

1004 2 A 45000 75000 5

1004 1 A 30000 30000 6

BTEQ -- Enter your DBC/SQL request or BTEQ command:

SELECT StoreID, SMonth, ProdID, Sales, RANK() OVER (PARTITION BY StoreID ORDER BY Sales )

FROM sales;

\*\*\* Query completed. 24 rows found. 5 columns returned.

\*\*\* Total elapsed time was 1 second.

StoreID SMonth ProdID Sales Rank(Sales ASC)

----------- ------ ------ ----------- ---------------

1001 4 C 25000 1

1001 2 C 25000 1

1001 6 C 30000 3

1001 5 C 30000 3

1001 1 C 35000 5

1001 3 C 40000 6

1002 4 M 20000 1

1002 5 M 35000 2

1002 1 M 50000 3

1002 2 M 65000 4

1002 6 M 200000 5

1002 3 M 310000 6

1003 2 I 45000 1

1003 1 I 60000 2

1003 4 I 60000 2

1003 5 I 75000 4

1003 3 I 310000 5

1003 6 I 800000 6

1004 1 A 30000 1

1004 2 A 45000 2

1004 4 A 60000 3

1004 5 A 75000 4

1004 3 A 610000 5

1004 6 A 800000 6

BTEQ -- Enter your DBC/SQL request or BTEQ command:

**RANK Teradata Function**

The use of Teradata RANK is strongly discouraged. It is a Teradata extension to the ANSI SQL:2008 standard, and is equivalent to the ANSI-compliant RANK window function. Teradata RANK is retained only for backward compatibility with existing applications.

SELECT StoreID,SMonth,ProdID,Sales, RANK(sales asc) from sales;

\*\*\* Query completed. 24 rows found. 5 columns returned.

\*\*\* Total elapsed time was 1 second.

StoreID SMonth ProdID Sales Rank(Sales ASC)

----------- ------ ------ ----------- ---------------

1002 4 M 20000 1

1001 4 C 25000 2

1001 2 C 25000 2

1001 6 C 30000 4

1001 5 C 30000 4

1004 1 A 30000 4

1001 1 C 35000 7

1002 5 M 35000 7

1001 3 C 40000 9

1004 2 A 45000 10

1003 2 I 45000 10

1002 1 M 50000 12

1004 4 A 60000 13

1003 4 I 60000 13

1003 1 I 60000 13

1002 2 M 65000 16

1004 5 A 75000 17

1003 5 I 75000 17

1002 6 M 200000 19

1003 3 I 310000 20

1002 3 M 310000 20

1004 3 A 610000 22

1004 6 A 800000 23

1003 6 I 800000 23

BTEQ -- Enter your DBC/SQL request or BTEQ command:

**NOTE: This will not work on oracle.**

SELECT StoreID,SMonth,ProdID,Sales, RANK() over(order by sales) from sales;

\*\*\* Query completed. 24 rows found. 5 columns returned.

\*\*\* Total elapsed time was 1 second.

StoreID SMonth ProdID Sales Rank(Sales ASC)

----------- ------ ------ ----------- ---------------

1002 4 M 20000 1

1001 2 C 25000 2

1001 4 C 25000 2

1001 5 C 30000 4

1001 6 C 30000 4

1004 1 A 30000 4

1002 5 M 35000 7

1001 1 C 35000 7

1001 3 C 40000 9

1004 2 A 45000 10

1003 2 I 45000 10

1002 1 M 50000 12

1003 4 I 60000 13

1003 1 I 60000 13

1004 4 A 60000 13

1002 2 M 65000 16

1003 5 I 75000 17

1004 5 A 75000 17

1002 6 M 200000 19

1002 3 M 310000 20

1003 3 I 310000 20

1004 3 A 610000 22

1004 6 A 800000 23

1003 6 I 800000 23

BTEQ -- Enter your SQL request or BTEQ command:

**Use of QUALIFY Clause**

Rows can be eliminated by applying conditions on the new column value. The QUALIFY clause is analogous to the HAVING clause of aggregate functions. The QUALIFY clause eliminates rows based on the function value, returning a new value for each of the participating rows.

For example:

SELECT StoreID, SUM(sales) OVER (PARTITION BY StoreID)

FROM sales

QUALIFY SUM(sales) OVER (PARTITION BY StoreID) < 200000;

\*\*\* Query completed. 6 rows found. 2 columns returned.

\*\*\* Total elapsed time was 1 second.

StoreID Group Sum(Sales)

----------- ----------------

1001 185000

1001 185000

1001 185000

1001 185000

1001 185000

1001 185000

BTEQ -- Enter your DBC/SQL request or BTEQ command:

**Computation Sort Order and Result Order**

The sort order that you specify in the window specification defines the sort order of the rows over which the function is applied; it does not define the ordering of the results.

For example, to compute the sum sales for the months following the current month, order the rows by month, default order is ASC:

SELECT StoreID, SMonth, ProdID, Sales, SUM(Sales) OVER (PARTITION BY StoreID ORDER BY SMonth ROWS BETWEEN 1 FOLLOWING AND UNBOUNDED FOLLOWING)

FROM sales;

\*\*\* Query completed. 24 rows found. 5 columns returned.

