More Ranking Functions

WINDOW FUNCTIONS IN SNOWFLAKE

Jake Roach
Field Data Engineer

RANK vs. DENSE_RANK

```
SELECT
    workout_duration,

    RANK() OVER(
        ORDER BY workout_duration DESC
    ) AS r,

    -- Without gaps!
    DENSE_RANK() OVER(
        ORDER BY workout_duration DESC
    ) AS dr

FROM FITNESS.workouts;
```

```
  workout_duration   |   r   |   dr
 ------------------  | ----- | -----
        78           |   1   |   1
        71           |   2   |   2
        71           |   2   |   2

   ***  68           |   4   |   3   ***

        67           |   5   |   4
        67           |   5   |   4
        67           |   5   |   4
        63           |   8   |   5
        61           |   9   |   6
```

NTH_VALUE

```
SELECT
    <field>,
    <another-field>,

    NTH_VALUE(<1>, <n>) OVER(

        PARTITION BY <2>
        ORDER BY <3>

    ) AS <alias>

FROM <SCHEMA>.<table>;
```

`NTH_VALUE` returns the specified value from the "N'th" record in a window, similar to `FIRST/LAST_VALUE`

`<1>` : Value to retrieve from row

`<n>` : Row number to retrieve

`<2>` : *Optional* field to partition by

`<3>` : `ORDER BY` determines the ranking of records

NTH_VALUE

```sql
SELECT
    gym_location,
    workout_duration,

    -- Return the second-longest workout duration for each gym location
    NTH_VALUE(workout_duration, 2) OVER(

        PARTITION BY gym_location
        ORDER BY workout_duration DESC

    ) AS second_longest_workout

FROM FITNESS.workouts;
```

NTH_VALUE

```
  gym_location   |  workout_duration  |  second_longest_workout
---------------- | ------------------ | -------------------------
   New York      |        71          |           68
   New York      |        68          |           68
   New York      |        67          |           68

   Los Angeles   |        78          |           67
   Los Angeles   |        67          |           67
   Los Angeles   |        67          |           67
   Los Angeles   |        63          |           67

   Miami         |        71          |           61
   Miami         |        61          |           61
```

Putting it all together

```
SELECT
    gym_location, workout_duration,

    NTH_VALUE(workout_duration, 2) OVER(
        PARTITION BY gym_location
        ORDER BY workout_duration DESC
    ) AS second_longest_workout,

    RANK() OVER(PARTITION BY gym_location ORDER BY workout_duration DESC) AS r,

    DENSE_RANK() OVER(PARTITION BY gym_location ORDER BY workout_duration DESC) AS dr,

FROM FITNESS.workouts;
```

Putting it all together

```
  gym_location   |  workout_duration  |  second_longest_workout  |  r  |  dr
---------------- | ------------------ | ------------------------- | --- | -----
   New York      |        71          |           68              |  1  |  1
   New York      |        68          |           68              |  2  |  2
   New York      |        67          |           68              |  3  |  3

   Los Angeles   |        78          |           67              |  1  |  1
   Los Angeles   |        67          |           67              |  2  |  2
   Los Angeles   |        67          |           67              |  2  |  2
   Los Angeles   |        63          |           67              |  4  |  3

   Miami         |        71          |           61              |  1  |  1
   Miami         |        61          |           61              |  2  |  2
```

Let's practice!

WINDOW FUNCTIONS IN SNOWFLAKE

NTILE and CUME_DIST

WINDOW FUNCTIONS IN SNOWFLAKE

Jake Roach
Field Data engineer

Creating buckets of rows

How can we classify gym members based on their workouts to market the right classes?

```
  member_id  |  gym_location  |  calories_burned  |  marketing_group
 ----------- | -------------- | ----------------- | -----------------
    m_192    |    Miami       |        45         |        1
    m_74     |    Miami       |        59         |        1
    m_233    |    Portland    |        60         |        1

    m_14     |    Cleveland   |        72         |        2
    m_346    |    Portland    |        77         |        2
    m_289    |    Cleveland   |        81         |        2

    m_565    |    Miami       |       1085        |        50
                               ...
```

NTILE

```
SELECT
    <fields>,
    <2>,
    <1>,

    NTILE(<n>) OVER(
        PARTITION BY <2>
        ORDER BY <1>
    )

...;
```

`NTILE` is used to create "N" number of equally-sized "buckets"

`<n>` : number of buckets

`<1>` : field used to create buckets

`<2>` : field used to evenly-distribute records using `PARTITION BY`

Bucketing fitness data

```sql
SELECT
    member_id,
    gym_location,
    calories_burned,

    -- Create 50 equally-sized buckets of data
    NTILE(50) OVER(
        ORDER BY calories_burned   -- Decides the records in each bucket
    ) AS marketing_group

FROM FITNESS.workouts
ORDER BY marketing_group, calories_burned;  -- ORDER the final result set
```

