Working with Window Functions

Mihaela Danci
Data Analyst

linkedin.com/in/mihaela-danci/

Aggregate Functions

```
SELECT
    OrderDate,
    COUNT(OrderNumber) AS Orders,
    SUM(OrderAmount) AS Amount
FROM Orders
WHERE SalesTeritorry = ‘USA’
GROUP BY OrderDate
ORDER BY OrderDate DESC;
```

All non-aggregate columns must be included in the GROUP BY clause

Result Table

OrderDate	Orders	Amount
01012023	2	880
15052023	2	600
06062023	1	500

```
SELECT OrderDate, COUNT(OrderNumber) AS Orders, SUM(OrderAmount) AS Amount
FROM Orders
WHERE SalesTeritorry = ‘USA’
GROUP BY OrderDate
ORDER BY OrderDate DESC;
```

Scenario

OrderKey	CustKey	Amount	CustTotal	Total
1	1	100	600	1600
2	1	200	600	1600
3	1	300	600	1600
4	2	500	1000	1600
5	2	500	1000	1600

Scenario

OrderKey	CustKey	Amount	CustTotal	Total
1	1	100	600	1600
2	1	200	600	1600
3	1	300	600	1600
4	2	500	1000	1600
5	2	500	1000	1600

Scenario

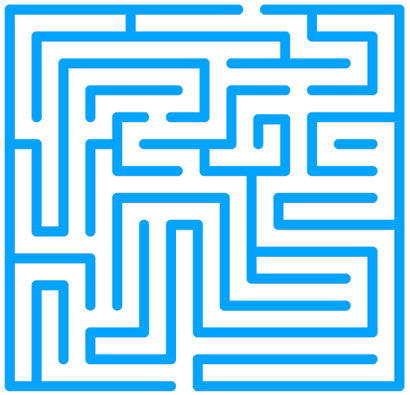
OrderKey	CustKey	Amount	CustTotal	Total
1	1	100	600	1600
2	1	200	600	1600
3	1	300	600	1600
4	2	500	1000	1600
5	2	500	1000	1600

Possible Solutions

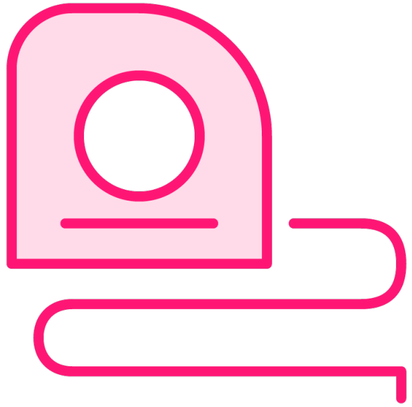
Table expressions

Subqueries

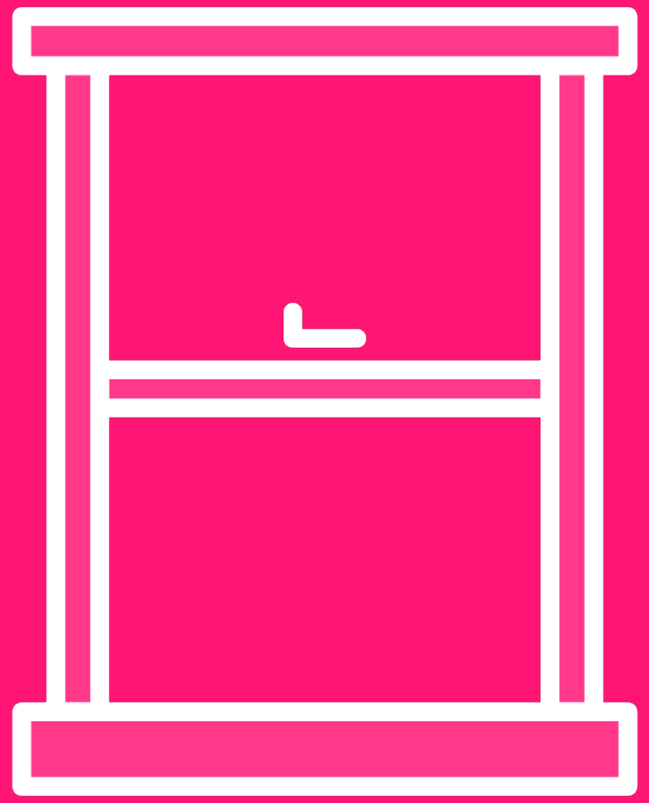
Drawbacks



Complex



Lengthy



Window Functions

Perform aggregation without the need to group data.

