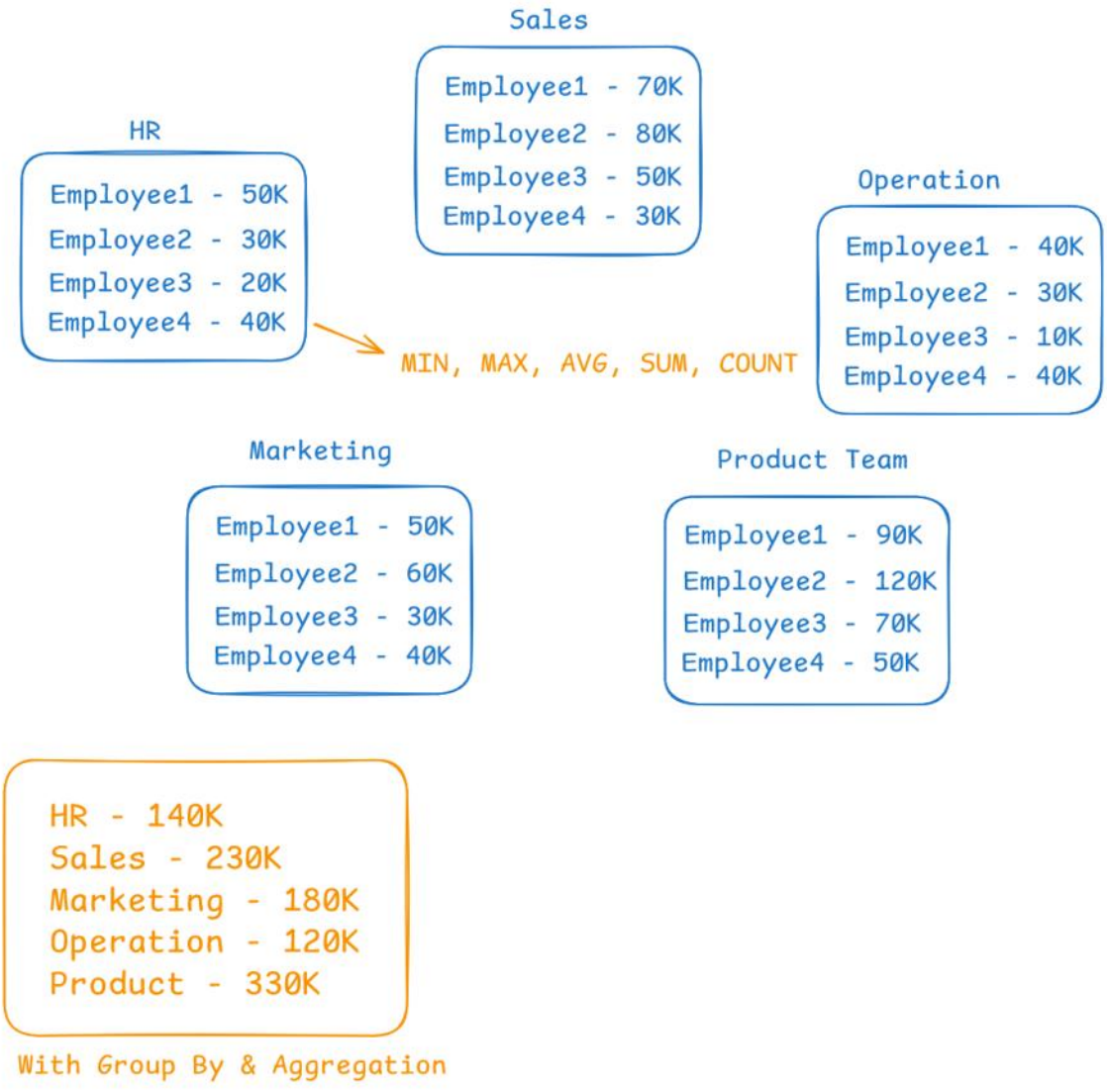
Window Functions-I

Session Goals

- ✅ 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.

Syntax:

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



```sql
-- WINDOW Functions
USE bike_analysis;
ALTER TABLE `session 13 dataset (1)`
RENAME To sales;
DESC sales;
SELECT * FROM sales;
```