Bucketing fitness data

```
 member_id  |  gym_location  |  calories_burned  |  marketing_group
----------- | -------------- | ----------------- | -----------------
  m_192     |    Miami       |        45         |        1
  m_74      |    Miami       |        59         |        1
  m_233     |    Portland    |        60         |        1

  m_14      |    Cleveland   |        72         |        2
  m_346     |    Portland    |        77         |        2
  m_289     |    Cleveland   |        81         |        2

  m_565     |    Miami       |       1085        |        50
                                 ...
```

Evenly-distributed buckets of fitness data

```sql
SELECT
    member_id,
    gym_location,
    calories_burned,

    NTILE(50) OVER(
        -- Evenly distribute records in bucket by each gym_location
        PARTITION BY gym_location
        ORDER BY calories_burned
    ) AS marketing_group

FROM FITNESS.workouts
ORDER BY marketing_group, calories_burned;
```

Evenly-distributed buckets of fitness data

```
 member_id  |  gym_location  |  calories_burned  |  marketing_group
----------- | -------------- | ----------------- | -----------------
   m_192    |    Miami       |        45         |        1
   m_233    |    Portland    |        60         |        1
   m_14     |    Cleveland   |        72         |        1

   m_74     |    Miami       |        59         |        2
   m_346    |    Portland    |        77         |        2
   m_289    |    Cleveland   |        81         |        2


                               ...
```

Understanding a distribution

- What is the distribution of the calories burned for each member's workout?
- Where does a specific workout fall in this distribution?
- What proportion of members burned the same of fewer calories than a specific member?

```
 member_id  |  cals_burned  |   cd
----------- | ------------- | ------
   m_192    |      45       |  .016
   m_74     |      59       |  .033
   m_233    |      60       |  .049
   m_14     |      72       |  .066
   m_346    |      77       |  .082
   m_289    |      81       |  .098

                    ....

   m_565    |     1085      | 1.000
```

CUME_DIST

```sql
SELECT
    member_id,
    gym_location,
    calories_burned,

    CUME_DIST() OVER(
        PARTITION BY gym_location,   -- Create a distribution for each location
        ORDER BY calories_burned
    ) AS cd

FROM FITNESS.workouts
ORDER BY gym_location, cd;  -- ORDER the final result set
```

CUME_DIST

```
SELECT
    <fields>,
    <1>,
    <2>,

    CUME_DIST() OVER(
        PARTITION BY <1>
        ORDER BY <2>
    )

...;
```

Compares each record to the distribution for that column/field, **cumulative distribution**

`<1>` : field that determines the window to evaluate

`<2>` : field to create distribution for

- Which proportion of records are less that or equal to this one?

CUME_DIST

```
 member_id  |  gym_location  | calories_burned |   cd
----------- | -------------- | --------------- | -------
   m_192    |    Miami       |       45        |  .033
   m_74     |    Miami       |       59        |  .066
   m_288    |    Miami       |       83        |  .098
   m_541    |    Miami       |       85        |  .131

                             ...


   m_233    |    Portland    |       60        |  .071
   m_346    |    Portland    |       77        |  .142

                             ...
```

Let's practice!

WINDOW FUNCTIONS IN SNOWFLAKE

LAG and LEAD

WINDOW FUNCTIONS IN SNOWFLAKE

Jake Roach
Field Data Engineer

datacamp

LAG

`LAG` allows for comparison of a value to a value in a **previous record**

```
SELECT
    <fields>,

    LAG(<1>, <2>, <3>) OVER(
        PARTITION BY <4>
        ORDER BY <5>
    )

...;
```

`<1>` : field in previous record to retrieve

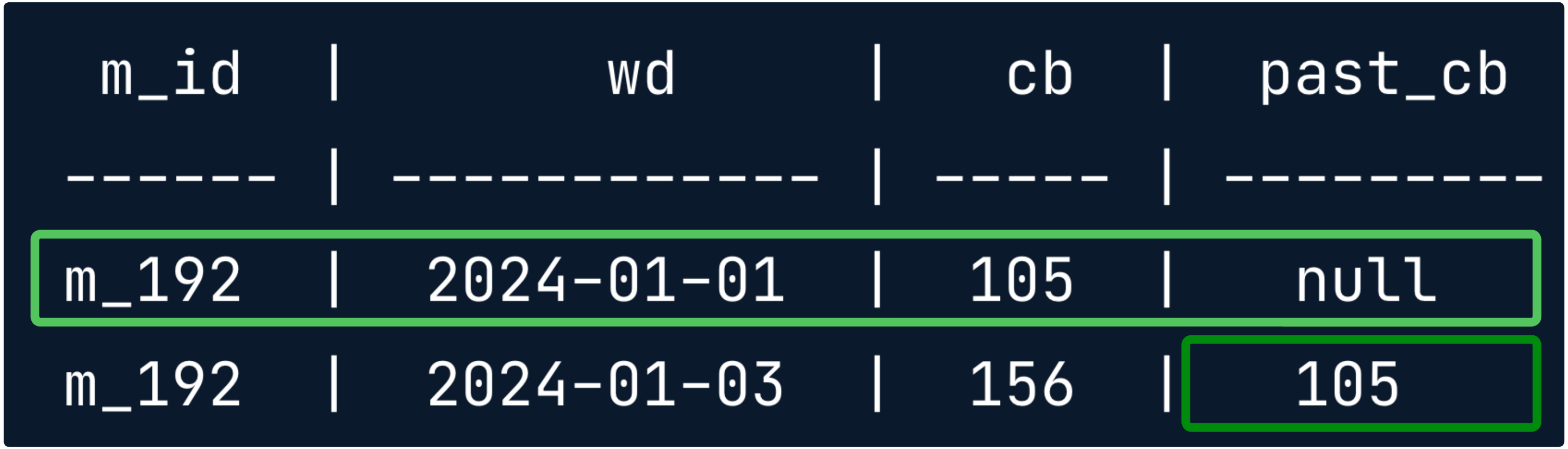
`<2>` : number of records to "look back"