Window Functions

```
function_name(<aggregation_element>) OVER (<set_of_rows>)
```

Window Functions

```
function_name(<aggregation_element>) OVER (<window>)
```

Scenario

OrderKey	CustKey	Amount	CustTotal
1	1	100	600
2	1	200	600
3	1	300	600
4	2	500	1000
5	2	500	1000

```
SELECT OrderKey, CustKey, Amount
FROM Orders;
```

Scenario

OrderKey	CustKey	Amount	CustTotal
1	1	100	600
2	1	200	600
3	1	300	600
4	2	500	1000
5	2	500	1000

```
SELECT OrderKey, CustKey, Amount
       SUM(Amount) OVER(PARTITION BY CustKey) AS CustTotal
FROM Orders;
```

Scenario

OrderKey	CustKey	Amount	CustTotal
1	1	100	600
2	1	200	600
3	1	300	600
4	2	500	1000
5	2	500	1000

```
SELECT OrderKey, CustKey, Amount
       SUM(Amount) OVER(PARTITION BY CustKey) AS CustTotal
FROM Orders;
```

Scenario

OrderKey	CustKey	Amount	CustTotal
1	1	100	600
2	1	200	600
3	1	300	600
4	2	500	1000
5	2	500	1000

```
SELECT OrderKey, CustKey, Amount,
       SUM(Amount) OVER(PARTITION BY CustKey) AS CustTotal
FROM Orders;
```

Scenario

OrderKey	CustKey	Amount	CustTotal	Total
1	1	100	600	1600
2	1	200	600	1600
3	1	300	600	1600
4	2	500	1000	1600
5	2	500	1000	1600

```
SELECT OrderKey, CustKey, Amount,
       SUM(Amount) OVER(PARTITION BY CustKey) AS CustTotal
       SUM(Amount) OVER() AS Total
FROM Orders;
```

