MySQL Window Functions

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.

Standard query

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 |
```

Group by query

- SELECT country, SUM(profit) AS country_profit
 FROM sales
 GROUP BY country ORDER BY country;

```
+---------+----------------+
| country | country_profit |
+---------+----------------+
| Finland |           1610 |
| India   |           1350 |
| USA     |           4575 |
+---------+----------------+
```

Examples

```
SELECT
  year, country, product, profit,
  SUM(profit) OVER() AS total_profit,   #grand total
  SUM(profit) OVER(PARTITION BY country) AS country_profit
FROM sales
ORDER BY country, year, product, profit;
```

Output of the prev query

```
+------+---------+------------+--------+--------------+----------------+
| 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 |
+------+---------+------------+--------+--------------+----------------+
```

Sequence of execution

- 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.

Over() over what?

The OVER clause is permitted for many aggregate functions

- Note that Aggregate functions 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()`

Non-aggregate fns in Over

- CUME_DIST()
- DENSE_RANK()
- FIRST_VALUE()
- LAG()
- LAST_VALUE()
- LEAD()
- NTH_VALUE()
- NTILE()
- PERCENT_RANK()
- RANK()
- ROW_NUMBER()

Examples of Non-aggregate

```
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;
```

Output of the previous query

```
+------+---------+------------+--------+----------+----------+
| 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 |
+------+---------+------------+--------+----------+----------+
```

- over_clause:

  {OVER (window_spec) | OVER window_name}
- window_spec:

  [window_name] [partition_clause] [order_clause] [frame_clause]
- partition_clause:

  PARTITION BY expr [, expr] ...

order_clause:

  ORDER BY expr [ASC|DESC] [, expr [ASC|DESC]] ...

Types of Window Function

https://www.javatpoint.com/mysql-window-functions

We can categorize the window functions mainly in three types

Aggregate Functions

It is a function that operates on multiple rows and produces the result in a single row. Some of the important aggregate functions are:

COUNT, SUM, AVG, MIN, MAX, and many more.

Types of Window Function

Ranking Functions

It is a function that allows us to rank each row of a partition in a given table. Some of the important ranking functions are:

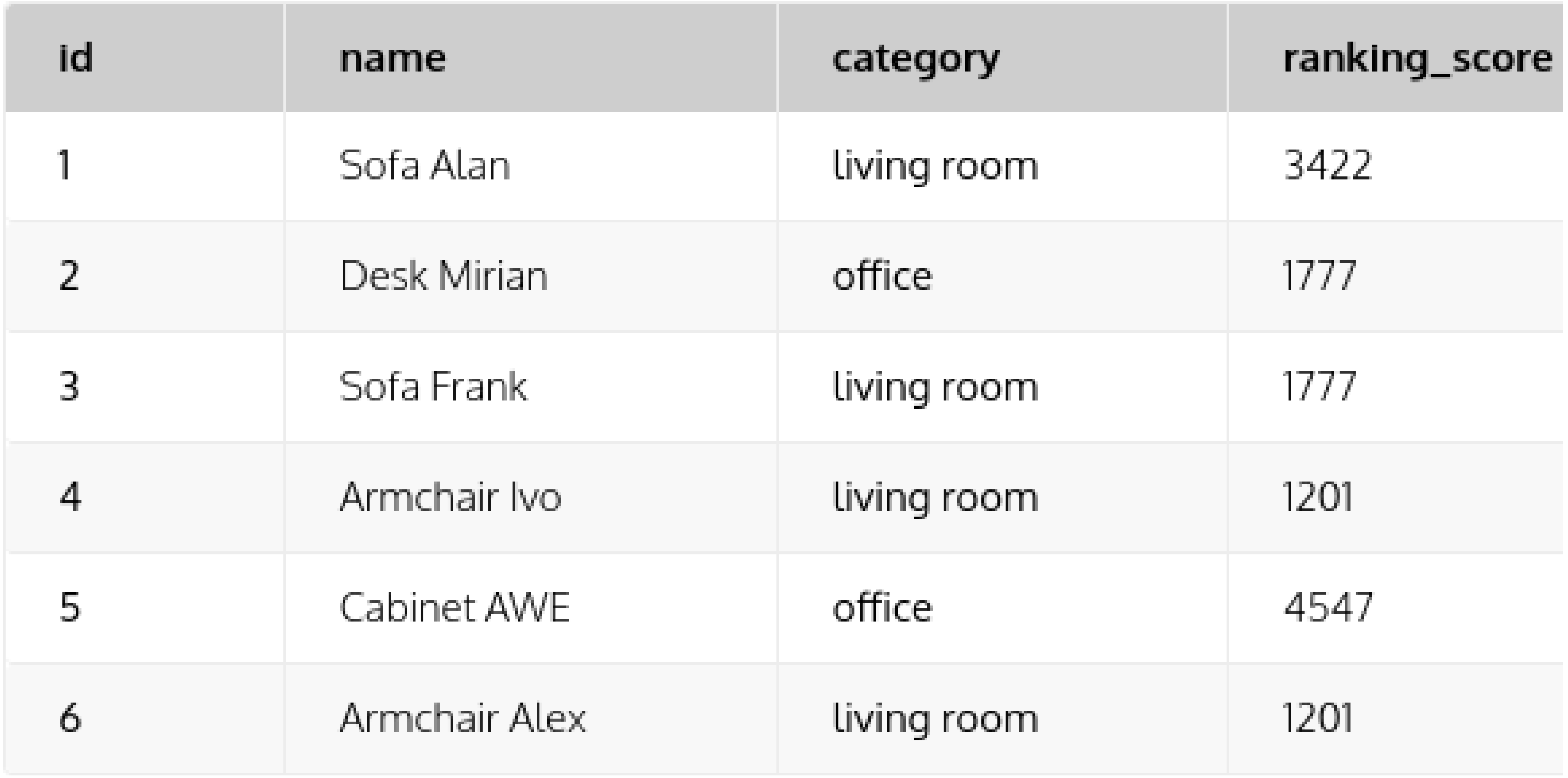
RANK, DENSE_RANK, PERCENT_RANK, ROW_NUMBER, CUME_DIST, etc.

Analytical Functions

It is a function, which is locally represented by a power series. Some of the important analytical functions are:

NTILE, LEAD, LAG, NTH, FIRST_VALUE, LAST_VALUE, etc.

Example 1



id	name	category	ranking_score
1	Sofa Alan	living room	3422
2	Desk Mirian	office	1777
3	Sofa Frank	living room	1777
4	Armchair Ivo	living room	1201
5	Cabinet AWE	office	4547
6	Armchair Alex	living room	1201

Rank Query

