12.21.2 Window Function Concepts and Syntax

This section describes how to use window functions. Examples use the same sales information data set as found in the discussion of the `GROUPING()` function in Section 12.20.2, “GROUP BY Modifiers”:

```
mysql> SELECT * FROM sales ORDER BY country, year, product;
+------+---------+------------+--------+
| year | country | product    | profit |
+------+---------+------------+--------+
| 2000 | Finland | Computer   |   1500 |
| 2000 | Finland | Phone      |    100 |
| 2001 | Finland | Phone      |     10 |
| 2000 | India   | Calculator |     75 |
| 2000 | India   | Calculator |     75 |
| 2000 | India   | Computer   |   1200 |
| 2000 | USA     | Calculator |     75 |
| 2000 | USA     | Computer   |   1500 |
| 2001 | USA     | Calculator |     50 |
| 2001 | USA     | Computer   |   1500 |
| 2001 | USA     | Computer   |   1200 |
| 2001 | USA     | TV         |    150 |
| 2001 | USA     | TV         |    100 |
+------+---------+------------+--------+
```

A window function performs an aggregate-like operation on a set of query rows. However, whereas an aggregate operation groups query rows into a single result row, a window function produces a result for each query row:

- The row for which function evaluation occurs is called the current row.
- The query rows related to the current row over which function evaluation occurs comprise the window for the current row.

For example, using the sales information table, these two queries perform aggregate operations that produce a single global sum for all rows taken as a group, and sums grouped per country:

```
mysql> SELECT SUM(profit) AS total_profit
       FROM sales;
+--------------+
| total_profit |
+--------------+
```

```
|         7535 |
+--------------+
mysql> SELECT country, SUM(profit) AS country_profit
       FROM sales
       GROUP BY country
       ORDER BY country;
+---------+----------------+
| country | country_profit |
+---------+----------------+
| Finland |           1610 |
| India   |           1350 |
| USA     |           4575 |
+---------+----------------+
```

By contrast, window operations do not collapse groups of query rows to a single output row. Instead, they produce a result for each row. Like the preceding queries, the following query uses `SUM()`, but this time as a window function:

```
mysql> SELECT
         year, country, product, profit,
         SUM(profit) OVER() AS total_profit,
         SUM(profit) OVER(PARTITION BY country) AS country_profit
       FROM sales
       ORDER BY country, year, product, profit;
+------+---------+------------+--------+--------------+----------------+
| year | country | product    | profit | total_profit | country_profit |
+------+---------+------------+--------+--------------+----------------+
| 2000 | Finland | Computer   |   1500 |         7535 |           1610 |
| 2000 | Finland | Phone      |    100 |         7535 |           1610 |
| 2001 | Finland | Phone      |     10 |         7535 |           1610 |
| 2000 | India   | Calculator |     75 |         7535 |           1350 |
| 2000 | India   | Calculator |     75 |         7535 |           1350 |
| 2000 | India   | Computer   |   1200 |         7535 |           1350 |
| 2000 | USA     | Calculator |     75 |         7535 |           4575 |
| 2000 | USA     | Computer   |   1500 |         7535 |           4575 |
| 2001 | USA     | Calculator |     50 |         7535 |           4575 |
| 2001 | USA     | Computer   |   1200 |         7535 |           4575 |
| 2001 | USA     | Computer   |   1500 |         7535 |           4575 |
| 2001 | USA     | TV         |    100 |         7535 |           4575 |
| 2001 | USA     | TV         |    150 |         7535 |           4575 |
+------+---------+------------+--------+--------------+----------------+
```

Each window operation in the query is signified by inclusion of an `OVER` clause that specifies how to partition query rows into groups for processing by the window function:

- The first `OVER` clause is empty, which treats the entire set of query rows as a single partition. The window function thus produces a global sum, but does so for each row.
- The second `OVER` clause partitions rows by country, producing a sum per partition (per country). The function produces this sum for each partition row.

Window functions are permitted only in the select list and `ORDER BY` clause. Query result rows are determined from the `FROM` clause, after `WHERE`, `GROUP BY`, and `HAVING` processing, and windowing execution occurs before `ORDER BY`, `LIMIT`, and `SELECT DISTINCT`.

The `OVER` clause is permitted for many aggregate functions, which therefore can be used as window or nonwindow functions, depending on whether the `OVER` clause is present or absent:

```
AVG()
BIT_AND()
BIT_OR()
BIT_XOR()
COUNT()
JSON_ARRAYAGG()
JSON_OBJECTAGG()
MAX()
MIN()
STDDEV_POP(), STDDEV(), STD()
STDDEV_SAMP()
SUM()
VAR_POP(), VARIANCE()
VAR_SAMP()
```

For details about each aggregate function, see Section 12.20.1, “Aggregate Function Descriptions”.

MySQL also supports nonaggregate functions that are used only as window functions. For these, the `OVER` clause is mandatory:

```
CUME_DIST()
DENSE_RANK()
FIRST_VALUE()
LAG()
LAST_VALUE()
LEAD()
NTH_VALUE()
NTILE()
PERCENT_RANK()
```

```
RANK()
ROW_NUMBER()
```

For details about each nonaggregate function, see Section 12.21.1, “Window Function Descriptions”.