Benefits

Easy to read

Efficient

Usage

Running totals

Moving averages

Rankings

Categories

Aggregate

Ranking

Offset

Statistical

Overview

OVER clause

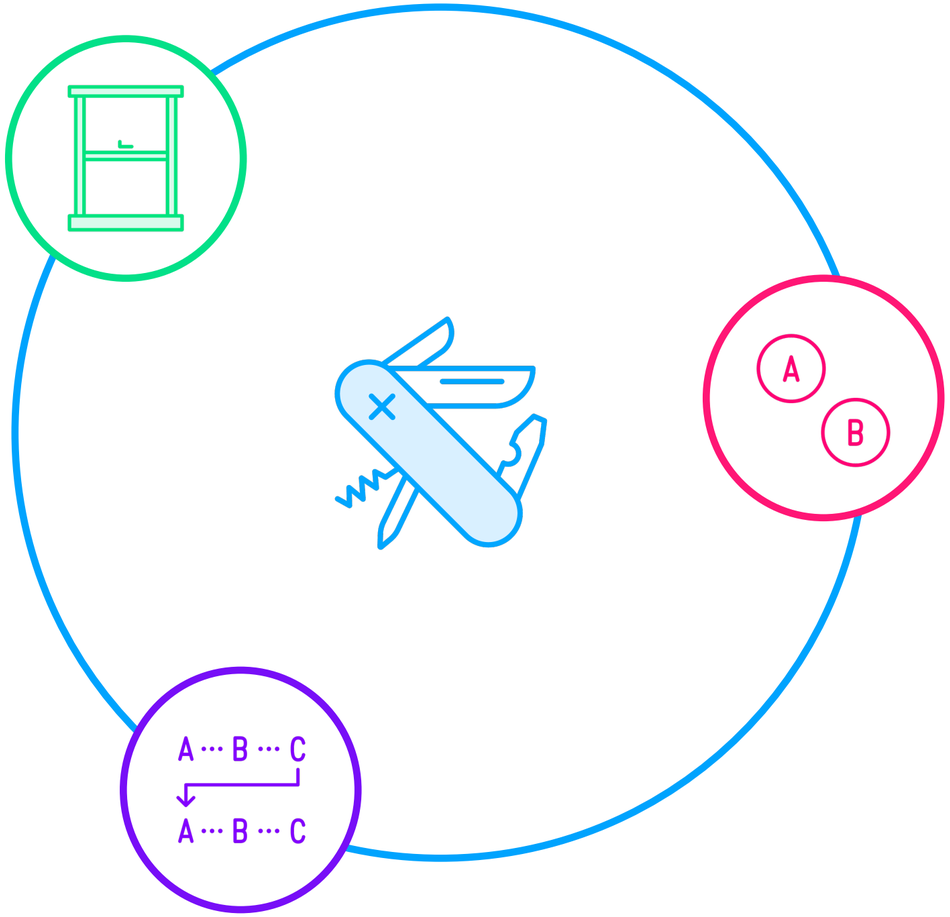
Ranking functions

Aggregate functions

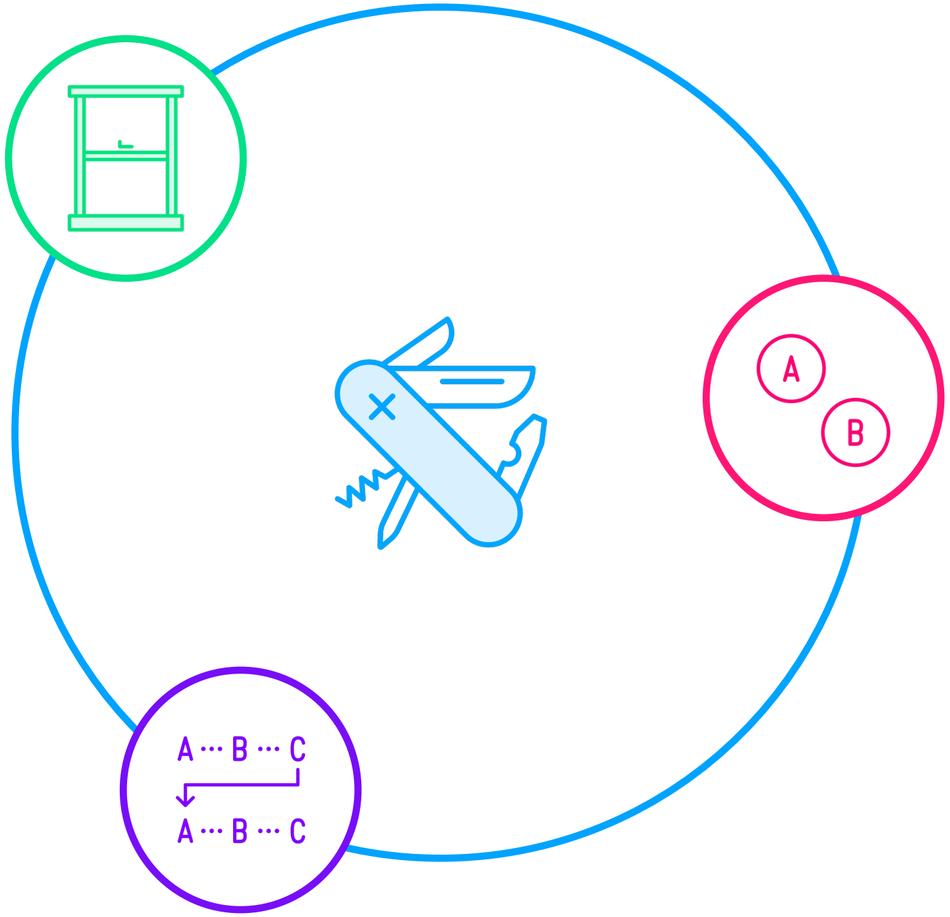
Offset functions

The OVER Clause

The OVER Clause



The OVER Clause



Restricts the window

The OVER Clause

```
function_name(<agg_element>) OVER([<window_partition_clause>]
                                  [<window_order_clause>]
                                  [<window_frame_clause>])
```

The OVER Clause

```
function_name(<agg_element>) OVER( PARTITION BY value
                                   [<window_order_clause>]
                                   [<window_frame_clause>])
```

Orders Table

OrderKey	CustomerKey	Amount
1	1	100
2	1	200
3	1	300
4	2	300
5	2	400

Window Partitioning

```
SELECT OrderKey, CustomerKey, Amount,
FROM Orders;
```

Result Table

OrderKey	CustomerKey	Amount	CustTotal
1	1	100	600
2	1	200	600
3	1	300	600
4	2	300	700
5	2	400	700

Window Partitioning

```
SELECT OrderKey, CustomerKey, Amount,
       SUM(Amount) OVER(PARTITION BY CustomerKey) AS CustTotal
FROM Orders;
```

Result Table

OrderKey	CustomerKey	Amount	CustTotal
1	1	100	600
2	1	200	600
3	1	300	600
4	2	300	700
5	2	400	700

Window Partitioning

```
SELECT OrderKey, CustomerKey, Amount,
       SUM(Amount) OVER(PARTITION BY CustomerKey) AS CustTotal
FROM Orders;
```

Result Table

OrderKey	CustomerKey	Amount	CustTotal
1	1	100	600
2	1	200	600
3	1	300	600
4	2	300	700
5	2	400	700

Window Partitioning

```
SELECT OrderKey, CustomerKey, Amount,
       SUM(Amount) OVER(PARTITION BY CustomerKey) AS CustTotal
FROM Orders;
```

Result Table

OrderKey	CustomerKey	Amount	CustTotal
1	1	100	600
2	1	200	600
3	1	300	600
4	2	300	700
5	2	400	700

Window Partitioning

```
SELECT OrderKey, CustomerKey, Amount,
       SUM(Amount) OVER(PARTITION BY CustomerKey) AS CustTotal
FROM Orders;
```

Result Table

OrderKey	CustomerKey	Amount	Total
1	1	100	1300
2	1	200	1300
3	1	300	1300
4	2	300	1300
5	2	400	1300

Window Partitioning

```
SELECT OrderKey, CustomerKey, Amount,
       SUM(Amount) OVER() AS Total
FROM Orders;
```

The OVER Clause

```
function_name(<agg_element>) OVER( PARTITION BY value
                                   ORDER BY value ASC | DESC
                                   [<window_frame_clause>])
```