```
SELECT

RANK() OVER(ORDER BY ranking_score) AS
rank_number,

name, category, ranking_score

FROM product;
```

rank_number	name	category	ranking_score
1	Armchair Ivo	living room	1201
1	Armchair Alex	living room	1201
3	Desk Mirian	office	1777
3	Sofa Frank	living room	1777
5	Sofa Alan	living room	3422
6	Cabinet AWE	office	4547

DENSE_RANK Query

```
SELECT
DENSE_RANK() OVER(ORDER BY ranking_score
DESC) AS dense_rank_number,
name, category, ranking_score
FROM product;
```

dense_rank_number	name	category	ranking_score
1	Cabinet AWE	office	4547
2	Sofa Alan	living room	3422
3	Desk Mirian	office	1777
3	Sofa Frank	living room	1777
4	Armchair Ivo	living room	1201
4	Armchair Alex	living room	1201

What happens?

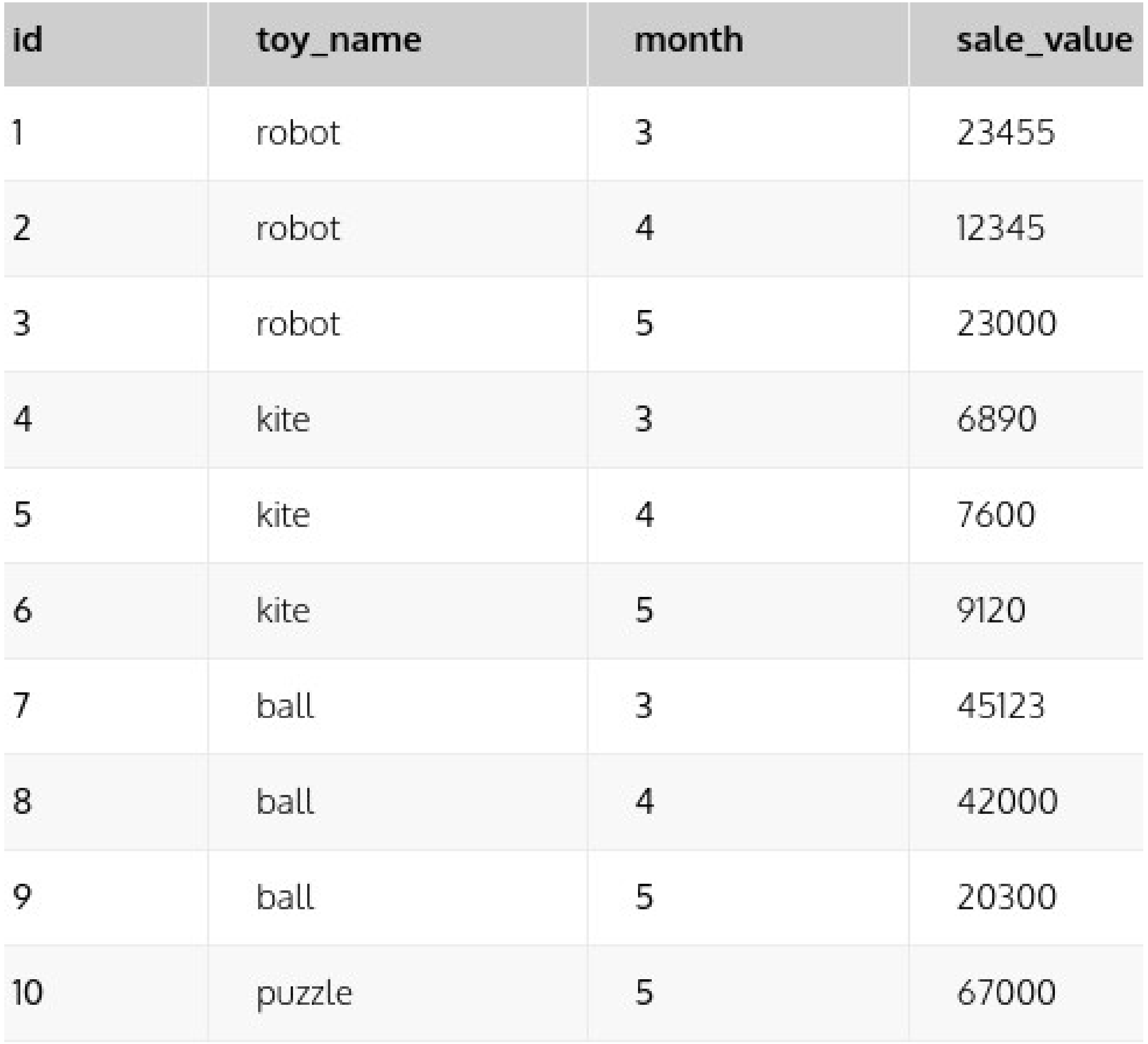
```
SELECT

RANK() OVER(PARTITION BY category ORDER
BY ranking_score) AS rank_number,

name, category, ranking_score

FROM product;
```

Analytic functions: Eg



id	toy_name	month	sale_value
1	robot	3	23455
2	robot	4	12345
3	robot	5	23000
4	kite	3	6890
5	kite	4	7600
6	kite	5	9120
7	ball	3	45123
8	ball	4	42000
9	ball	5	20300
10	puzzle	5	67000

LEAD()

```
SELECT
   toy_name, month, sale_value,
   LEAD(sale_value) OVER(PARTITION BY toy_name ORDER BY
month)
     AS next_month_value
FROM toys_sale;
```

The LEAD() function returns the value of sale_value in the next row.

toy_name	month	sale_value	next_month_
ball	3	45123	42000
ball	4	42000	20300
ball	5	20300	**NULL**
kite	3	6890	7600
kite	4	7600	9120
kite	5	9120	**NULL**
puzzle	5	67000	**NULL**
robot	3	23455	12345
robot	4	12345	23000
robot	5	23000	**NULL**

LAG() : Guess what happens

```
SELECT toy_name, month, sale_value,
  LAG(sale_value) OVER(PARTITION BY toy_name
ORDER BY month) AS prev_month_value,
  LAG(sale_value) OVER(PARTITION BY toy_name
ORDER BY month)- sale_value as difference
FROM toys_sale;
```

SQL Window Cheat Sheet

- https://learnsql.com/blog/sql-window-functions-cheat-sheet/