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

Find the cumulative total sum by Salesperson

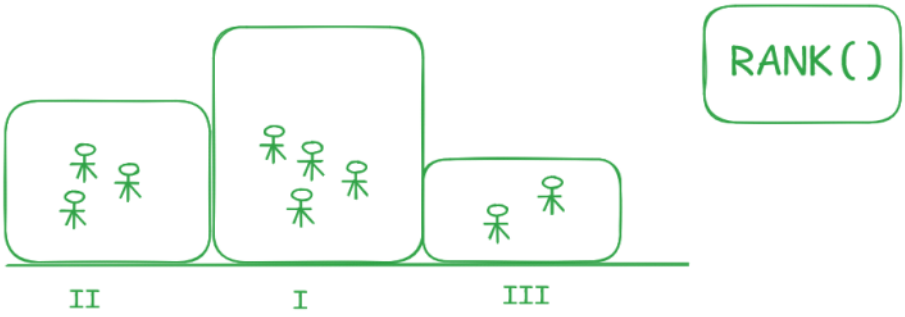
```sql
-- Find the cumulative total sum by Salesperson
SELECT
    *,
    SUM(SaleAmount) OVER(
        PARTITION BY Salesperson
        ORDER BY SaleDate
    ) AS CumulativeSalesPerPerson
FROM sales;
```

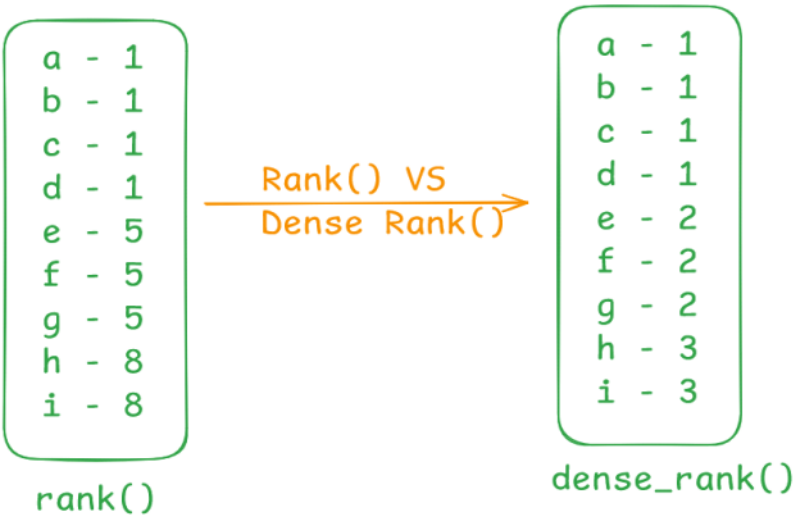
SaleID	Salesperson	SaleAmount	SaleDate	CumulativeSalesPerPerson
1	Alice	300	2023-01-01	300
3	Alice	200	2023-01-03	500
6	Alice	100	2023-01-06	600
8	Alice	450	2023-01-08	1050
11	Alice	150	2023-01-11	1200
14	Alice	350	2023-01-14	1550
2	Bob	150	2023-01-02	150
5	Bob	300	2023-01-05	450
9	Bob	200	2023-01-09	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

```
SELECT
    Salesperson,
    SUM(SaleAmount) AS TotalSales
FROM sales
GROUP BY Salesperson;
```

Salesperson	TotalSales
Alice	1550
Bob	1000
Charlie	1300

Rank the Sales by Sales Amount





Rank skips when ties while dense_rank not

```
SELECT
    *,
    RANK() OVER(
        ORDER BY SaleAmount DESC
) AS SalesRank
FROM sales;
```

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

Memory

```
counter = 1
counter++
```

```
SELECT
    *,
    DENSE_RANK() OVER(
        ORDER BY SaleAmount DESC
) AS SalesRank
FROM sales;
```

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

percentage[100]

```
personA - 75
personB - 80
personC - 90
personD - 95
```

percentile

```
personA - 75
personB - 80
personC - 90
personD - 95
```

```
75/95  = 78.9
80/95  = 84.21
```

3 Days Moving Average


```
-- Finding the 3 Days Moving Average
SELECT
    *,
    AVG(SaleAmount) OVER(
        ORDER BY SaleDate
    ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING
) AS MovingAverage
FROM sales;
```

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

```
-- Finding the 5 Days Moving Average
SELECT
        *,
    AVG(SaleAmount) OVER(
        ORDER BY SaleDate
        ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING
) AS MovingAverage
FROM sales;
```

SaleID	Salesperson	SaleAmount	SaleDate	MovingAverage
1	Alice	300	2023-01-01	216.6667
2	Bob	150	2023-01-02	225.0000
3	Alice	200	2023-01-03	240.0000
4	Charlie	250	2023-01-04	200.0000
5	Bob	300	2023-01-05	240.0000
6	Alice	100	2023-01-06	290.0000
7	Charlie	350	2023-01-07	280.0000
8	Alice	450	2023-01-08	300.0000
9	Bob	200	2023-01-09	310.0000
10	Charlie	400	2023-01-10	290.0000
11	Alice	150	2023-01-11	260.0000
12	Bob	250	2023-01-12	290.0000
13	Charlie	300	2023-01-13	230.0000
14	Alice	350	2023-01-14	250.0000
15	Bob	100	2023-01-15	250.0000

Football Analogy

Aggregate

- We wanted to know the total goals per match

Window Function [Aggregate]

- Here, we need to know each and individual player performance of a match.