As an example of one of those nonaggregate window functions, this query uses `ROW_NUMBER()`, which produces the row number of each row within its partition. In this case, rows are numbered per country. By default, partition rows are unordered and row numbering is nondeterministic. To sort partition rows, include an `ORDER BY` clause within the window definition. The query uses unordered and ordered partitions (the `row_num1` and `row_num2` columns) to illustrate the difference between omitting and including `ORDER BY`:

```
mysql> SELECT
         year, country, product, profit,
         ROW_NUMBER() OVER(PARTITION BY country) AS row_num1,
         ROW_NUMBER() OVER(PARTITION BY country ORDER BY year, product) AS row_num2
       FROM sales;
+------+---------+------------+--------+----------+----------+
| year | country | product    | profit | row_num1 | row_num2 |
+------+---------+------------+--------+----------+----------+
| 2000 | Finland | Computer   |   1500 |        2 |        1 |
| 2000 | Finland | Phone      |    100 |        1 |        2 |
| 2001 | Finland | Phone      |     10 |        3 |        3 |
| 2000 | India   | Calculator |     75 |        2 |        1 |
| 2000 | India   | Calculator |     75 |        3 |        2 |
| 2000 | India   | Computer   |   1200 |        1 |        3 |
| 2000 | USA     | Calculator |     75 |        5 |        1 |
| 2000 | USA     | Computer   |   1500 |        4 |        2 |
| 2001 | USA     | Calculator |     50 |        2 |        3 |
| 2001 | USA     | Computer   |   1500 |        3 |        4 |
| 2001 | USA     | Computer   |   1200 |        7 |        5 |
| 2001 | USA     | TV         |    150 |        1 |        6 |
| 2001 | USA     | TV         |    100 |        6 |        7 |
+------+---------+------------+--------+----------+----------+
```

As mentioned previously, to use a window function (or treat an aggregate function as a window function), include an `OVER` clause following the function call. The `OVER` clause has two forms:

```
over_clause:
    {OVER (window_spec) | OVER window_name}
```

Both forms define how the window function should process query rows. They differ in whether the window is defined directly in the `OVER` clause, or supplied by a reference to a named window defined elsewhere in the query:

- In the first case, the window specification appears directly in the `OVER` clause, between the parentheses.
- In the second case, ***`window_name`*** is the name for a window specification defined by a `WINDOW` clause elsewhere in the query. For details, see Section 12.21.4, “Named Windows”.

For `OVER (`***`window_spec`***`)` syntax, the window specification has several parts, all optional:

```
window_spec:
    [window_name] [partition_clause] [order_clause] [frame_clause]
```

If `OVER()` is empty, the window consists of all query rows and the window function computes a result using all rows. Otherwise, the clauses present within the parentheses determine which query rows are used to compute the function result and how they are partitioned and ordered:

- ***`window_name`***: The name of a window defined by a `WINDOW` clause elsewhere in the query. If ***`window_name`*** appears by itself within the `OVER` clause, it completely defines the window. If partitioning, ordering, or framing clauses are also given, they modify interpretation of the named window. For details, see Section 12.21.4, “Named Windows”.
- ***`partition_clause`***: A `PARTITION BY` clause indicates how to divide the query rows into groups. The window function result for a given row is based on the rows of the partition that contains the row. If `PARTITION BY` is omitted, there is a single partition consisting of all query rows.

  > **Note**
  >
  > Partitioning for window functions differs from table partitioning. For information about table partitioning, see Chapter 24, *Partitioning*.

  `partition_clause` has this syntax:

  ```
  partition_clause:
      PARTITION BY expr [, expr] ...
  ```

Standard SQL requires `PARTITION BY` to be followed by column names only. A MySQL extension is to permit expressions, not just column names. For example, if a table contains a `TIMESTAMP` column named `ts`, standard SQL permits `PARTITION BY ts` but not `PARTITION BY HOUR(ts)`, whereas MySQL permits both.

- ***`order_clause`***: An `ORDER BY` clause indicates how to sort rows in each partition. Partition rows that are equal according to the `ORDER BY` clause are considered peers. If `ORDER BY` is omitted, partition rows are unordered, with no processing order implied, and all partition rows are peers.

  `order_clause` has this syntax:

  ```
  order_clause:
      ORDER BY expr [ASC|DESC] [, expr [ASC|DESC]] ...
  ```

  Each `ORDER BY` expression optionally can be followed by `ASC` or `DESC` to indicate sort direction. The default is `ASC` if no direction is specified. `NULL` values sort first for ascending sorts, last for descending sorts.

  An `ORDER BY` in a window definition applies within individual partitions. To sort the result set as a whole, include an `ORDER BY` at the query top level.

- ***`frame_clause`***: A frame is a subset of the current partition and the frame clause specifies how to define the subset. The frame clause has many subclauses of its own. For details, see Section 12.21.3, “Window Function Frame Specification”.