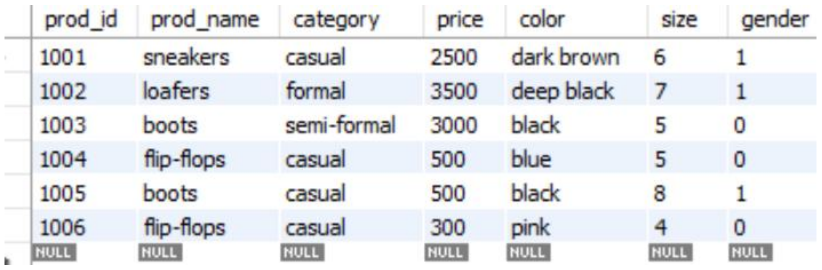
Window Functions and Ranking - Task 7

RDBMS used : MySql

Window functions are used to perform calculations across a set of table rows.

- **Displaying table rows**

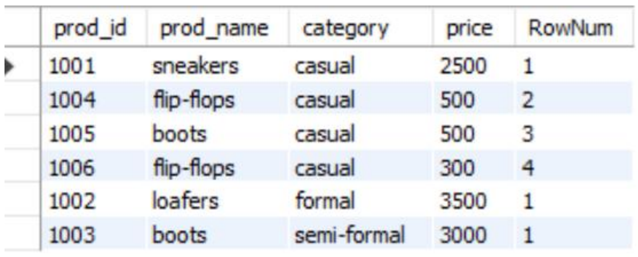
  select * from products;



prod_id	prod_name	category	price	color	size	gender
1001	sneakers	casual	2500	dark brown	6	1
1002	loafers	formal	3500	deep black	7	1
1003	boots	semi-formal	3000	black	5	0
1004	flip-flops	casual	500	blue	5	0
1005	boots	casual	500	black	8	1
1006	flip-flops	casual	300	pink	4	0
NULL	NULL	NULL	NULL	NULL	NULL	NULL

- **ROW_NUMBER()**
  - Assigns a unique sequential number to each row within a partition.
  - No two rows will have the same number.
  - Each row gets a unique number, even if prices are the same.

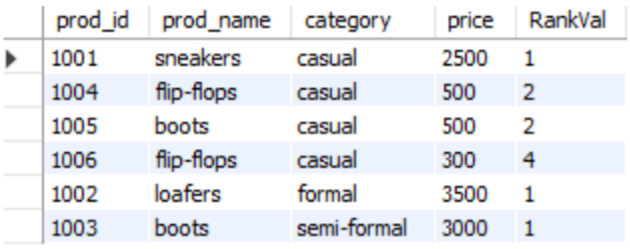
```
select prod_id, prod_name, category, price,

row_number() over (partition by category order by price desc) as RowNum

from products;
```



prod_id	prod_name	category	price	RowNum
1001	sneakers	casual	2500	1
1004	flip-flops	casual	500	2
1005	boots	casual	500	3
1006	flip-flops	casual	300	4
1002	loafers	formal	3500	1
1003	boots	semi-formal	3000	1

- **RANK()**
  - Ranks rows within a partition.
  - If multiple rows have the same value, they get the same rank, but the next rank is skipped.

```
select prod_id, prod_name, category, price,
rank() over (partition by category order by price desc) as RankVal
from products;
```



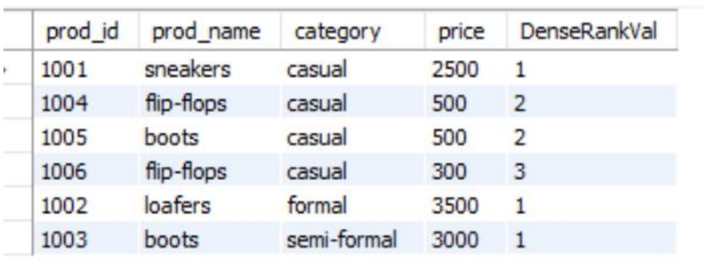
prod_id	prod_name	category	price	RankVal
1001	sneakers	casual	2500	1
1004	flip-flops	casual	500	2
1005	boots	casual	500	2
1006	flip-flops	casual	300	4
1002	loafers	formal	3500	1
1003	boots	semi-formal	3000	1

- 1004 and 1005 products have the same price, so they share Rank 2.
- Rank 3 is skipped (the next rank is 4).

➢ **DENSE_RANK()**

- Similar to RANK(), but it does not skip ranks.

  select prod_id, prod_name, category, price,
  dense_rank() over (partition by category order by price desc) as
  DenseRankVal from products;



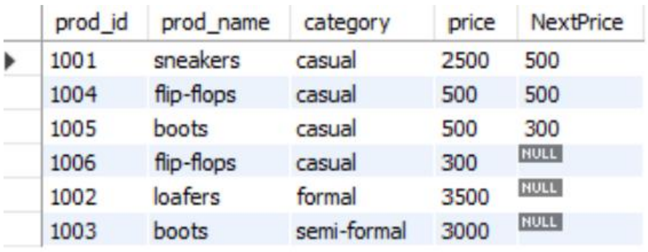
prod_id	prod_name	category	price	DenseRankVal
1001	sneakers	casual	2500	1
1004	flip-flops	casual	500	2
1005	boots	casual	500	2
1006	flip-flops	casual	300	3
1002	loafers	formal	3500	1
1003	boots	semi-formal	3000	1

- 1004 and 1005 products still share Rank 2.
- But now, the next rank is 3 instead of skipping to 4.

➢ **LEAD()**

- Retrieves the ***next*** row's value within the partition.

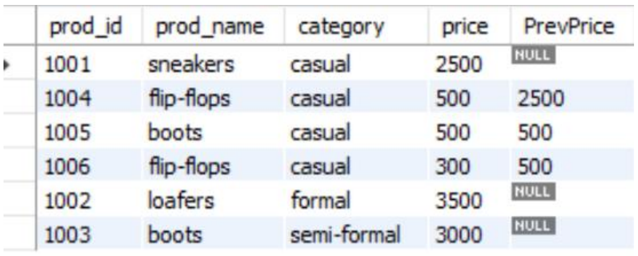
  select prod_id, prod_name, category, price,
  lead(price) over (partition by category order by price desc) as NextPrice
  from products;



prod_id	prod_name	category	price	NextPrice
1001	sneakers	casual	2500	500
1004	flip-flops	casual	500	500
1005	boots	casual	500	300
1006	flip-flops	casual	300	NULL
1002	loafers	formal	3500	NULL
1003	boots	semi-formal	3000	NULL

- **LAG()**
  - Retrieves the ***previous*** row’s value within the partition.

```
select prod_id, prod_name, category, price,
lag(price) over (partition by category order by price desc) as PrevPrice
from products;
```



prod_id	prod_name	category	price	PrevPrice
1001	sneakers	casual	2500	NULL
1004	flip-flops	casual	500	2500
1005	boots	casual	500	500
1006	flip-flops	casual	300	500
1002	loafers	formal	3500	NULL
1003	boots	semi-formal	3000	NULL