14.20.3 Window Function Frame Specification

The definition of a window used with a window function can include a frame clause. A frame is a subset of the current partition and the frame clause specifies how to define the subset.

Frames are determined with respect to the current row, which enables a frame to move within a partition depending on the location of the current row within its partition. Examples:

- By defining a frame to be all rows from the partition start to the current row, you can compute running totals for each row.
- By defining a frame as extending ***N*** rows on either side of the current row, you can compute rolling averages.

The following query demonstrates the use of moving frames to compute running totals within each group of time-ordered `level` values, as well as rolling averages computed from the current row and the rows that immediately precede and follow it:

```
mysql> SELECT
         time, subject, val,
         SUM(val) OVER (PARTITION BY subject ORDER BY time
                        ROWS UNBOUNDED PRECEDING)
           AS running_total,
         AVG(val) OVER (PARTITION BY subject ORDER BY time
                        ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)
           AS running_average
       FROM observations;
+----------+---------+------+---------------+-----------------+
| time     | subject | val  | running_total | running_average |
+----------+---------+------+---------------+-----------------+
| 07:00:00 | st113   |   10 |            10 |          9.5000 |
| 07:15:00 | st113   |    9 |            19 |         14.6667 |
| 07:30:00 | st113   |   25 |            44 |         18.0000 |
| 07:45:00 | st113   |   20 |            64 |         22.5000 |
| 07:00:00 | xh458   |    0 |             0 |          5.0000 |
| 07:15:00 | xh458   |   10 |            10 |          5.0000 |
| 07:30:00 | xh458   |    5 |            15 |         15.0000 |
| 07:45:00 | xh458   |   30 |            45 |         20.0000 |
| 08:00:00 | xh458   |   25 |            70 |         27.5000 |
+----------+---------+------+---------------+-----------------+
```

For the `running_average` column, there is no frame row preceding the first one or following the last. In these cases, `AVG()` computes the average of the rows that are available.

Aggregate functions used as window functions operate on rows in the current row frame, as do these nonaggregate window functions:

```
FIRST_VALUE()
LAST_VALUE()
NTH_VALUE()
```

Standard SQL specifies that window functions that operate on the entire partition should have no frame clause. MySQL permits a frame clause for such functions but ignores it. These functions use the entire partition even if a frame is specified:

```
CUME_DIST()
DENSE_RANK()
LAG()
LEAD()
NTILE()
PERCENT_RANK()
RANK()
ROW_NUMBER()
```

The frame clause, if given, has this syntax:

```
frame_clause:
    frame_units frame_extent

frame_units:
    {ROWS | RANGE}
```

In the absence of a frame clause, the default frame depends on whether an `ORDER BY` clause is present, as described later in this section.

The ***`frame_units`*** value indicates the type of relationship between the current row and frame rows:

- `ROWS`: The frame is defined by beginning and ending row positions. Offsets are differences in row numbers from the current row number.
- `RANGE`: The frame is defined by rows within a value range. Offsets are differences in row values from the current row value.

The ***`frame_extent`*** value indicates the start and end points of the frame. You can specify just the start of the frame (in which case the current row is implicitly the end) or use `BETWEEN` to specify both frame endpoints:

```
frame_extent:
    {frame_start | frame_between}

frame_between:
    BETWEEN frame_start AND frame_end

frame_start, frame_end: {
    CURRENT ROW
  | UNBOUNDED PRECEDING
  | UNBOUNDED FOLLOWING
  | expr PRECEDING
  | expr FOLLOWING
}
```

With `BETWEEN` syntax, ***`frame_start`*** must not occur later than ***`frame_end`***.

The permitted ***`frame_start`*** and ***`frame_end`*** values have these meanings:

- `CURRENT ROW`: For `ROWS`, the bound is the current row. For `RANGE`, the bound is the peers of the current row.
- `UNBOUNDED PRECEDING`: The bound is the first partition row.
- `UNBOUNDED FOLLOWING`: The bound is the last partition row.
- ***`expr`*** `PRECEDING`: For `ROWS`, the bound is ***`expr`*** rows before the current row. For `RANGE`, the bound is the rows with values equal to the current row value minus ***`expr`***; if the current row value is `NULL`, the bound is the peers of the row.

  For ***`expr`*** `PRECEDING` (and ***`expr`*** `FOLLOWING`), ***`expr`*** can be a `?` parameter marker (for use in a prepared statement), a nonnegative numeric literal, or a temporal interval of the form `INTERVAL` ***`val unit`***. For `INTERVAL` expressions, ***`val`*** specifies nonnegative interval value, and ***`unit`*** is a keyword indicating the units in which the value should be interpreted. (For details about the permitted ***`units`*** specifiers, see the description of the `DATE_ADD()` function in Section 14.7, “Date and Time Functions”.)

  `RANGE` on a numeric or temporal ***`expr`*** requires `ORDER BY` on a numeric or temporal expression, respectively.