The OVER Clause

```
function_name(<agg_element>) OVER( PARTITION BY value
                                   ORDER BY value ASC | DESC
                                   ROWS BETWEEN <start> AND
                                   <finish>
```

Window framing requires window ordering

Orders Table

Partition start point

Partition end point

OrderKey	CustomerKey	Amount
1	1	100
2	1	200
3	1	300

```
SELECT Order, CustKey, SUM(Amount) AS Amount,
       SUM(Amount) OVER(PARITION BY CustomerKey ORDER BY Amount
                        ROWS BETWEEN <start> AND <finish>)
FROM Orders;
```

Window Frame

```
ROWS BETWEEN N PRECEDING AND N FOLLOWING
```

```
ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
```

```
ROWS BETWEEN CURRENT ROW AND N FOLLOWING
```

Window Clause

```sql
SELECT OrderDate, CustomerKey, Amount,
     SUM(Amount) OVER(PARTITION BY CustomerKey
                      ORDER BY OrderDate
                      ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS RunningVal

     MAX(Amount) OVER(PARTITION BY CustomerKey
                      ORDER BY OrderDate
                      ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS MaxValue
FROM Orders;
```

Window Clause

```
SELECT OrderDate, CustomerKey, Amount,
     SUM(Amount) OVER W1 AS RunningVal

     MAX(Amount) OVER W1 AS MaxValue
FROM Orders

WINDOW W1 AS (PARTITION BY CustomerKey
              ORDER BY OrderDate
              ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) ;
```

Ranking Window Functions

Ranking Window Functions

ROW_NUMBER()

RANK()

DENSE_RANK()

NTILE()

Ranking window functions require ordered data.

Accepted Clauses

ORDER BY

PARTITION BY

Ranking Window Functions

ROW_NUMBER()

Orders Table

OrderKey	CustomerKey	Amount
1	1	100
2	1	300
3	1	300
4	2	300
5	2	400

ROW_NUMBER()

```
SELECT OrderKey, CustomerKey, Amount,
FROM Orders;
```

Result Table

OrderKey	CustomerKey	Amount	Row_No
1	1	100	1
2	1	300	2
3	1	300	3
4	2	300	1
5	2	400	2

ROW_NUMBER()

```
SELECT OrderKey, CustomerKey, Amount,
    ROW_NUMBER() OVER(PARTITION BY CustomerKey ORDER BY Amount) AS Row_No
FROM Orders;
```

Result Table

OrderKey	CustomerKey	Amount	Row_No
1	1	100	1
2	1	300	2
3	1	300	3
4	2	300	1
5	2	400	2

ROW_NUMBER()

```
SELECT OrderKey, CustomerKey, Amount,
    ROW_NUMBER() OVER(PARTITION BY CustomerKey ORDER BY Amount) AS Row_No
FROM Orders;
```

Result Table

OrderKey	CustomerKey	Amount	Row_No
1	1	100	1
2	1	300	2
3	1	300	3
4	2	300	1
5	2	400	2

Nondeterministic

Result Table

OrderKey	CustomerKey	Amount	Row_No
1	1	100	1
2	1	300	2
3	1	300	3
4	2	300	1
5	2	400	2

ROW_NUMBER()

```
SELECT OrderKey, CustomerKey, Amount,
   ROW_NUMBER() OVER(PARTITION BY CustomerKey ORDER BY Amount, OrderKey) AS Row_No
FROM Orders;
```

Ranking Window Functions

ROW_NUMBER()

RANK()

DENSE_RANK()

Orders Table

OrderKey	CustomerKey	Amount
1	1	100
2	1	200
3	1	300
4	2	300
5	2	400

RANK() and DENSE_RANK()

```
SELECT OrderKey, CustomerKey, Amount,
FROM Orders;
```

Result Table

OrderKey	CustomerKey	Amount	Rnk	Dns_Rnk
1	1	100	1	1
2	1	200	2	2
3	1	300	3	3
4	2	300	3	3
5	2	400	5	4

RANK() and DENSE_RANK()

```
SELECT OrderKey, CustomerKey, Amount,
   RANK()       OVER(ORDER BY Amount) AS Rnk
   DENSE_RANK() OVER(ORDER BY Amount) AS Dns_Rnk
FROM Orders;
```

Result Table

OrderKey	CustomerKey	Amount	Rnk	Dns_Rnk
1	1	100	1	1
2	1	200	2	2
3	1	300	3	3
4	2	300	3	3
5	2	400	5	4

RANK() and DENSE_RANK()

```
SELECT OrderKey, CustomerKey, Amount,
   RANK()       OVER(ORDER BY Amount) AS Rnk
   DENSE_RANK() OVER(ORDER BY Amount) AS Dns_Rnk
FROM Orders;
```

