LanguageManual WindowingAndAnalytics

Windowing and Analytics Functions

Enhancements to Hive QL

> **Version**
>
> Introduced in Hive version 0.11.

This section introduces the Hive QL enhancements for windowing and analytics functions. See "Windowing Specifications in HQL" (attached to HIVE-4197) for details. HIVE-896 has more information, including links to earlier documentation in the initial comments.

All of the windowing and analytics functions operate as per the SQL standard.

The current release supports the following functions for windowing and analytics:

1. Windowing functions
   - LEAD
     - The number of rows to lead can optionally be specified. If the number of rows to lead is not specified, the lead is one row.
     - Returns null when the lead for the current row extends beyond the end of the window.
   - LAG
     - The number of rows to lag can optionally be specified. If the number of rows to lag is not specified, the lag is one row.
     - Returns null when the lag for the current row extends before the beginning of the window.
   - FIRST_VALUE
     - This takes at most two parameters. The first parameter is the column for which you want the first value, the second (optional) parameter must be a boolean which is `false` by default. If set to true it skips null values.
   - LAST_VALUE
     - This takes at most two parameters. The first parameter is the column for which you want the last value, the second (optional) parameter must be a boolean which is `false` by default. If set to true it skips null values.
2. The OVER clause
   - OVER with standard aggregates:
     - COUNT
     - SUM
     - MIN
     - MAX
     - AVG
   - OVER with a PARTITION BY statement with one or more partitioning columns of any primitive datatype.
   - OVER with PARTITION BY and ORDER BY with one or more partitioning and/or ordering columns of any datatype.
     - OVER with a window specification. Windows can be defined separately in a WINDOW clause. Window specifications support the following formats:

       ```
       (ROWS | RANGE) BETWEEN (UNBOUNDED | [num]) PRECEDING AND ([num] PRECEDING | CURRENT ROW |
       (UNBOUNDED | [num]) FOLLOWING)
       (ROWS | RANGE) BETWEEN CURRENT ROW AND (CURRENT ROW | (UNBOUNDED | [num]) FOLLOWING)
       (ROWS | RANGE) BETWEEN [num] FOLLOWING AND (UNBOUNDED | [num]) FOLLOWING
       ```

       When ORDER BY is specified with missing WINDOW clause, the WINDOW specification defaults to `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW.`

       When both ORDER BY and WINDOW clauses are missing, the WINDOW specification defaults to `ROW BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING.`

> ⚠ The OVER clause supports the following functions, but it does not support a window with them (see HIVE-4797):
>
> Ranking functions: Rank, NTile, DenseRank, CumeDist, PercentRank.
>
> Lead and Lag functions.

3. Analytics functions
   - RANK
   - ROW_NUMBER
   - DENSE_RANK
   - CUME_DIST
   - PERCENT_RANK
   - NTILE
4. Distinct support in Hive 2.1.0 and later (see HIVE-9534)

   Distinct is supported for aggregation functions including SUM, COUNT and AVG, which aggregate over the distinct values within each partition. Current implementation has the limitation that no ORDER BY or window specification can be supported in the partitioning clause for performance reason. The supported syntax is as follows.

   ```
   COUNT(DISTINCT a) OVER (PARTITION BY c)
   ```

   ORDER BY and window specification is supported for distinct in Hive 2.2.0 (see HIVE-13453). An example is as follows.

   ```
   COUNT(DISTINCT a) OVER (PARTITION BY c ORDER BY d ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)
   ```

5. Aggregate functions in OVER clause support in Hive 2.1.0 and later (see HIVE-13475)

   Support to reference aggregate functions within the OVER clause has been added. For instance, currently we can use the SUM aggregation function within the OVER clause as follows.

   ```
   SELECT rank() OVER (ORDER BY sum(b))
   FROM T
   GROUP BY a;
   ```

Examples

This section provides examples of how to use the Hive QL windowing and analytics functions in SELECT statements. See HIVE-896 for additional examples.

PARTITION BY with one partitioning column, no ORDER BY or window specification

```
SELECT a, COUNT(b) OVER (PARTITION BY c)
FROM T;
```

PARTITION BY with two partitioning columns, no ORDER BY or window specification

```
SELECT a, COUNT(b) OVER (PARTITION BY c, d)
FROM T;
```

PARTITION BY with one partitioning column, one ORDER BY column, and no window specification

```
SELECT a, SUM(b) OVER (PARTITION BY c ORDER BY d)
FROM T;
```

PARTITION BY with two partitioning columns, two ORDER BY columns, and no window specification

```
SELECT a, SUM(b) OVER (PARTITION BY c, d ORDER BY e, f)
FROM T;
```

PARTITION BY with partitioning, ORDER BY, and window specification

```
SELECT a, SUM(b) OVER (PARTITION BY c ORDER BY d ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
FROM T;
```

```
SELECT a, AVG(b) OVER (PARTITION BY c ORDER BY d ROWS BETWEEN 3 PRECEDING AND CURRENT ROW)
FROM T;
```

```
SELECT a, AVG(b) OVER (PARTITION BY c ORDER BY d ROWS BETWEEN 3 PRECEDING AND 3 FOLLOWING)
FROM T;
```

```
SELECT a, AVG(b) OVER (PARTITION BY c ORDER BY d ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING)
FROM T;
```

There can be multiple `OVER` clauses in a single query. A single `OVER` clause only applies to the immediately preceding function call. In this example, the first OVER clause applies to COUNT(b) and the second OVER clause applies to SUM(b):

```
SELECT
 a,
 COUNT(b) OVER (PARTITION BY c),
 SUM(b) OVER (PARTITION BY c)
FROM T;
```

Aliases can be used as well, with or without the keyword AS:

```
SELECT
 a,
 COUNT(b) OVER (PARTITION BY c) AS b_count,
 SUM(b) OVER (PARTITION BY c) b_sum
FROM T;
```

WINDOW clause

```
SELECT a, SUM(b) OVER w
FROM T
WINDOW w AS (PARTITION BY c ORDER BY d ROWS UNBOUNDED PRECEDING);
```

LEAD using default 1 row lead and not specifying default value

```
SELECT a, LEAD(a) OVER (PARTITION BY b ORDER BY C)
FROM T;
```

LAG specifying a lag of 3 rows and default value of 0

```
SELECT a, LAG(a, 3, 0) OVER (PARTITION BY b ORDER BY C)
FROM T;
```

Distinct counting for each partition

```
SELECT a, COUNT(distinct a) OVER (PARTITION BY b)
FROM T;
```