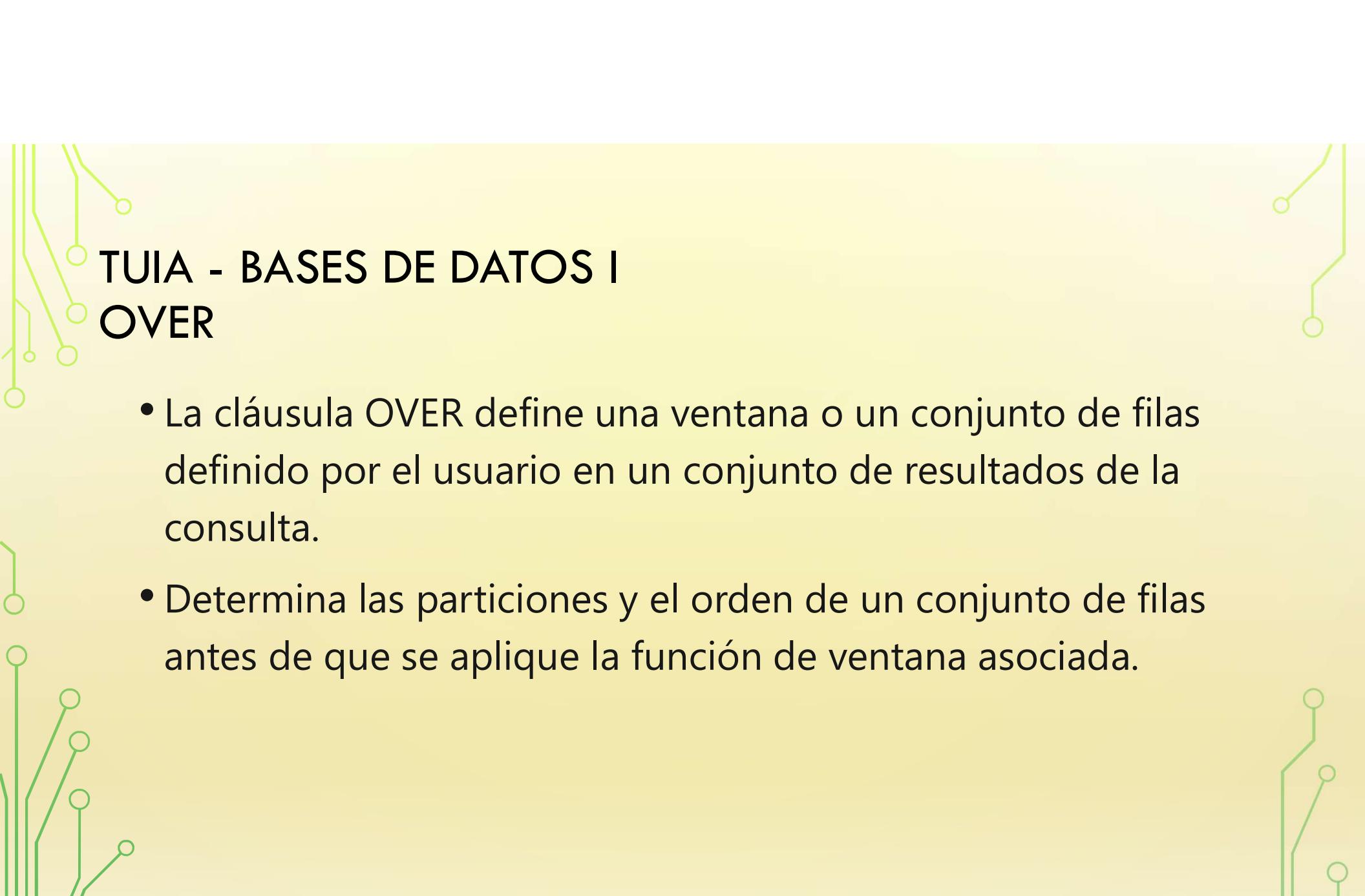




# TUIA

## BASES DE DATOS I – 1 SEM 2025 – COM R1

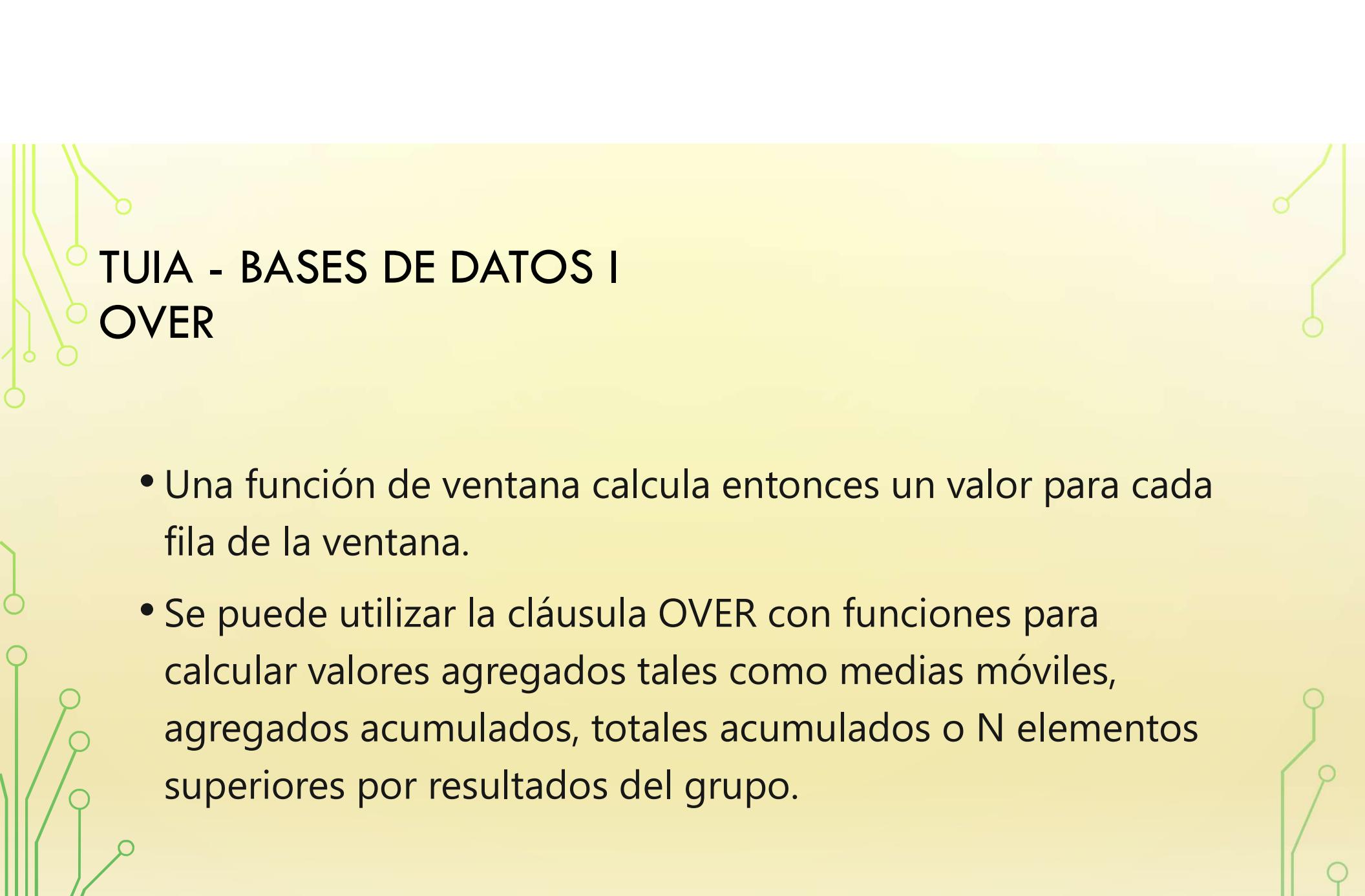
- Docentes:
  - Fernando Roldan
  - Cristian Gallo
  - Lara Della Ceca
- FCEIA-UNR



# TUIA - BASES DE DATOS I

## OVER

- La cláusula OVER define una ventana o un conjunto de filas definido por el usuario en un conjunto de resultados de la consulta.
- Determina las particiones y el orden de un conjunto de filas antes de que se aplique la función de ventana asociada.



# TUIA - BASES DE DATOS I

## OVER

- Una función de ventana calcula entonces un valor para cada fila de la ventana.
- Se puede utilizar la cláusula OVER con funciones para calcular valores agregados tales como medias móviles, agregados acumulados, totales acumulados o N elementos superiores por resultados del grupo.

# TUIA - BASES DE DATOS I

## OVER

Las funciones de ventana (Windows function) podrían tener los siguientes argumentos en su cláusula OVER:

- PARTITION BY, que divide el conjunto de resultados de la consulta en particiones.
- ORDER BY, que define el orden lógico de las filas dentro de cada partición del conjunto de resultados.
- ROWS/RANGE, que limita aún más las filas de la partición especificando los puntos inicial y final. Requiere el argumento ORDER BY y el valor predeterminado es desde el inicio de la partición al elemento actual si se especifica el argumento ORDER BY.

# TUIA - BASES DE DATOS I

## OVER

Las funciones de agregado actúan sobre toda la tabla y devuelven valores utilizando la cláusula GROUP BY.

Sin embargo, a veces es necesario utilizarlas sobre conjuntos más pequeños o ventanas.

