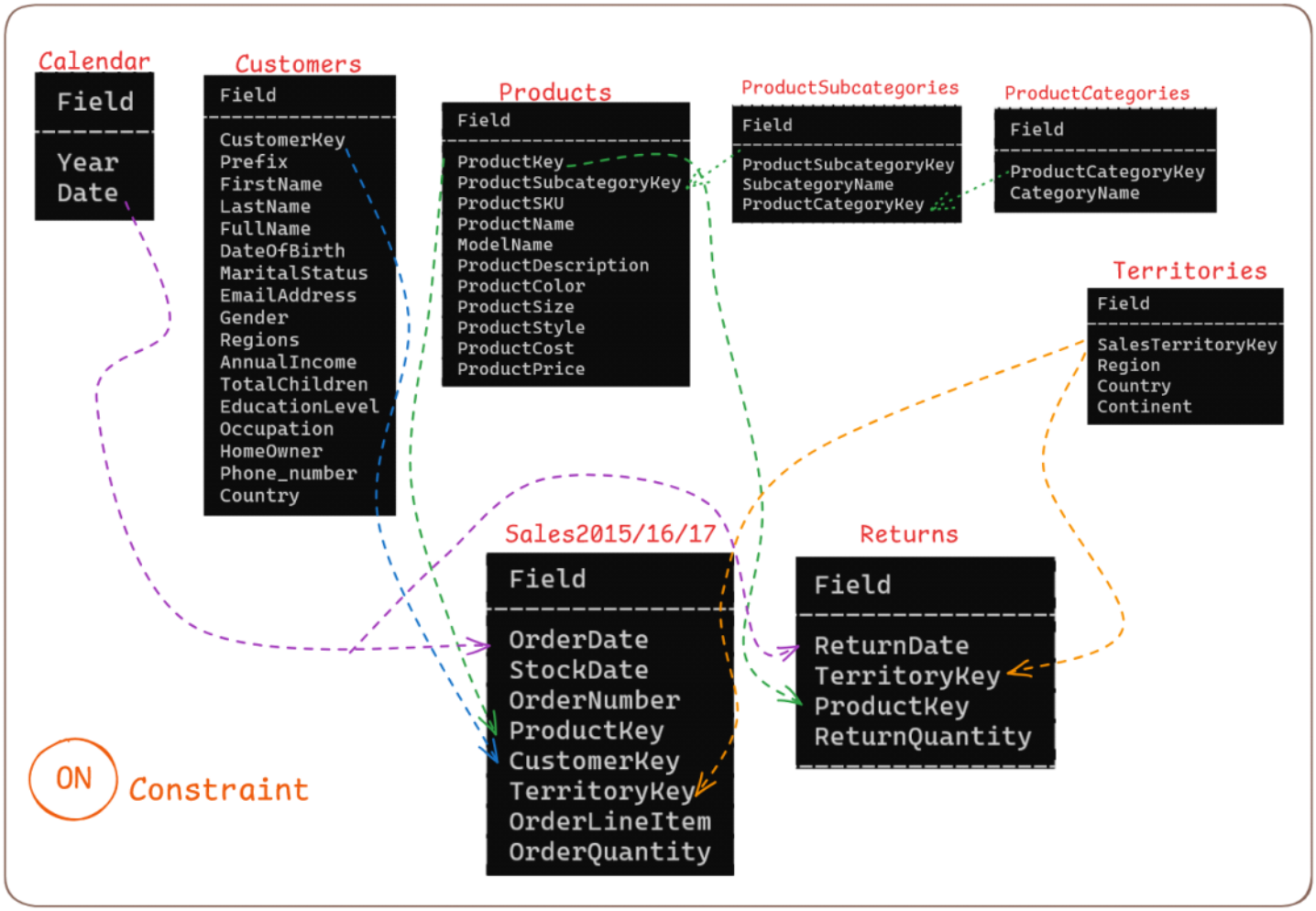
CTEs & Window Functions

🎯 Session Objectives:

- ✅ Understand Common Table Expressions (CTEs) and why we use them
- ✅ Understand what window functions are and when to use them.
- ✅ Break down and apply the syntax of common window functions like ROW_NUMBER(), SUM(), AVG(), etc.
- ✅ Differentiate window functions from regular aggregate functions.

CTE SYNTAX:

```
WITH CTE_NAME AS (
  SELECT column1, column2 FROM table_name WHERE condition
)
SELECT * FROM CTE_NAME;
```



```
-- Challenge : ProductPrice > AvgProductPrice within their Subcategory
```

```
SELECT * FROM Products;

With AvgProductSubcategory AS (
    SELECT
        ProductSubcategoryKey,
        ROUND(AVG(ProductPrice),0) AS AvgPrice
    FROM Products
    GROUP BY 1
)
SELECT * FROM AvgProductSubcategory;
```

ProductSubcategoryKey	AvgPrice
31	34
23	9
19	9
21	51
14	672
12	644
2	1530
1	1637
10	184
11	87
4	69
17	221
22	64
35	125
34	25
36	22

ProductKey	ProductSubcategoryKey	ProductName	ProductPrice
344	1	Mountain-100 Silver, 38	3399.99
345	1	Mountain-100 Silver, 42	3399.99
346	1	Mountain-100 Silver, 44	3399.99
347	1	Mountain-100 Silver, 48	3399.99
348	1	Mountain-100 Black, 38	3374.99
349	1	Mountain-100 Black, 42	3374.99
350	1	Mountain-100 Black, 44	3374.99
351	1	Mountain-100 Black, 48	3374.99
352	1	Mountain-200 Silver, 38	2071.4196
354	1	Mountain-200 Silver, 42	2071.4196
356	1	Mountain-200 Silver, 46	2071.4196
358	1	Mountain-200 Black, 38	2049.0982
360	1	Mountain-200 Black, 42	2049.0982
362	1	Mountain-200 Black, 46	2049.0982
364	1	Mountain-300 Black, 38	1079.99
365	1	Mountain-300 Black, 40	1079.99
366	1	Mountain-300 Black, 44	1079.99
367	1	Mountain-300 Black, 48	1079.99
587	1	Mountain-400-W Silver...	769.49

> 1637

```
SELECT
    ProductKey,
    ProductSubcategoryKey,
    ProductName,
    ProductPrice
FROM Products WHERE ProductSubcategoryKey = 1;
```

```
With AvgProductSubcategory AS (
    SELECT
        ProductSubcategoryKey,
        ROUND(AVG(ProductPrice),0) AS AvgPrice
    FROM Products
    GROUP BY 1
)
SELECT
    p.ProductSubcategoryKey
    ProductKey,
    ProductName,
    ProductPrice,
    AvgPrice
FROM Products p
JOIN AvgProductSubcategory a
ON a.ProductSubcategoryKey = p.ProductSubcategoryKey
WHERE ProductPrice > AvgPrice
ORDER BY p.ProductSubcategoryKey;
```

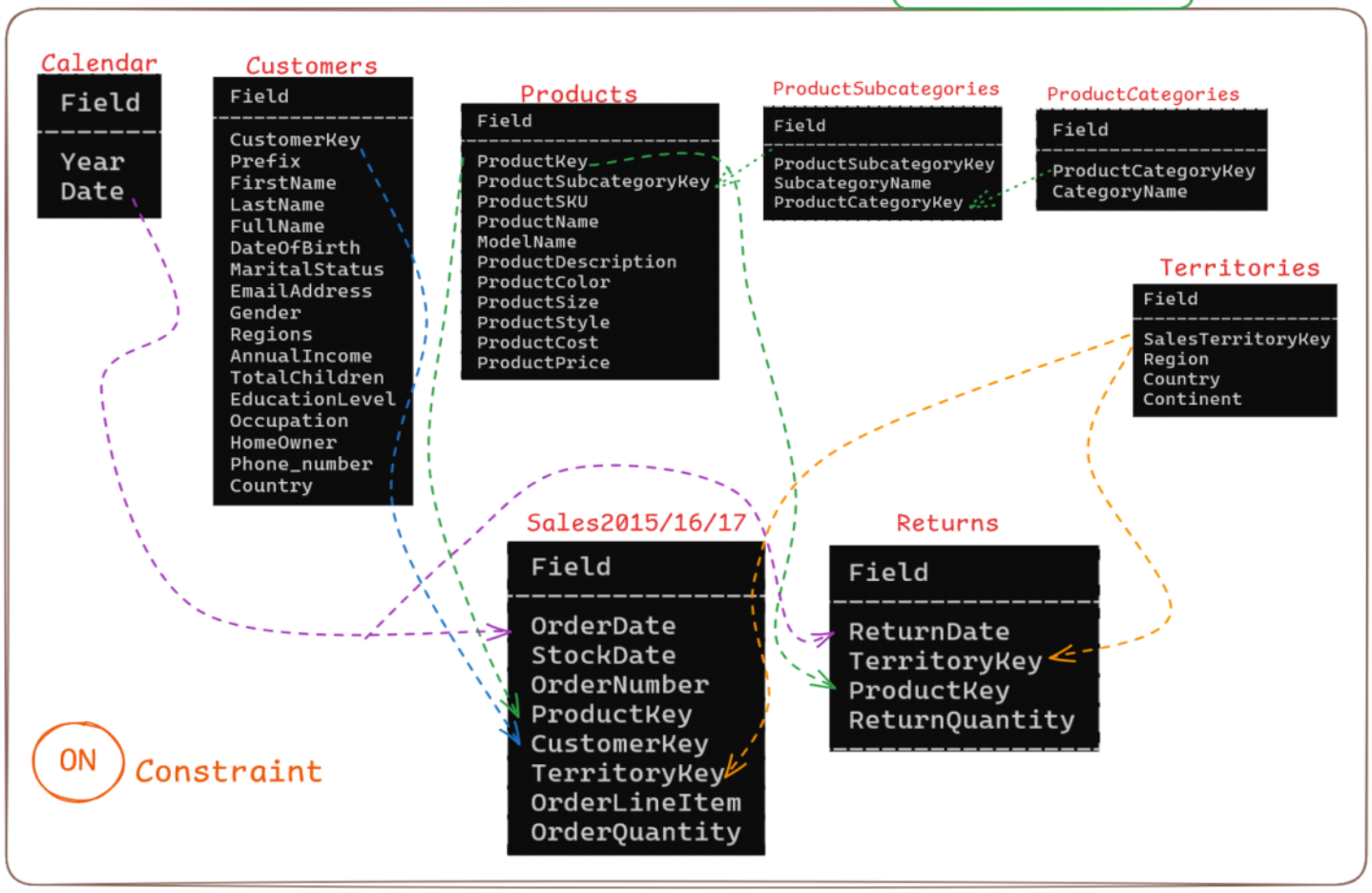
ProductKey	ProductName	ProductPrice	AvgPrice
1	Mountain-200 Silver, 42	2071.4196	1637
1	Mountain-200 Silver, 46	2071.4196	1637
1	Mountain-200 Black, 38	2049.0982	1637
1	Mountain-200 Black, 42	2049.0982	1637
1	Mountain-200 Black, 46	2049.0982	1637
2	Road-150 Red, 62	3578.27	1530
2	Road-150 Red, 44	3578.27	1530
2	Road-150 Red, 48	3578.27	1530
2	Road-150 Red, 52	3578.27	1530
2	Road-150 Red, 56	3578.27	1530

ProductKey	ProductName	ProductPrice	AvgPrice
3	Touring-1000 Yellow, 54	2384.07	1425
3	Touring-1000 Yellow, 60	2384.07	1425
3	Touring-1000 Blue, 46	2384.07	1425
3	Touring-1000 Blue, 50	2384.07	1425
3	Touring-1000 Blue, 54	2384.07	1425
3	Touring-1000 Blue, 60	2384.07	1425
4	HL Mountain Handlebars	109.3364	69
4	HL Road Handlebars	109.3364	69
4	HL Touring Handlebars	91.57	69
5	ML Bottom Bracket	101.24	92
5	HL Bottom Bracket	121.49	92
6	Rear Brakes	106.5	106
6	Front Brakes	106.5	106
7	Chain	20.24	20
8	HL Crankset	404.99	279
9	Rear Derailleur	121.46	106
10	HL Fork	229.49	184

Multiple CTEs

CategoryName | TotalReturns | TotalRevenue

Snowflake Schema's



CategoryReturns

```
WITH CategoryReturns AS(
    SELECT
        pc.CategoryName,
        SUM(ReturnQuantity) AS TotalReturns
    FROM `Product-Categories` pc
    JOIN `Product-Subcategories` ps
    ON pc.ProductCategoryKey = ps.ProductCategoryKey
    JOIN Products p
    ON ps.ProductSubcategoryKey = p.ProductSubcategoryKey
    JOIN Returns r
    ON p.ProductKey = r.productKey
    GROUP BY 1
)
SELECT * FROM CategoryReturns;
```

CategoryName	TotalReturns
Bikes	429
Accessories	1130
Clothing	269

CategoryRevenue

```
WITH CategoryRevenue AS(
    SELECT
        pc.CategoryName,
        ROUND(SUM(p.ProductPrice * s.OrderQuantity),0) AS TotalRevenue
    FROM `Product-Categories` pc
    JOIN `Product-Subcategories` ps
    ON pc.ProductCategoryKey = ps.ProductCategoryKey
    JOIN Products p
    ON ps.ProductSubcategoryKey = p.ProductSubcategoryKey
    JOIN `Sales-2017` s
    ON p.ProductKey = s.productKey
    GROUP BY 1
)
SELECT * FROM CategoryRevenue;
```

CategoryName	TotalRevenue
Accessories	507331
Bikes	8468855
Clothing	209264

```sql
-- Finalizing the above 2 separate CTE.

WITH CategoryReturns AS(
    SELECT
        pc.CategoryName,
        SUM(ReturnQuantity) AS TotalReturns
    FROM `Product-Categories` pc
    JOIN `Product-Subcategories` ps
    ON pc.ProductCategoryKey = ps.ProductCategoryKey
    JOIN Products p
    ON ps.ProductSubcategoryKey = p.ProductSubcategoryKey
    JOIN Returns r
    ON p.ProductKey = r.productKey
    GROUP BY 1
),
CategorySales AS(
    SELECT
        pc.CategoryName,
        ROUND(SUM(p.ProductPrice * s.OrderQuantity),0) AS TotalRevenue
    FROM `Product-Categories` pc
    JOIN `Product-Subcategories` ps
    ON pc.ProductCategoryKey = ps.ProductCategoryKey
    JOIN Products p
    ON ps.ProductSubcategoryKey = p.ProductSubcategoryKey
    JOIN `Sales-2017` s
    ON p.ProductKey = s.productKey
    GROUP BY 1
)
SELECT
    cr.CategoryName,
    TotalReturns,
    TotalRevenue
FROM CategoryReturns cr
JOIN CategorySales cs
ON cr.CategoryName = cs.CategoryName
ORDER BY TotalRevenue DESC;
```

CategoryName	TotalReturns	TotalRevenue
Bikes	429	8468855
Accessories	1130	507331
Clothing	269	209264

AllSales -> Sales2015 U Sales2016 U Sales2017

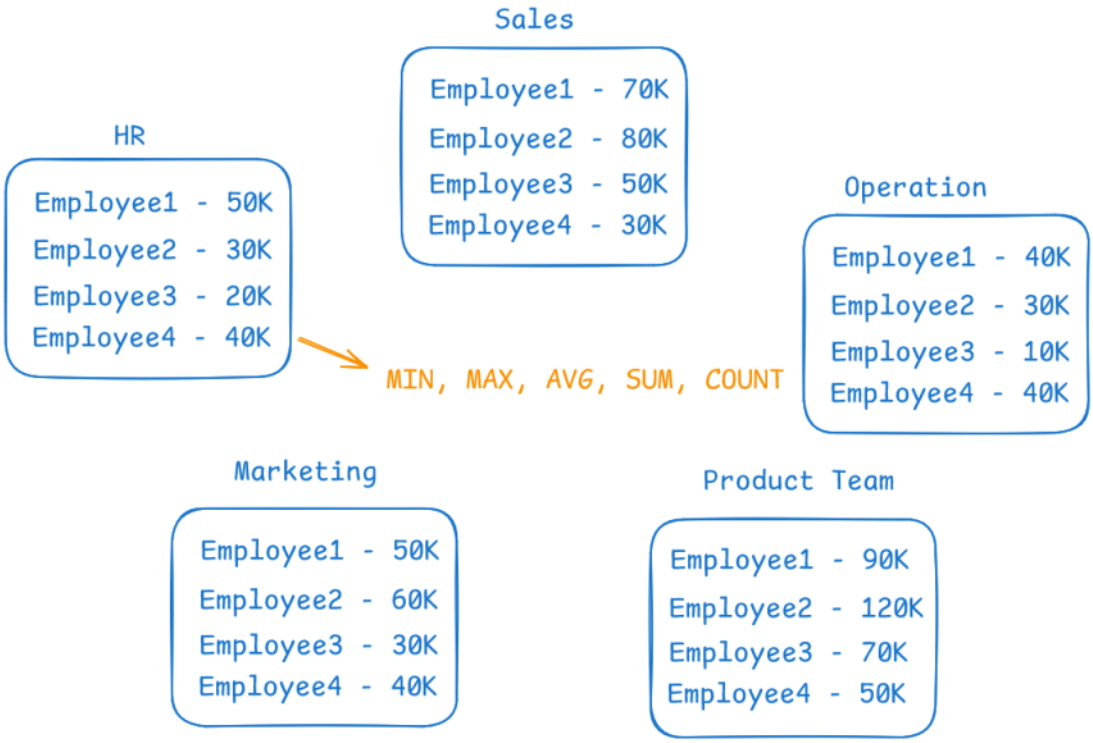
```
WITH AllSales AS(
    SELECT * FROM `Sales-2015`
    UNION ALL
    SELECT * FROM `Sales-2016`
    UNION ALL
    SELECT * FROM `Sales-2017`
),
CategoryReturns AS(
    SELECT
        pc.CategoryName,
        SUM(ReturnQuantity) AS TotalReturns
    FROM `Product-Categories` pc
    JOIN `Product-Subcategories` ps
    ON pc.ProductCategoryKey = ps.ProductCategoryKey
    JOIN Products p
    ON ps.ProductSubcategoryKey = p.ProductSubcategoryKey
    JOIN Returns r
    ON p.ProductKey = r.productKey
    GROUP BY 1
),
CategorySales AS(
    SELECT
        pc.CategoryName,
        ROUND(SUM(p.ProductPrice * s.OrderQuantity),0) AS TotalRevenue
    FROM `Product-Categories` pc
    JOIN `Product-Subcategories` ps
    ON pc.ProductCategoryKey = ps.ProductCategoryKey
    JOIN Products p
    ON ps.ProductSubcategoryKey = p.ProductSubcategoryKey
    JOIN AllSales s
    ON p.ProductKey = s.productKey
    GROUP BY 1
)
SELECT
    cr.CategoryName,
    TotalReturns,
    TotalRevenue
FROM CategoryReturns cr
JOIN CategorySales cs
ON cr.CategoryName = cs.CategoryName
ORDER BY TotalRevenue DESC;
```

CategoryName	TotalReturns	TotalRevenue
Bikes	429	23642495
Accessories	1130	906673
Clothing	269	365419

Window Functions

Syntax:

```
SELECT
  window_function(...) OVER (
    PARTITION BY column_name
    ORDER BY column_name
    ROWS/RANGE ...
  ) AS result_column
FROM table_name;
```



HR - 140K
Sales - 230K
Marketing - 180K
Operation - 120K
Product - 330K

With Group By & Aggregation

SaleID	Salesperson	SaleAmount	SaleDate
1	Alice	300	2023-01-01
2	Bob	150	2023-01-02
3	Alice	200	2023-01-03
4	Charlie	250	2023-01-04
5	Bob	300	2023-01-05
6	Alice	100	2023-01-06
7	Charlie	350	2023-01-07
8	Alice	450	2023-01-08
9	Bob	200	2023-01-09
10	Charlie	400	2023-01-10
11	Alice	150	2023-01-11
12	Bob	250	2023-01-12
13	Charlie	300	2023-01-13
14	Alice	350	2023-01-14
15	Bob	100	2023-01-15

```
172 •  SELECT DISTINCT SalesPerson FROM Sale;
173
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

SalesPerson
Alice
Bob
Charlie

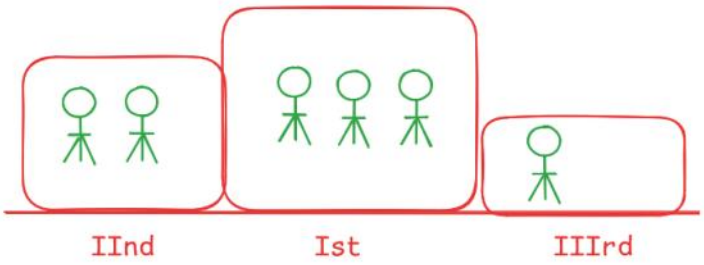
```
-- Challenge 1 : Find the Cumulative TotalSales by SalesPerson

SELECT
    *,
    SUM(SaleAmount) OVER(
        PARTITION BY SalesPerson
        ORDER BY SaleDate
    ) AS CumulativeSalesPerPerson
FROM Sale;
```

SaleID	Salesperson	SaleAmount	SaleDate	CumulativeSalesPerPerson
1	Alice	300	2023-01-01	300
3	Alice	200	2023-01-03	500
6	Alice	100	2023-01-06	600
8	Alice	450	2023-01-08	1050
11	Alice	150	2023-01-11	1200
14	Alice	350	2023-01-14	1550
2	Bob	150	2023-01-02	150
5	Bob	300	2023-01-05	450
9	Bob	200	2023-01-09	650
12	Bob	250	2023-01-12	900
15	Bob	100	2023-01-15	1000
4	Charlie	250	2023-01-04	250
7	Charlie	350	2023-01-07	600
10	Charlie	400	2023-01-10	1000
13	Charlie	300	2023-01-13	1300

Rank the SalesPerson by SalesAmount

RANK()
DENSE_RANK()



A - 1	Rank() Vs	A - 1
B - 1		B - 1
C - 1	Dense_Rank()	C - 1
D - 4		D - 2
E - 4		E - 2
F - 6		F - 3
RANK()		DENSE_RANK()

Rank() skips when ties , Dense_Rank() not.....

Percentage
Vs Percentile

Percentage[100]

PersonA - 75
PersonB - 90
PersonC - 85
PersonD - 95

Percentile[High Score] [95]

PersonA - 75
PersonB - 90
PersonC - 85
PersonD - 95

```sql
-- Challenge 2 : Rank the SalesPerson by SaleAmount
SELECT
    *,
    RANK() OVER(
        ORDER BY SaleAmount DESC
    ) AS SalesRank
FROM Sale;
```

SaleID	Salesperson	SaleAmount	SaleDate	SalesRank
8	Alice	450	2023-01-08	1
10	Charlie	400	2023-01-10	2
7	Charlie	350	2023-01-07	3
14	Alice	350	2023-01-14	3
1	Alice	300	2023-01-01	5
5	Bob	300	2023-01-05	5
13	Charlie	300	2023-01-13	5
4	Charlie	250	2023-01-04	8
12	Bob	250	2023-01-12	8
3	Alice	200	2023-01-03	10
9	Bob	200	2023-01-09	10
2	Bob	150	2023-01-02	12
11	Alice	150	2023-01-11	12
6	Alice	100	2023-01-06	14
15	Bob	100	2023-01-15	14

```sql
-- Challenge 2 : Rank the SalesPerson by SaleAmount
SELECT
    *,
    DENSE_RANK() OVER(
        ORDER BY SaleAmount DESC
    ) AS SalesRank
FROM Sale;
```

SaleID	Salesperson	SaleAmount	SaleDate	SalesRank
8	Alice	450	2023-01-08	1
10	Charlie	400	2023-01-10	2
7	Charlie	350	2023-01-07	3
14	Alice	350	2023-01-14	3
1	Alice	300	2023-01-01	4
5	Bob	300	2023-01-05	4
13	Charlie	300	2023-01-13	4
4	Charlie	250	2023-01-04	5
12	Bob	250	2023-01-12	5
3	Alice	200	2023-01-03	6
9	Bob	200	2023-01-09	6
2	Bob	150	2023-01-02	7
11	Alice	150	2023-01-11	7
6	Alice	100	2023-01-06	8
15	Bob	100	2023-01-15	8

3 days Moving Average


```sql
-- Challenge 3 : Find the 3 days Moving Average Sales

SELECT
    *,
    AVG(SaleAmount) OVER(
        ORDER BY SaleDate
        ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING
    ) AS MovingAverage
FROM Sale;
```

SaleID	Salesperson	SaleAmount	SaleDate	MovingAverage
1	Alice	300	2023-01-01	225.0000
2	Bob	150	2023-01-02	216.6667
3	Alice	200	2023-01-03	200.0000
4	Charlie	250	2023-01-04	250.0000
5	Bob	300	2023-01-05	216.6667
6	Alice	100	2023-01-06	250.0000
7	Charlie	350	2023-01-07	300.0000
8	Alice	450	2023-01-08	333.3333
9	Bob	200	2023-01-09	350.0000
10	Charlie	400	2023-01-10	250.0000
11	Alice	150	2023-01-11	266.6667
12	Bob	250	2023-01-12	233.3333
13	Charlie	300	2023-01-13	300.0000
14	Alice	350	2023-01-14	250.0000
15	Bob	100	2023-01-15	225.0000

x+y+z/3

AVG()

pandas -> bfill [Backward Filling] -> Fill Up
-> ffill [forward Filling] -> Fill Down