Result Table

OrderKey	CustomerKey	Amount	Rnk	Dns_Rnk
1	1	100	1	1
2	1	200	2	2
3	1	300	3	3
4	2	300	3	3
5	2	400	5 = 4 + 1	4

RANK() and DENSE_RANK()

```
SELECT OrderKey, CustomerKey, Amount,
   RANK()       OVER(ORDER BY Amount) AS Rnk
   DENSE_RANK() OVER(ORDER BY Amount) AS Dns_Rnk
FROM Orders;
```

Result Table

OrderKey	CustomerKey	Amount	Rnk	Dns_Rnk
1	1	100	1	1
2	1	200	2	2
3	1	300	3	3
4	2	300	3	3
5	2	400	5 = 4 + 1	4 = 3 + 1

RANK() and DENSE_RANK()

```
SELECT OrderKey, CustomerKey, Amount,
   RANK()       OVER(ORDER BY Amount) AS Rnk
   DENSE_RANK() OVER(ORDER BY Amount) AS Dns_Rnk
FROM Orders;
```

Result Table

OrderKey	CustomerKey	Amount	Rnk	Dns_Rnk
1	1	100	1	1
2	1	200	2	2
3	1	300	3	3
4	2	300	3	3
5	2	400	5 = 4 + 1	4 = 3 + 1

RANK() and DENSE_RANK()

```
SELECT OrderKey, CustomerKey, Amount,
   RANK()       OVER(ORDER BY Amount) AS Rnk
   DENSE_RANK() OVER(ORDER BY Amount) AS Dns_Rnk
FROM Orders;
```

Ranking Window Functions

ROW_NUMBER()

RANK()

DENSE_RANK()

NTILE()

Orders Table

OrderKey	CustomerKey	Amount
1	1	100
2	1	300
3	1	300
4	2	300

NTILE()

```
SELECT OrderKey, CustomerKey, Amount,
FROM Orders;
```

Result Table

OrderKey	CustomerKey	Amount	Group
1	1	100	1
2	1	300	1
3	1	300	2
4	2	300	2

NTILE()

```
SELECT OrderKey, CustomerKey, Amount,
   NTILE(2) OVER(ORDER BY Amount) AS Group
FROM Orders;
```

Result Table

OrderKey	CustomerKey	Amount	Group
1	1	100	1
2	1	300	1
3	1	300	2
4	2	300	2

NTILE()

```
SELECT OrderKey, CustomerKey, Amount,
   NTILE(2) OVER(ORDER BY Amount) AS Group
FROM Orders;
```

Demo

ROW_NUMBER()

- Create a list with the first order placed by each customer

Demo

Retrieve the most expensive products in each category based on list price

Demo

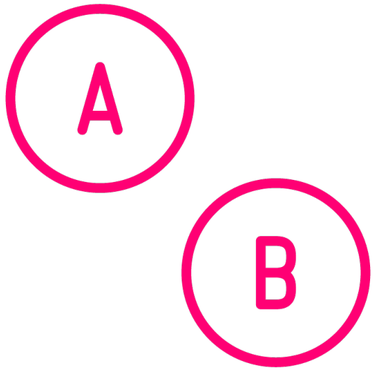
NTILE()

Window Aggregate Functions

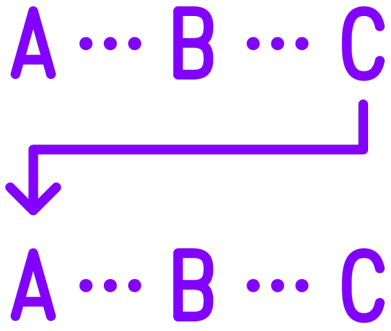
Functions

Grouped aggregate	Window aggregate
GROUP BY	OVER

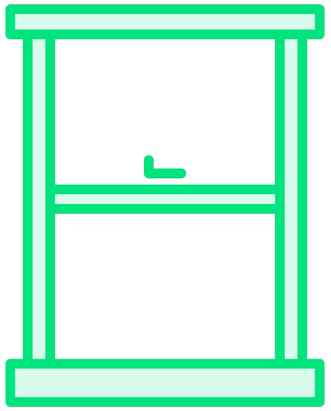
Supported Clauses



Partitioning



Ordering



Framing

Result Table

OrderKey	CustomerKey	Amount	RunTotal
1	1	100	100
2	1	200	300
3	1	300	600
4	2	300	300
5	2	400	700

Window Partitioning

```
SELECT OrderKey, CustomerKey, Amount,
       SUM(Amount) OVER(PARTITION BY CustomerKey ORDER BY OrderKey
                        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS RunTotal
FROM Orders;
```

Result Table

OrderKey	CustomerKey	Amount	RunTotal
1	1	100	100
2	1	200	300
3	1	300	600
4	2	300	300
5	2	400	700

Window Ordering

```
SELECT OrderKey, CustomerKey, Amount,
       SUM(Amount) OVER(PARTITION BY CustomerKey ORDER BY OrderKey
                        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS RunTotal
FROM Orders;
```

