

Fetching

POSTGRES SQL SUMMARY STATS AND WINDOW FUNCTIONS



Michel Semaan
Data Scientist

The four functions

Relative

- `LAG(column, n)` returns `column`'s value at the row `n` rows before the current row
- `LEAD(column, n)` returns `column`'s value at the row `n` rows after the current row

Absolute

- `FIRST_VALUE(column)` returns the first value in the table or partition
- `LAST_VALUE(column)` returns the last value in the table or partition

LEAD

Query

```
WITH Hosts AS (  
  SELECT DISTINCT Year, City  
  FROM Summer_Medals)  
  
SELECT  
  Year, City,  
  LEAD(City, 1) OVER (ORDER BY Year ASC)  
    AS Next_City,  
  LEAD(City, 2) OVER (ORDER BY Year ASC)  
    AS After_Next_City  
FROM Hosts  
ORDER BY Year ASC;
```

Result

Year	City	Next_City	After_Next_City
1896	Athens	Paris	St Louis
1900	Paris	St Louis	London
1904	St Louis	London	Stockholm
1908	London	Stockholm	Antwerp
1912	Stockholm	Antwerp	Paris
...

FIRST_VALUE and LAST_VALUE

Query

```
SELECT
  Year, City,
  FIRST_VALUE(City) OVER
    (ORDER BY Year ASC) AS First_City,
  LAST_VALUE(City) OVER (
    ORDER BY Year ASC
    RANGE BETWEEN
      UNBOUNDED PRECEDING AND
      UNBOUNDED FOLLOWING
  ) AS Last_City
FROM Hosts
ORDER BY Year ASC;
```

Result

Year	City	First_City	Last_City
1896	Athens	Athens	London
1900	Paris	Athens	London
1904	St Louis	Athens	London
1908	London	Athens	London
1912	Stockholm	Athens	London

- By default, a window starts at the beginning of the table or partition and ends at the current row
- `RANGE BETWEEN ...` clause extends the window to the end of the table or partition

Partitioning with LEAD

- `LEAD(Champion, 1)` without `PARTITION BY`

Year	Event	Champion	Next_Champion
2004	Discus Throw	LTU	EST
2008	Discus Throw	EST	GER
2012	Discus Throw	GER	SWE
2004	Triple Jump	SWE	POR
2008	Triple Jump	POR	USA
2012	Triple Jump	USA	null

- `LEAD(Champion, 1)` with `PARTITION BY Event`

Year	Event	Champion	Next_Champion
2004	Discus Throw	LTU	EST
2008	Discus Throw	EST	GER
2012	Discus Throw	GER	null
2004	Triple Jump	SWE	POR
2008	Triple Jump	POR	USA
2012	Triple Jump	USA	null

Partitioning with FIRST_VALUE

- `FIRST_VALUE(Champion)` without `PARTITION BY Event`

Year	Event	Champion	First_Champion
2004	Discus Throw	LTU	LTU
2008	Discus Throw	EST	LTU
2012	Discus Throw	GER	LTU
2004	Triple Jump	SWE	LTU
2008	Triple Jump	POR	LTU
2012	Triple Jump	USA	LTU

- `FIRST_VALUE(Champion)` with `PARTITION BY Event`

Year	Event	Champion	First_Champion
2004	Discus Throw	LTU	LTU
2008	Discus Throw	EST	LTU
2012	Discus Throw	GER	LTU
2004	Triple Jump	SWE	SWE
2008	Triple Jump	POR	SWE
2012	Triple Jump	USA	SWE

Let's practice!

POSTGRESQL SUMMARY STATS AND WINDOW FUNCTIONS

Ranking

POSTGRESQL SUMMARY STATS AND WINDOW FUNCTIONS



Michel Semaan
Data Scientist

The ranking functions

- `ROW_NUMBER()` always assigns unique numbers, even if two rows' values are the same
- `RANK()` assigns the same number to rows with identical values, skipping over the next numbers in such cases
- `DENSE_RANK()` also assigns the same number to rows with identical values, but doesn't skip over the next numbers

Source table

Query

```
SELECT
  Country, COUNT(DISTINCT Year) AS Games
FROM Summer_Medals
WHERE
  Country IN ('GBR', 'DEN', 'FRA',
              'ITA', 'AUT', 'BEL',
              'NOR', 'POL', 'ESP')
GROUP BY Country
ORDER BY Games DESC;
```

Result

Country	Games
-----	-----
GBR	27
DEN	26
FRA	26
ITA	25
AUT	24
BEL	24
NOR	22
POL	20
ESP	18

Different ranking functions - ROW_NUMBER

Query

```
WITH Country_Games AS (...)  
  
SELECT  
  Country, Games,  
  ROW_NUMBER()  
    OVER (ORDER BY Games DESC) AS Row_N  
FROM Country_Games  
ORDER BY Games DESC, Country ASC;
```

Result

Country	Games	Row_N
GBR	27	1
DEN	26	2
FRA	26	3
ITA	25	4
AUT	24	5
BEL	24	6
NOR	22	7
POL	20	8
ESP	18	9

Different ranking functions - RANK

Query

```
WITH Country_Games AS (...)  
  
SELECT  
  Country, Games,  
  ROW_NUMBER()  
    OVER (ORDER BY Games DESC) AS Row_N,  
  RANK()  
    OVER (ORDER BY Games DESC) AS Rank_N  
FROM Country_Games  
ORDER BY Games DESC, Country ASC;
```

Result

Country	Games	Row_N	Rank_N
GBR	27	1	1
DEN	26	2	2
FRA	26	3	2
ITA	25	4	4
AUT	24	5	5
BEL	24	6	5
NOR	22	7	7
POL	20	8	8
ESP	18	9	9