Income Difference from AvgIncome

```
081 •  SELECT AVG(AnnualIncome) FROM Customers;
```

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

AVG(AnnualIncome)
57256.3353

CustomerKey	FullName	AnnualIncome	income_difference_from_avg
11000	JON YANG	90000	32743.6647
11001	EUGENE HUANG	60000	2743.6647
11002	RUBEN TORRES	60000	2743.6647
11003	CHRISTY ZHU	NULL	NULL
11004	ELIZABETH JOHNSON	80000	22743.6647
11005	JULIO RUIZ	70000	12743.6647
11007	MARCO MEHTA	60000	2743.6647
11008	ROBIN VERHOFF	60000	2743.6647
11009	SHANNON CARLSON	70000	12743.6647
11010	JACQUELYN SUAREZ	70000	12743.6647
11011	CURTIS LU	60000	2743.6647
11012	LAUREN WALKER	100000	42743.6647
11013	IAN JENKINS	100000	42743.6647
11014	SYDNEY BENNETT	100000	42743.6647
11015	CHLOE YOUNG	NULL	NULL
11016	WYATT HILL	30000	-27256.3353
11017	SHANNON WANG	20000	-37256.3353
11018	CLARENCE RAI	30000	-27256.3353
11019	LUKE LAL	40000	-17256.3353

```
SELECT
    CustomerKey,
    FullName,
    AnnualIncome,
    AnnualIncome - AVG(AnnualIncome) OVER() AS income_difference_from_avg
FROM Customers;
```

Find the maximum income based on Prefix.

```
085 •  SELECT DISTINCT Prefix FROM customers;
```

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

Prefix
MR.
MS.
MRS.
NULL

```
SELECT
    CustomerKey,
    Prefix,
    FullName,
    AnnualIncome,
    MAX(AnnualIncome) OVER(Partition BY Prefix) AS max_income_by_prefix
FROM Customers
WHERE Prefix IS NOT NULL;
```

CustomerKey	Prefix	FullName	AnnualIncome	max_income_by_prefix
11000	MR.	JON YANG	90000	170000
11001	MR.	EUGENE HUANG	60000	170000
11002	MR.	RUBEN TORRES	60000	170000
12054	MR.	LUKE DIAZ	90000	170000
11005	MR.	JULIO RUIZ	70000	170000
11007	MR.	MARCO MEHTA	60000	170000
13098	MR.	MIGUEL RUSSELL	60000	170000
11009	MR.	SHANNON CARLSON	70000	170000
12056	MR.	IAN WARD	120000	170000
11011	MR.	CURTIS LU	60000	170000
12059	MR.	JASON JENKINS	130000	170000
11013	MR.	IAN JENKINS	100000	170000
12060	MR.	ANDREW WEDGE	150000	170000
11016	MR.	WYATT HILL	30000	170000
12061	MR.	BRYCE BROOKS	160000	170000
11018	MR.	CLARENCE RAI	30000	170000
11019	MR.	LUKE LAL	40000	170000
11020	MR.	JORDAN KING	40000	170000
12062	MR.	EDWARD LONG	70000	170000

```
-- Find the max annualIncome based on Gender

SELECT DISTINCT Gender FROM customers;

SELECT
    CustomerKey,
    FullName,
    Gender,
    AnnualIncome,
    MAX(AnnualIncome) OVER(Partition BY Gender) AS max_income_by_prefix
FROM Customers;
```

CustomerKey	FullName	Gender	AnnualIncome	max_income_by_prefix
13098	MIGUEL RUSSELL	M	60000	170000
12 13098	ALEXA WATSON	NA	30000	130000
12446	JASMINE HALL	NA	40000	130000
11025	ALEJANDRO BECK	NA	10000	130000
11965	KATIE SHE	NA	70000	130000
12283	TODD ZENG	NA	20000	130000
12765	ROBYN GILL	NA	20000	130000
11475	CESAR SUBRAM	NA	30000	130000
11643	NAOMI MUNOZ	NA	50000	130000
11234	ANNA GRIFFIN	NA	70000	130000
12276	ALISHA SHAN	NA	30000	130000
12372	DARREN PRASAD	NA	60000	130000
11082	ANGELA BUTLER	NA	130000	130000
12300	ADRIANA GONZALEZ	NA	80000	130000
11623	ANGELA PERRY	NA	50000	130000
12790	BRANDON KUMAR	NA	60000	130000
11526	KATHERINE DIAZ	NA	40000	130000
11035	WENDY DOMINGUEZ	NA	10000	130000
12936	RENEE MORENO	NA	40000	130000