Result Table

OrderKey	CustomerKey	Amount	RunTotal
1	1	100	100
2	1	200	300
3	1	300	600
4	2	300	300
5	2	400	700

Window Framing

```
SELECT OrderKey, CustomerKey, Amount,
       SUM(Amount) OVER(PARTITION BY CustomerKey ORDER BY OrderKey
                        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS RunTotal
FROM Orders;
```

Nesting Group Functions within Window Functions

CustomerKey	CustTotal	Pct
1	100	10
2	200	20
3	300	30
4	400	40

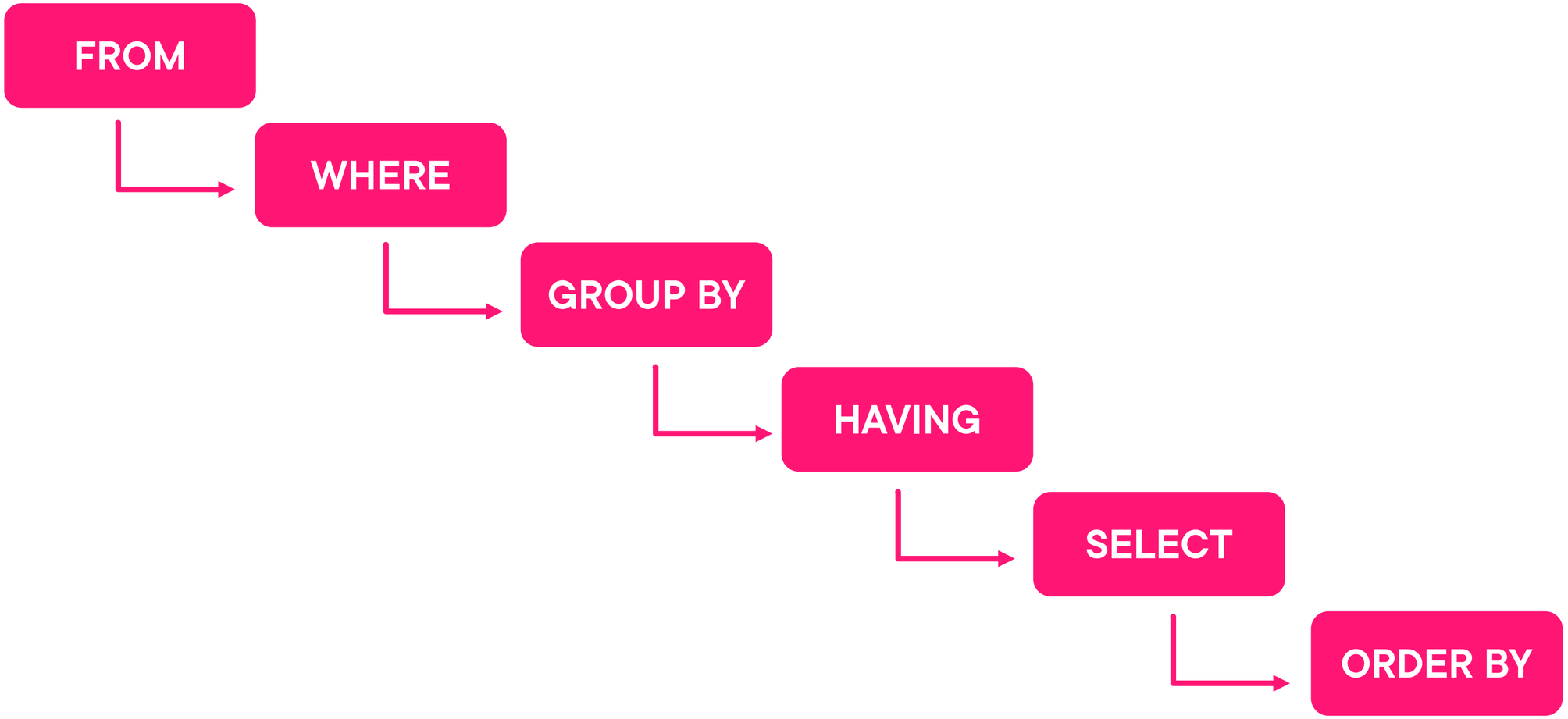
```sql
SELECT CustomerKey,
       SUM(Amount) AS CustTotal,
       SUM(Amount) / SUM(SUM(Amount)) OVER() * 100 AS Pct
FROM Orders
GROUP BY CustomerKey;
```