# TUIA - BASES DE DATOS I

## OVER

```
USE [SalesLT]
```

```
CREATE TABLE [Orders]  
(  
    orderid INT,  
    orderdate DATE,  
    customerName VARCHAR(100),  
    City VARCHAR(50),  
    amount MONEY  
)
```

# TUIA - BASES DE DATOS I

## OVER

```
INSERT INTO [Orders]

SELECT 1,'01/01/2021','Mohan Gupta','Alwar',10000
UNION ALL
SELECT 2,'02/04/2021','Lucky Ali','Kota',20000
UNION ALL
SELECT 3,'03/02/2021','Raj Kumar','Jaipur',5000
UNION ALL
SELECT 4,'04/02/2021','Jyoti Kumari','Jaipur',15000
UNION ALL
SELECT 5,'05/03/2021','Rahul Gupta','Jaipur',7000
UNION ALL
SELECT 6,'06/04/2021','Mohan Kumar','Alwar',25000
UNION ALL
SELECT 7,'07/02/2021','Kashish Agarwal','Alwar',15000
UNION ALL
SELECT 8,'08/03/2021','Nagar Singh','Kota',2000
UNION ALL
SELECT 9,'09/04/2021','Anil KG','Alwar',1000
```

Go

# TUIA - BASES DE DATOS I

## OVER

Si queremos conocer el monto total de las órdenes a cada ciudad podemos usar GROUP BY y SUM

```
SELECT City AS CustomerCity  
, SUM(amount) AS totalamount  
FROM [Orders]  
GROUP BY city  
ORDER BY city
```

# TUIA - BASES DE DATOS I

## OVER

No podemos utilizar columnas no agregadas (que no estén en el GROUP BY) o en funciones de agregado

```
SELECT City AS CustomerCity,  
customerName,  
SUM(amount) AS totalamount  
FROM [Orders]  
GROUP BY city  
ORDER BY city
```

# TUIA - BASES DE DATOS I

## OVER – PARTITION BY

La cláusula PARTITION BY crea una ventana , realiza la agregación y la muestra.  
También se pueden usar columnas no agregadas en el SELECT

```
SELECT City AS CustomerCity, CustomerName, amount,  
SUM(amount) OVER(PARTITION BY city) TotalOrderAmount  
FROM [Orders]
```

	CustomerCity	CustomerName	amount	TotalOrderAmount
1	Alwar	Mohan Gupta	10000.00	51000,00
2	Alwar	Mohan Kumar	25000.00	51000,00
3	Alwar	Kashish Agarwal	15000.00	51000,00
4	Alwar	Anil KG	1000.00	51000,00
5	Jaipur	Raj Kumar	5000.00	27000,00
6	Jaipur	Jyoti Kumari	15000.00	27000,00
7	Jaipur	Rahul Gupta	7000.00	27000,00
8	Kota	Lucky Ali	20000.00	22000,00
9	Kota	Nagar Singh	2000.00	22000,00

# TUIA - BASES DE DATOS I

## OVER – PARTITION BY

Podemos utilizar otras funciones de agregado:

```
SELECT City AS CustomerCity, CustomerName, amount,  
SUM(amount) OVER(PARTITION BY city) TotalOrderAmount,  
Avg(amount) OVER(PARTITION BY city) AvgOrderAmount,  
Min(amount) OVER(PARTITION BY city) MinOrderAmount,  
MAX(amount) OVER(PARTITION BY city) MaxOrderAmount  
FROM [Orders]
```

# TUIA - BASES DE DATOS I

## OVER – PARTITION BY – ROW\_NUMBER()

Podemos utilizar la función ROW\_Number() para opciones de paginado:

```
SELECT City AS CustomerCity, CustomerName, amount,  
ROW_NUMBER() OVER(PARTITION BY city ORDER BY amount DESC) AS [Row Number]  
FROM [Orders]
```

# TUIA - BASES DE DATOS I

## OVER – PARTITION BY – ROW\_NUMBER()

Podemos utilizar CURRENT ROW y FOLLOWING para sumas acumulativas parciales:

```
SELECT City AS CustomerCity, CustomerName, amount,  
ROW_NUMBER() OVER(PARTITION BY city ORDER BY amount DESC) AS [Row Number],  
SUM(amount) OVER(PARTITION BY city ORDER BY amount DESC ROWS BETWEEN  
CURRENT ROW AND 1 FOLLOWING) AS CumulativeSUM  
FROM [Orders]
```

# TUIA - BASES DE DATOS I

## OVER – PARTITION BY – ROW\_NUMBER()

Podemos utilizar CURRENT ROW y PRECEDING para sumas acumulativas :

```
SELECT City AS CustomerCity, CustomerName, amount,  
ROW_NUMBER() OVER(PARTITION BY city ORDER BY amount DESC) AS [Row Number],  
SUM(amount) OVER(PARTITION BY city ORDER BY amount DESC  
ROWS UNBOUNDED PRECEDING) AS CumulativeSUM  
FROM [Orders]
```

# TUIA - BASES DE DATOS I

## OVER – PARTITION BY – ROW\_NUMBER()

Comparativa entre GROUP BY vs PARTITION BY:

GROUP BY	PARTITION BY
Devuelve una fila por grupo despues de agrupar los valores	Devuelve todas las filas del SELECT con los valores agregados.
No podemos usar columnas no agregadas en el SELECT.	Podemos usar columnas no agregadas en el SELECT.
Se filtra con HAVING	La clausula PARTITION puede tener predicados adicionales en el WHERE
GROUP BY se usa para agregados regulares	PARTITION BY se usa en agregados de funciones de ventana
No podemos utilizarlo para calcular numeros de fila o ranks.	Podemos utilizarlo para calcular numeros de fila o ranks.