Different ranking functions - DENSE_RANK

Query

```
WITH Country_Games AS (...)  
  
SELECT  
  Country, Games,  
  ROW_NUMBER()  
    OVER (ORDER BY Games DESC) AS Row_N,  
  RANK()  
    OVER (ORDER BY Games DESC) AS Rank_N,  
  DENSE_RANK()  
    OVER (ORDER BY Games DESC) AS Dense_Rank_N  
FROM Country_Games  
ORDER BY Games DESC, Country ASC;
```

- **ROW_NUMBER** and **RANK** will have the same last rank, the count of rows

Result

Country	Games	Row_N	Rank_N	Dense_Rank_N
GBR	27	1	1	1
DEN	26	2	2	2
FRA	26	3	2	2
ITA	25	4	4	3
AUT	24	5	5	4
BEL	24	6	5	5
NOR	22	7	7	5
POL	20	8	8	6
ESP	18	9	9	7

- **DENSE_RANK** 's last rank is the count of unique values being ranked

Ranking without partitioning - Source table

Query

```
SELECT
  Country, Athlete, COUNT(*) AS Medals
FROM Summer_Medals
WHERE
  Country IN ('CHN', 'RUS')
  AND Year = 2012
GROUP BY Country, Athlete
HAVING COUNT(*) > 1
ORDER BY Country ASC, Medals DESC;
```

Result

Country	Athlete	Medals
CHN	SUN Yang	4
CHN	Guo Shuang	3
CHN	WANG Hao	3
...
RUS	MUSTAFINA Aliya	4
RUS	ANTYUKH Natalya	2
RUS	ISHCHENKO Natalia	2
...

Ranking without partitioning

Query

```
WITH Country_Medals AS (...)  
  
SELECT  
  Country, Athlete, Medals,  
  DENSE_RANK()  
    OVER (ORDER BY Medals DESC) AS Rank_N  
FROM Country_Medals  
ORDER BY Country ASC, Medals DESC;
```

Result

Country	Athlete	Medals	Rank_N
CHN	SUN Yang	4	1
CHN	Guo Shuang	3	2
CHN	WANG Hao	3	2
...
RUS	MUSTAFINA Aliya	4	1
RUS	ANTYUKH Natalya	2	3
RUS	ISHCHENKO Natalia	2	3
...

Ranking with partitioning

Query

```
WITH Country_Medals AS (...)  
  
SELECT  
  Country, Athlete,  
  DENSE_RANK()  
    OVER (PARTITION BY Country  
          ORDER BY Medals DESC) AS Rank_N  
FROM Country_Medals  
ORDER BY Country ASC, Medals DESC;
```

Result

Country	Athlete	Medals	Rank_N
CHN	SUN Yang	4	1
CHN	Guo Shuang	3	2
CHN	WANG Hao	3	2
...
RUS	MUSTAFINA Aliya	4	1
RUS	ANTYUKH Natalya	2	2
RUS	ISHCHENKO Natalia	2	2
...

Let's practice!

POSTGRESQL SUMMARY STATS AND WINDOW FUNCTIONS

Paging

POSTGRES SQL SUMMARY STATS AND WINDOW FUNCTIONS



Michel Semaan
Data Scientist

What is paging?

- **Paging:** Splitting data into (approximately) equal chunks
- **Uses**
 - Many APIs return data in "pages" to reduce data being sent
 - Separating data into quartiles or thirds (top middle 33%, and bottom thirds) to judge performance

Enter NTILE

- `NTILE(n)` splits the data into `n` approximately equal pages

Paging - Source table

Query

```
SELECT
  DISTINCT Discipline
FROM Summer_Medals;
```

- Split the data into 15 approx. equally sized pages
- $67/15 \simeq 4$, so each each page will contain four or five rows

Result

```
| Discipline |
|-----|
| Wrestling Freestyle |
| Archery |
| Baseball |
| Lacrosse |
| Judo |
| Athletics |
| ... |
```

(67 rows)

Paging

Query

```
WITH Disciplines AS (  
  SELECT  
    DISTINCT Discipline  
  FROM Summer_Medals)  
  
SELECT  
  Discipline, NTILE(15) OVER () AS Page  
FROM Disciplines  
ORDER BY Page ASC;
```

Result

Discipline	Page
Wrestling Freestyle	1
Archery	1
Baseball	1
Lacrosse	1
Judo	1
Athletics	2
...	...

Top, middle, and bottom thirds

Query

```
WITH Country_Medals AS (  
  SELECT  
    Country, COUNT(*) AS Medals  
  FROM Summer_Medals  
  GROUP BY Country),  
  
SELECT  
  Country, Medals,  
  NTILE(3) OVER (ORDER BY Medals DESC) AS Third  
FROM Country_Medals;
```

Result

Country	Medals	Third
USA	4585	1
URS	2049	1
GBR	1720	1
...
CZE	56	2
LTU	55	2
...
DOM	6	3
BWI	5	3
...

Thirds averages

Query

```
WITH Country_Medals AS (...),

Thirds AS (
  SELECT
    Country, Medals,
    NTILE(3) OVER (ORDER BY Medals DESC) AS Third
  FROM Country_Medals)

SELECT
  Third,
  ROUND(AVG(Medals), 2) AS Avg_Medals
FROM Thirds
GROUP BY Third
ORDER BY Third ASC;
```

Result

Third	Avg_Medals
1	598.74
2	22.98
3	2.08

Let's practice!

POSTGRESQL SUMMARY STATS AND WINDOW FUNCTIONS