  Examples of valid ***`expr`*** `PRECEDING` and ***`expr`*** `FOLLOWING` indicators:

  ```
  10 PRECEDING
  INTERVAL 5 DAY PRECEDING
  5 FOLLOWING
  INTERVAL '2:30' MINUTE_SECOND FOLLOWING
  ```

- ***expr*** `FOLLOWING`: For `ROWS`, the bound is ***expr*** rows after the current row. For `RANGE`, the bound is the rows with values equal to the current row value plus ***expr***; if the current row value is `NULL`, the bound is the peers of the row.

  For permitted values of ***expr***, see the description of ***expr*** `PRECEDING`.

The following query demonstrates `FIRST_VALUE()`, `LAST_VALUE()`, and two instances of `NTH_VALUE()`:

```
mysql> SELECT
         time, subject, val,
         FIRST_VALUE(val)  OVER w AS 'first',
         LAST_VALUE(val)   OVER w AS 'last',
         NTH_VALUE(val, 2) OVER w AS 'second',
         NTH_VALUE(val, 4) OVER w AS 'fourth'
       FROM observations
       WINDOW w AS (PARTITION BY subject ORDER BY time
                    ROWS UNBOUNDED PRECEDING);
+----------+---------+------+-------+------+--------+--------+
| time     | subject | val  | first | last | second | fourth |
+----------+---------+------+-------+------+--------+--------+
| 07:00:00 | st113   |   10 |    10 |   10 |   NULL |   NULL |
| 07:15:00 | st113   |    9 |    10 |    9 |      9 |   NULL |
| 07:30:00 | st113   |   25 |    10 |   25 |      9 |   NULL |
| 07:45:00 | st113   |   20 |    10 |   20 |      9 |     20 |
| 07:00:00 | xh458   |    0 |     0 |    0 |   NULL |   NULL |
| 07:15:00 | xh458   |   10 |     0 |   10 |     10 |   NULL |
| 07:30:00 | xh458   |    5 |     0 |    5 |     10 |   NULL |
| 07:45:00 | xh458   |   30 |     0 |   30 |     10 |     30 |
| 08:00:00 | xh458   |   25 |     0 |   25 |     10 |     30 |
+----------+---------+------+-------+------+--------+--------+
```

Each function uses the rows in the current frame, which, per the window definition shown, extends from the first partition row to the current row. For the `NTH_VALUE()` calls, the current frame does not always include the requested row; in such cases, the return value is `NULL`.

In the absence of a frame clause, the default frame depends on whether an `ORDER BY` clause is present:

- With `ORDER BY`: The default frame includes rows from the partition start through the current row, including all peers of the current row (rows equal to the current row according to the `ORDER BY` clause). The default is equivalent to this frame specification:

  ```
  RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
  ```

- Without `ORDER BY`: The default frame includes all partition rows (because, without `ORDER BY`, all partition rows are peers). The default is equivalent to this frame specification:

```
RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
```

Because the default frame differs depending on presence or absence of `ORDER BY`, adding `ORDER BY` to a query to get deterministic results may change the results. (For example, the values produced by `SUM()` might change.) To obtain the same results but ordered per `ORDER BY`, provide an explicit frame specification to be used regardless of whether `ORDER BY` is present.

The meaning of a frame specification can be nonobvious when the current row value is `NULL`. Assuming that to be the case, these examples illustrate how various frame specifications apply:

- `ORDER BY X ASC RANGE BETWEEN 10 FOLLOWING AND 15 FOLLOWING`

  The frame starts at `NULL` and stops at `NULL`, thus includes only rows with value `NULL`.

- `ORDER BY X ASC RANGE BETWEEN 10 FOLLOWING AND UNBOUNDED FOLLOWING`

  The frame starts at `NULL` and stops at the end of the partition. Because an `ASC` sort puts `NULL` values first, the frame is the entire partition.

- `ORDER BY X DESC RANGE BETWEEN 10 FOLLOWING AND UNBOUNDED FOLLOWING`

  The frame starts at `NULL` and stops at the end of the partition. Because a `DESC` sort puts `NULL` values last, the frame is only the `NULL` values.

- `ORDER BY X ASC RANGE BETWEEN 10 PRECEDING AND UNBOUNDED FOLLOWING`

  The frame starts at `NULL` and stops at the end of the partition. Because an `ASC` sort puts `NULL` values first, the frame is the entire partition.

- `ORDER BY X ASC RANGE BETWEEN 10 PRECEDING AND 10 FOLLOWING`

  The frame starts at `NULL` and stops at `NULL`, thus includes only rows with value `NULL`.

- `ORDER BY X ASC RANGE BETWEEN 10 PRECEDING AND 1 PRECEDING`

  The frame starts at `NULL` and stops at `NULL`, thus includes only rows with value `NULL`.

- `ORDER BY X ASC RANGE BETWEEN UNBOUNDED PRECEDING AND 10 FOLLOWING`

  The frame starts at the beginning of the partition and stops at rows with value `NULL`. Because an `ASC` sort puts `NULL` values first, the frame is only the `NULL` values.