\*\*\* Total elapsed time was 1 second.

StoreID SMonth ProdID Sales Remaining Sum(Sales)

----------- ------ ------ ----------- --------------------

1001 6 C 30000 ?

1001 5 C 30000 30000

1001 4 C 25000 60000

1001 3 C 40000 85000

1001 2 C 25000 125000

1001 1 C 35000 150000

1002 6 M 200000 ?

1002 5 M 35000 200000

1002 4 M 20000 235000

1002 3 M 310000 255000

1002 2 M 65000 565000

1002 1 M 50000 630000

1003 6 I 800000 ?

1003 5 I 75000 800000

1003 4 I 60000 875000

1003 3 I 310000 935000

1003 2 I 45000 1245000

1003 1 I 60000 1290000

1004 6 A 800000 ?

1004 5 A 75000 800000

1004 4 A 60000 875000

1004 3 A 610000 935000

1004 2 A 45000 1545000

1004 1 A 30000 1590000

BTEQ -- Enter your DBC/SQL request or BTEQ command:

The default sort order is ASC for the computation. However, the results are returned in the reverse order.

SELECT StoreID, SMonth, ProdID, Sales, SUM(Sales) OVER (PARTITION BY StoreID ORDER BY SMonth DESC ROWS BETWEEN 1 FOLLOWING AND UNBOUNDED FOLLOWING)

FROM sales;

\*\*\* Query completed. 24 rows found. 5 columns returned.

\*\*\* Total elapsed time was 1 second.

StoreID SMonth ProdID Sales Remaining Sum(Sales)

----------- ------ ------ ----------- --------------------

1001 1 C 35000 ?

1001 2 C 25000 35000

1001 3 C 40000 60000

1001 4 C 25000 100000

1001 5 C 30000 125000

1001 6 C 30000 155000

1002 1 M 50000 ?

1002 2 M 65000 50000

1002 3 M 310000 115000

1002 4 M 20000 425000

1002 5 M 35000 445000

1002 6 M 200000 480000

1003 1 I 60000 ?

1003 2 I 45000 60000

1003 3 I 310000 105000

1003 4 I 60000 415000

1003 5 I 75000 475000

1003 6 I 800000 550000

1004 1 A 30000 ?

1004 2 A 45000 30000

1004 3 A 610000 75000

1004 4 A 60000 685000

1004 5 A 75000 745000

1004 6 A 800000 820000

BTEQ -- Enter your DBC/SQL request or BTEQ command:

**ROW\_NUMBER**

**Purpose**

Returns the sequential row number, where the first row is number one, of the row within its window partition according to the window ordering of the window.

SELECT ROW\_NUMBER() OVER (ORDER BY StoreID), e.\*

FROM sales e;

\*\*\* Query completed. 24 rows found. 5 columns returned.

\*\*\* Total elapsed time was 1 second.

Row\_Number() StoreID SMonth ProdID Sales

------------ ----------- ------ ------ -----------

1 1001 5 C 30000

2 1001 4 C 25000

3 1001 3 C 40000

4 1001 1 C 35000

5 1001 2 C 25000

6 1001 6 C 30000

7 1002 1 M 50000

8 1002 5 M 35000

9 1002 3 M 310000

10 1002 6 M 200000

11 1002 4 M 20000

12 1002 2 M 65000

13 1003 5 I 75000

14 1003 6 I 800000

15 1003 4 I 60000

16 1003 3 I 310000

17 1003 2 I 45000

18 1003 1 I 60000

19 1004 6 A 800000

20 1004 5 A 75000

21 1004 4 A 60000

22 1004 3 A 610000

23 1004 2 A 45000

24 1004 1 A 30000

BTEQ -- Enter your DBC/SQL request or BTEQ command:

SELECT ROW\_NUMBER() OVER (partition by storeid order by smonth) , StoreID, SMonth, ProdID, Sales, SUM(sales) over(partition by storeid order by smonth ROWS BETWEEN unbounded PRECEDING AND CURRENT ROW)

from sales;

\*\*\* Query completed. 24 rows found. 6 columns returned.

\*\*\* Total elapsed time was 1 second.

Row\_Number() StoreID SMonth ProdID Sales Cumulative Sum(Sales)

------------ ----------- ------ ------ ----------- ---------------------

1 1001 1 C 35000 35000

2 1001 2 C 25000 60000

3 1001 3 C 40000 100000

4 1001 4 C 25000 125000

5 1001 5 C 30000 155000

6 1001 6 C 30000 185000

1 1002 1 M 50000 50000

2 1002 2 M 65000 115000

3 1002 3 M 310000 425000

4 1002 4 M 20000 445000

5 1002 5 M 35000 480000

6 1002 6 M 200000 680000

1 1003 1 I 60000 60000

2 1003 2 I 45000 105000

3 1003 3 I 310000 415000

4 1003 4 I 60000 475000

5 1003 5 I 75000 550000

6 1003 6 I 800000 1350000

1 1004 1 A 30000 30000

2 1004 2 A 45000 75000

3 1004 3 A 610000 685000

4 1004 4 A 60000 745000

5 1004 5 A 75000 820000

6 1004 6 A 800000 1620000

BTEQ -- Enter your DBC/SQL request or BTEQ command: