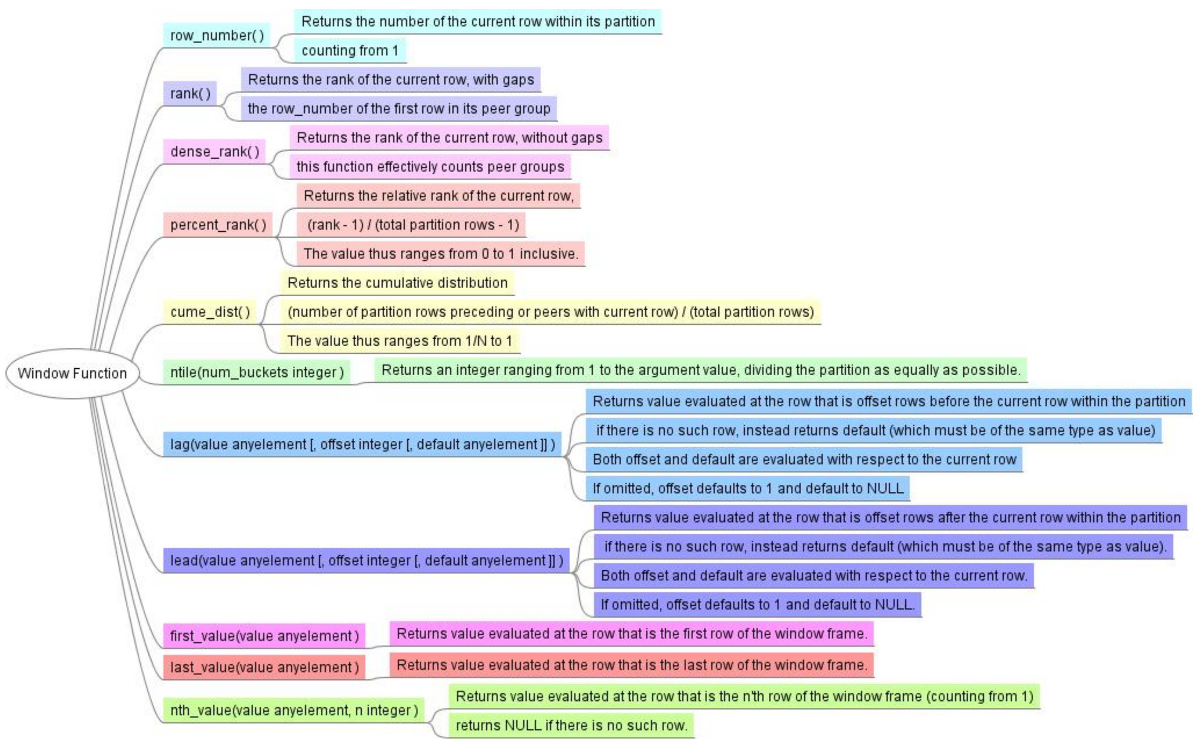
Window Function

A *window function* performs a calculation across a set of table rows that are somehow related to the current row. This is comparable to the type of calculation that can be done with an aggregate function.

However, **window functions do not cause rows to become grouped into a single output row like non-window aggregate calls would**. Instead, the rows retain their separate identities. Behind the scenes, the window function is able to access more than just the current row of the query result.

```
[ existing_window_name ]
[ PARTITION BY expression [, ...] ]
[ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS { FIRST | LAST } ]
[, ...] ]
[ frame_clause ]
```

List of General-Purpose Window Functions


Window Function
row_number( )
Returns the number of the current row within its partition
counting from 1
rank( )
Returns the rank of the current row, with gaps
the row_number of the first row in its peer group
dense_rank( )
Returns the rank of the current row, without gaps
this function effectively counts peer groups
percent_rank( )
Returns the relative rank of the current row,
(rank - 1) / (total partition rows - 1)
The value thus ranges from 0 to 1 inclusive.
cume_dist( )
Returns the cumulative distribution
(number of partition rows preceding or peers with current row) / (total partition rows)
The value thus ranges from 1/N to 1
ntile(num_buckets integer)
Returns an integer ranging from 1 to the argument value, dividing the partition as equally as possible.
lag(value anyelement [, offset integer [, default anyelement]])
Returns value evaluated at the row that is offset rows before the current row within the partition
if there is no such row, instead returns default (which must be of the same type as value)
Both offset and default are evaluated with respect to the current row
If omitted, offset defaults to 1 and default to NULL
lead(value anyelement [, offset integer [, default anyelement]])
Returns value evaluated at the row that is offset rows after the current row within the partition
if there is no such row, instead returns default (which must be of the same type as value).
Both offset and default are evaluated with respect to the current row.
If omitted, offset defaults to 1 and default to NULL.
first_value(value anyelement)
Returns value evaluated at the row that is the first row of the window frame.
last_value(value anyelement)
Returns value evaluated at the row that is the last row of the window frame.
nth_value(value anyelement, n integer)
Returns value evaluated at the row that is the n'th row of the window frame (counting from 1)
returns NULL if there is no such row.

https://www.postgresql.org/docs/13/functions-window.html

OVER()

A window function call always contains an `OVER` clause directly following the window function's name and argument(s). This is what syntactically distinguishes it from a normal function or non-window aggregate. The `OVER` clause determines exactly how the rows of the query are split up for processing by the window function.

Experiment 1:

Over, and window func vs aggregate func

```
SELECT movieid, title, runtime, year_released,
    sum(runtime) OVER () sum_all,
    max(runtime) OVER () max,
    avg(runtime) OVER() avg
FROM movies
WHERE country='cn';

SELECT sum(runtime) ,
    max(runtime) ,
    avg(runtime)
FROM movies
WHERE country='cn';
--you can also test:SELECT runtime,sum(runtime) from...syntax error
```

窗口为空 在from和where选择的全局上产生作用

即使在最后加上group by movidid,title
但含义也不再相同：表示的是相同moviedid title 下的最大值和平均值

Aggregation with over

The `PARTITION BY` clause within `OVER` divides the rows into groups, or partitions, that share the same values of the `PARTITION BY` expression(s). For each row, the window function is computed across the rows that fall into the same partition as the current row. You can also control the order in which rows are processed by window functions using `ORDER BY` within `OVER`. (The window `ORDER BY` does not even have to match the order in which rows are output.)

Experiment 2:

OVER (PARTITION BY ... ORDER BY...)

对于over() sum_all会是同一个值
而如果改为over(order by movieid)
--sum all等于从第一个值到当前值的加和

```
SELECT country,
       title,
       runtime,
       year_released,
       rank() OVER (order by year_released)
FROM movies
WHERE year_released > 2015;

SELECT country,
       title,
       runtime,
       year_released,
       rank() OVER (partition by country)  as win_func
FROM movies
WHERE year_released > 2015;

SELECT country,
       title,
       runtime,
       year_released,
       rank() OVER (partition by country order by year_released) win_func
```

相同country的为一组 他们的win_func是相同的

```
FROM movies
WHERE year_released > 2015;
```

Experiment 3:

ORDER BY... invalidated

```
SELECT country, title, runtime, year_released,        没什么实际意义）
    avg(runtime) OVER ~~(partition by country order by year_released)~~
sum_by_country              partition by country,year_released 每个国家每年的平均值
FROM movies
WHERE year_released>2015;
```

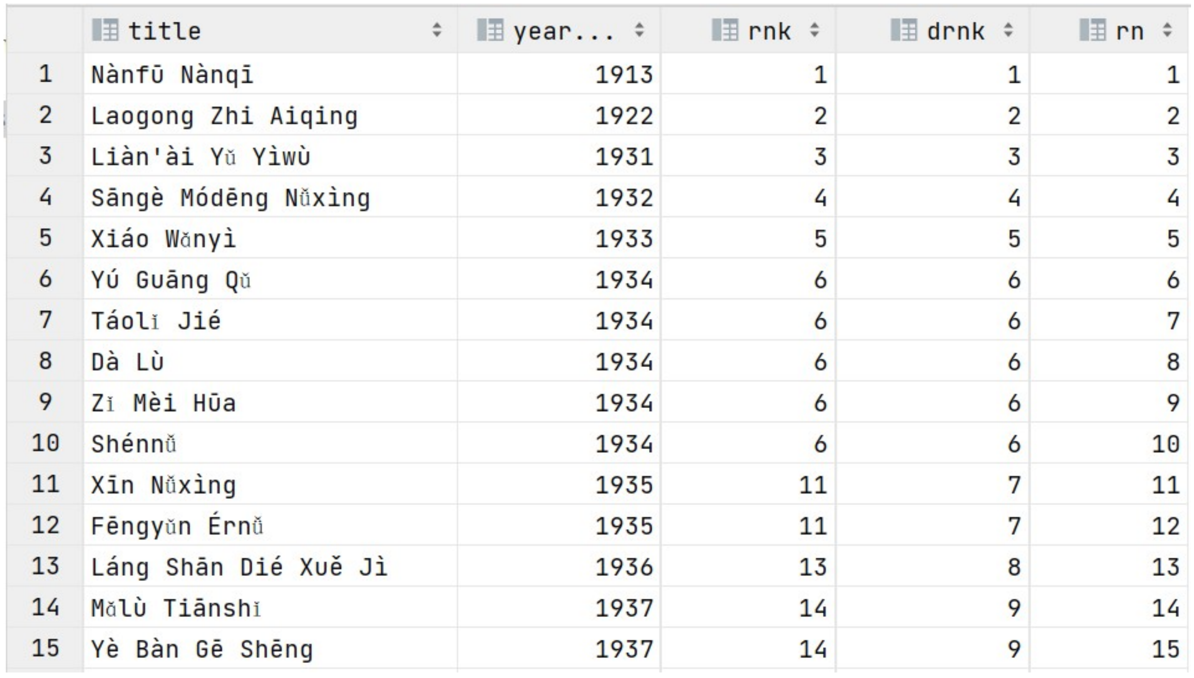
Tips: The window `ORDER BY` does not even have to match the order in which the rows are output. The following Queries are not same

```
SELECT country,
       title,
       runtime,
       year_released,
       avg(runtime)  OVER (partition by country order by year_released) win_func
FROM movies
WHERE year_released > 2015;

SELECT country,title,
       runtime,
       year_released,
       avg(runtime)  OVER (partition by country,year_released) win_func
FROM movies
WHERE year_released > 2015;
```

RANK()/DENSE_RANK()/ROW_NUMBER()

`rank` needs no explicit parameter, because its behavior is entirely determined by the `OVER` clause.



	title	year...	rnk	drnk	rn
1	Nànfū Nànqī	1913	1	1	1
2	Laogong Zhi Aiqing	1922	2	2	2
3	Liàn'ài Yǔ Yìwù	1931	3	3	3
4	Sāngè Módēng Nǚxìng	1932	4	4	4
5	Xiáo Wǎnyì	1933	5	5	5
6	Yú Guāng Qǔ	1934	6	6	6
7	Táolǐ Jié	1934	6	6	7
8	Dà Lù	1934	6	6	8
9	Zǐ Mèi Hūa	1934	6	6	9
10	Shénnǚ	1934	6	6	10
11	Xīn Nǚxìng	1935	11	7	11
12	Fēngyǔn Érnǚ	1935	11	7	12
13	Láng Shān Dié Xuě Jì	1936	13	8	13
14	Mǎlù Tiānshǐ	1937	14	9	14
15	Yè Bàn Gē Shēng	1937	14	9	15

Experiment 4:

```
--Returns the rank of the current row, with gaps
select title, year_released,
    rank() over (order by year_released) rnk
from movies
where country = 'cn';
```

```
...
1934  7
1934  7
1934  7
1934  7
1934  7
1935  12
...
```

```
--Returns the rank of the current row, without gaps
select title, year_released,
    dense_rank() over (order by year_released) rnk
from movies
where country = 'cn';
```

```
...
1934  7
1934  7
1934  7
1934  7
1934  7
1935  8
...
```

```
select title, year_released,
    row_number()  over (order by year_released) rnk
from movies
where country = 'cn';
```

```
...
1934  7
1934  8
1934  9
1934  10
1934  11
1935  12
...
```

```
select title, year_released,
    rank() over (order by year_released desc) rnk
from movies
where country = 'cn';
```

LAG()

> *lag(value anyelement [offset integer [,defaultanyelement]])*

Returns `value` evaluated at the row that is `offset` rows before the current row within the partition; if there is no such row, instead returns `default` (which must be of the same type as `value`). Both `offset` and `default` are evaluated with respect to the current row. If omitted, `offset` defaults to 1 and `default` to `NULL`.

Experiment 5:

```sql
SELECT title, runtime, year_released,
    lag(year_released) OVER (order by year_released)
FROM movies
WHERE country = 'cn';--default offset is 1; default placeholder is null

SELECT title, runtime, year_released,
    lag(year_released, 1, 0) OVER (order by year_released)
FROM movies
WHERE country = 'cn';--offset is 1; placeholder is 0

SELECT title, runtime, year_released,
    lag(year_released, 3) OVER (order by year_released)
FROM movies
WHERE country = 'cn';

SELECT title, runtime, year_released,
    lag(year_released, 3, 0) OVER (order by year_released)
FROM movies
WHERE country = 'cn';
```

Multi-Window Functions

Experiment 6:

```sql
 SELECT movieid, title, country, runtime, year_released,
        sum(runtime) OVER w sum_all,
        max(runtime) OVER w max,
        avg(runtime) OVER w avg
 FROM movies
 WHERE year_released>2010
 WINDOW w AS (partition by country order by year_released);
```

Tips:

Window functions are **permitted only** in the `SELECT` list and the `ORDER BY` clause of the query. They are **forbidden** elsewhere, such as in `GROUP BY`, `HAVING` and `WHERE` clauses. This is because they logically execute after the processing of those clauses. Also, window functions execute after non-window aggregate functions. This means it is valid to include an aggregate function call in the arguments of a window function, but not vice versa.

Summary

- window function will not aggregate multi-record to output a single one, which has big different with aggregate function
- window function always has `over`
- there could be several window function in one select query
- window function deal with the data in virtual table filtered by where, group by, having clauses if any

- if there is no order by or partition by, window function deals with all records