Nesting Group Functions within Window Functions

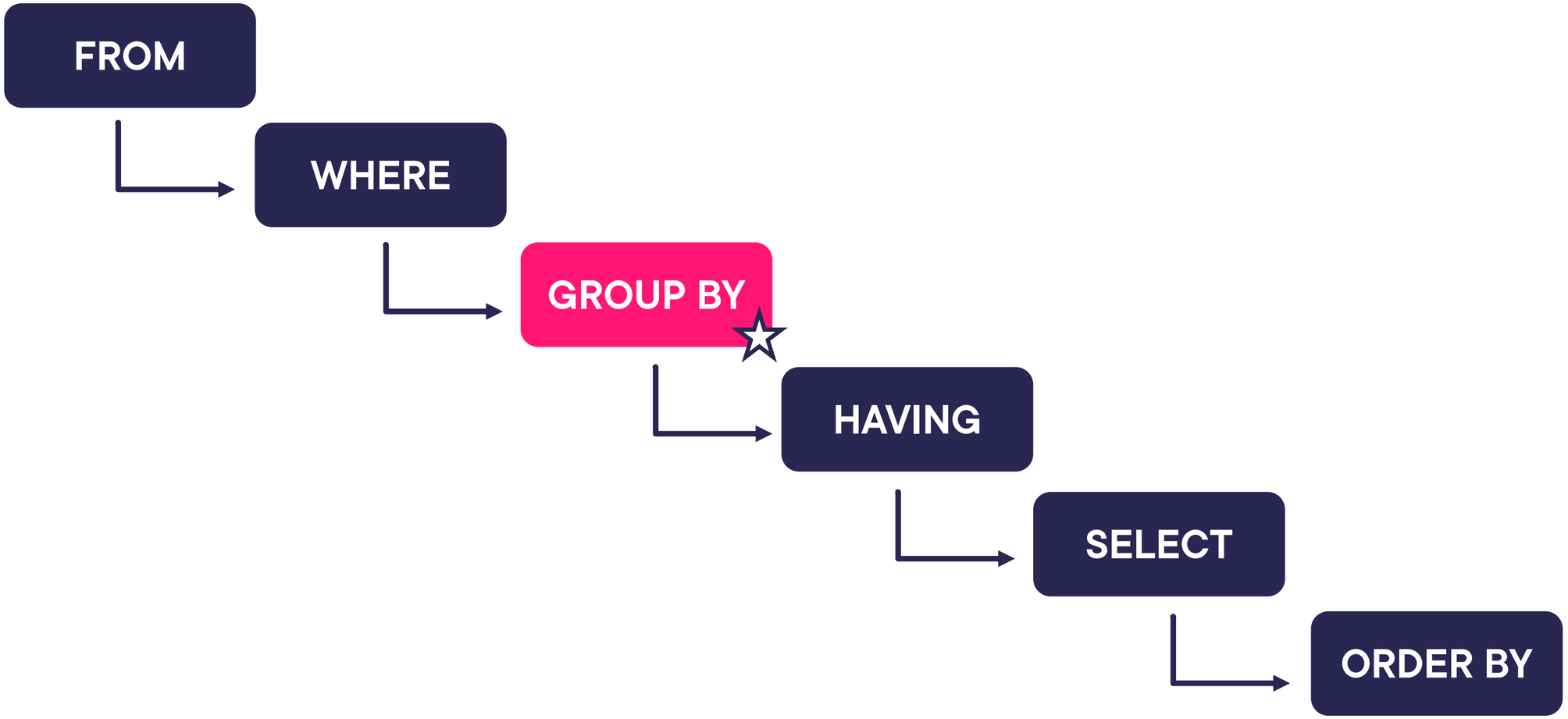
CustomerKey	CustTotal	Pct
1	100	10
2	200	20
3	300	30
4	400	40

```
SELECT CustomerKey,
       SUM(Amount) AS CustTotal,
       SUM(Amount) / SUM(SUM(Amount)) OVER() * 100 AS Pct
FROM Orders
GROUP BY CustomerKey;
```

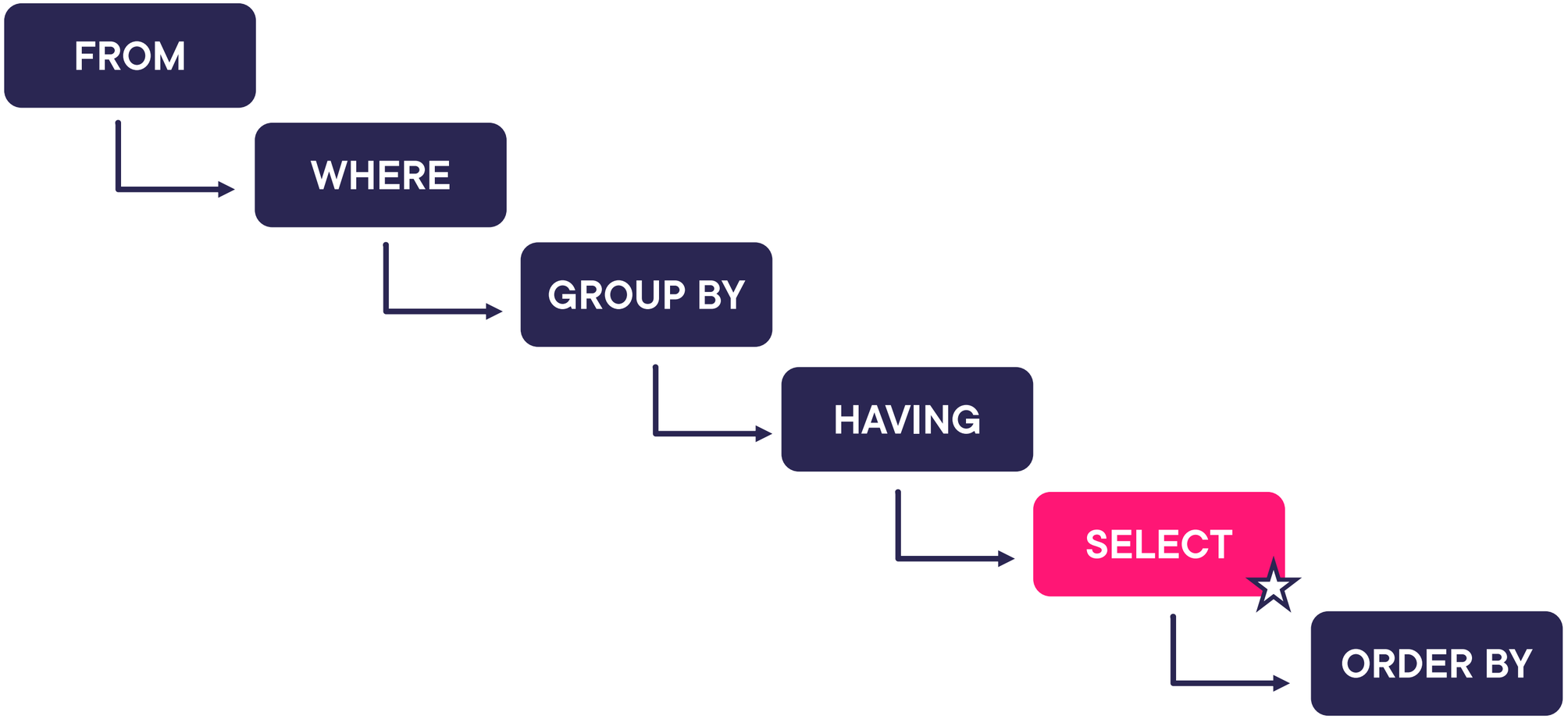
Logical Processing Order



Logical Processing Order



Logical Processing Order



Demo

Aggregate window functions

- Retrieve a list of products along with their list price and the average price within their subcategory
- Calculate grand total and running total
- Display previous and next order values along with the current order value

Offset Window Functions

Offset Functions

LAG()

LEAD()

Offset Functions

```
function_name(<expression>, <offset>, <default>) OVER ()
```

Offset Functions

```
function_name(<expression>, <offset>, <default>) OVER ()
```

Offset Functions

```
function_name(<expression>, <offset>, <default>) OVER ()
```

Result Table

OrderKey	CustomerKey	Amount	PrevVal	NextVal
1	1	100	0	200
2	1	200	100	300
3	1	300	200	300
4	2	300	300	400
5	2	400	300	0

LAG() and LEAD()

```
SELECT OrderKey, CustomerKey, Amount,
   LAG(Amount, 1, 0)  OVER(PARTITON BY CustomerKey ORDER BY Amount) AS PrevVal
   LEAD(Amount, 1, 0) OVER(PARTITON BY CustomerKey ORDER BY Amount) AS NextVal
FROM Orders;
```

Result Table

OrderKey	CustomerKey	Amount	PrevVal	NextVal
1	1	100	0	200
2	1	200	100	300
3	1	300	200	300
4	2	300	300	400
5	2	400	300	0

LAG() and LEAD()

```
SELECT OrderKey, CustomerKey, Amount,
    LAG(Amount, 1, 0)  OVER(PARTITON BY CustomerKey ORDER BY Amount) AS PrevVal
    LEAD(Amount, 1, 0) OVER(PARTITON BY CustomerKey ORDER BY Amount) AS NextVal
FROM Orders;
```

Result Table

OrderKey	CustomerKey	Amount	PrevVal	NextVal
1	1	100	0	200
2	1	200	100	300
3	1	300	200	300
4	2	300	300	400
5	2	400	300	0

LAG() and LEAD()

```
SELECT OrderKey, CustomerKey, Amount,
    LAG(Amount, 1, 0)  OVER(PARTITON BY CustomerKey ORDER BY Amount) AS PrevVal
    LEAD(Amount, 1, 0) OVER(PARTITON BY CustomerKey ORDER BY Amount) AS NextVal
FROM Orders;
```

Offset Functions

LAG()

LEAD()

FIRST_VALUE()

LAST_VALUE()

Demo

Offset functions

- Display previous and next order values along with the current order value

Summary

Window functions

OVER clause

- Partitioning
- Ordering
- Framing

Ranking functions

Aggregate functions

Offset functions

Thank you!

linkedin.com/in/mihaela-danci/