`<3>` : default value if record is not there, 0

`<4>` : field to partition by

`<5>` : field to determine order of records

LAG



```
  m_id  |      wd       |  cb  |  past_cb
------- | ------------- | ---- | ---------
m_192   |  2024-01-01   | 105  |   null
m_192   |  2024-01-03   | 156  |   105
```

LAG

```sql
SELECT
    member_id AS m_id,
    workout_date AS wd,
    calories_burned AS cb,

    -- Retrieve the calories burned
    -- from the last workout
    LAG(calories_burned, 1) OVER(
        PARTITION BY member_id
        ORDER BY workout_date
    ) AS past_cb,

FROM fitness.workouts;
```

```
 m_id  |       wd     |   cb  |  past_cb
------ | ------------ | ----- | ---------
m_192  |  2024-01-01  |  105  |   null
m_192  |  2024-01-03  |  156  |   105
m_192  |  2024-01-04  |   69  |   156
m_192  |  2024-01-10  |  102  |    69

m_74   |  2024-02-10  |  374  |   null
m_74   |  2024-02-13  |  396  |   374
m_74   |  2024-02-14  |  504  |   396

m_233  |  2024-03-05  |  51   |   null
m_233  |  2024-03-12  |  81   |    51
```

LAG

~~~
SELECT
    ...
    LAG(calories_burned, 1) OVER(
        PARTITION BY member_id
        ORDER BY workout_date
    ) AS past_cb,

    -- Find the difference in the number of calories burned
``` {sql}
    calories_burned - LAG(calories_burned, 1, calories_burned) OVER(
        PARTITION BY member_id
        ORDER BY workout_date
    ) AS more_cb,

FROM fitness.workouts;
~~~

LAG

```
  m_id  |       wd      |   cb  |  past_cb  |  more_cb
------- | ------------ | ----- | --------- | ---------
 m_192  |  2024-01-01  |  105  |    null   |      0
 m_192  |  2024-01-03  |  156  |    105    |     51
 m_192  |  2024-01-04  |   69  |    156    |    -87
 m_192  |  2024-01-10  |  102  |     69    |     33

 m_74   |  2024-02-10  |  374  |    null   |      0
 m_74   |  2024-02-13  |  396  |    374    |     22
 m_74   |  2024-02-14  |  504  |    396    |    108

 m_233  |  2024-03-05  |  51   |    null   |      0
 m_233  |  2024-03-12  |  81   |     51    |     30
```

LEAD

`LEAD` allows for comparison of a value to a value in a "future" record

`<1>` : field in previous record to retrieve

`<2>` : number of records to "look ahead"

`<3>` : default value if record is not there

`<4>` : field to partition by

`<5>` : field to determine order of records

```
SELECT
    <fields>,

    LEAD(<1>, <2>, <3>) OVER(
        PARTITION BY <4>
        ORDER BY <5>
    )

...;
```

- Commonly used for predictive tasks

LEAD

```
SELECT
    member_id,
    workout_date
    calories_burned,

    -- After this workout, find the next workout date
    LEAD(workout_date, 1) OVER(
        PARTITION BY member_id
        ORDER BY workout_date
    ) AS next_workout_date

FROM fitness.workouts;
```

LEAD

```
  m_id   |  workout_date  |  calories_bured  |  next_workout_date
-------- | -------------- | ---------------- | -------------------
 m_192   |   2024-01-01   |       105        |     2024-01-03
 m_192   |   2024-01-03   |       156        |     2024-01-04
 m_192   |   2024-01-04   |        69        |     2024-01-10
 m_192   |   2024-01-10   |       102        |        null

 m_74    |   2024-02-10   |       374        |     2024-02-13
 m_74    |   2024-02-13   |       396        |     2024-02-14
 m_74    |   2024-02-14   |       504        |        null

 m_233   |   2024-03-05   |        51        |     2024-03-12
 m_233   |   2024-03-12   |        81        |        null
```

Let's practice!

WINDOW FUNCTIONS IN SNOWFLAKE