UNIVERSIDAD PERUANA LOS ANDES FACULTAD DE INGENIERÍA

Asignatura: Base de Datos II

Docente: Dr. Raúl Enrique Fernández Bejarano

Alumno: SORIANO TIMOTEO Joel Kevin

Ciclo: V

huancayo-Perú-2025

¿Qué es la Cláusula OVER en SQL?

La cláusula **OVER** permite definir una "ventana" o rango específico de filas dentro del resultado de una consulta, sobre el cual se ejecuta una **Función de Ventana** (muchas veces una función de agregación).
A diferencia de **GROUP BY**, que combina varias filas en una sola fila resumen, **OVER** calcula valores agregados **sin eliminar las filas originales**.
Esto provoca que el resultado calculado (por ejemplo, suma, promedio o máximo) aparezca **repetido en cada fila perteneciente a la misma ventana**.

Una forma sencilla de entenderlo es imaginar el cálculo del promedio de sueldos de un departamento y luego mostrar ese promedio junto al sueldo individual de cada trabajador.

¿Cómo se Utiliza? (Funciones de Ventana)

La cláusula **OVER** se usa únicamente junto con una **Función de Ventana**.
Estas funciones pueden ser de tres categorías:

1. Funciones de Agregación (en contexto de ventana)

Las funciones tradicionales como **SUM, AVG, COUNT, MIN, MAX** pasan a actuar como **funciones de ventana** cuando se les agrega la cláusula **OVER()**.

Sintaxis General:

```
FUNCION_DE_AGREGACION(columna) OVER (
    [PARTITION BY columna_para_dividir]
    [ORDER BY columna_para_ordenar]
    [Frame Specification]
)
```

2. Elementos Clave Dentro de OVER()

Componente	Función	Analogía
PARTITION BY	**Divide** las filas en grupos separados (ventanas). El cálculo se reinicia para cada grupo.	Clasificar empleados por **Departamento**.
ORDER BY	**Ordena** las filas dentro de cada partición. Esto es crucial para los cálculos de rango y las medias móviles.	Ordenar los empleados dentro de cada departamento por **Salario**.

Ejemplo 1: Promedio del Grupo (Sin colapsar filas)

Imaginemos que deseamos visualizar el sueldo de cada trabajador y, además, mostrar el promedio salarial correspondiente a su departamento, pero **sin combinar las filas**.

```
SELECT
    Nombre,
    Departamento,
    Salario,
           AVG(Salario)  OVER  (PARTITION  BY  Departamento)  AS
PromedioDepartamento
FROM Empleados;
```

En este caso, la función **AVG()** calcula el promedio de los sueldos por departamento, pero gracias a **OVER()**, cada fila sigue mostrando la información individual del empleado junto con el promedio de su grupo.

Nombre	Departamento	Salario	PromedioDepartamento
Ana	Ventas	40000	45000
Luis	Ventas	50000	45000
Clara	IT	70000	75000
Pedro	IT	80000	75000

Resultado:

A diferencia de usar **GROUP BY**, aquí obtenemos las cuatro filas originales, y la columna **PromedioDepartamento** muestra el mismo valor promedio en todas las filas que pertenecen al mismo departamento.

Ejemplo 2: Funciones de Ranking (La Aplicación más Útil)

Las funciones de ranking solo pueden emplearse junto con la cláusula **OVER**.

• **ROW_NUMBER()**: Genera un número secuencial único para cada fila dentro de la ventana.
• **RANK()**: Asigna un puesto; si existen empates, el siguiente número de rango se salta.
• **DENSE_RANK()**: Asigna un puesto; si hay empates, el siguiente rango **no** se salta.

Consulta: Obtener la posición de salario más alta (ranking) dentro de cada departamento

```
SELECT
    Nombre,
    Departamento,
    Salario,
     DENSE_RANK() OVER (PARTITION BY Departamento ORDER BY Salario
DESC) AS RangoSalarial
FROM Empleados;
```

Explicación:

1. **PARTITION BY Departamento**: Reinicia el ranking para cada área (por ejemplo, IT, Ventas, etc.).

2. **ORDER BY Salario DESC**: El orden de clasificación se basa en el salario, desde el monto más elevado hacia el más bajo.

El resultado indicará si un empleado ocupa la primera posición salarial (Rango 1), la segunda (Rango 2), y así sucesivamente, considerando únicamente a los trabajadores de su propio departamento.

CLÁUSULA OVER

III. Explica de manera clara y didáctica qué son la **CLÁUSULA OVER** en SQL y cómo se utilizan.

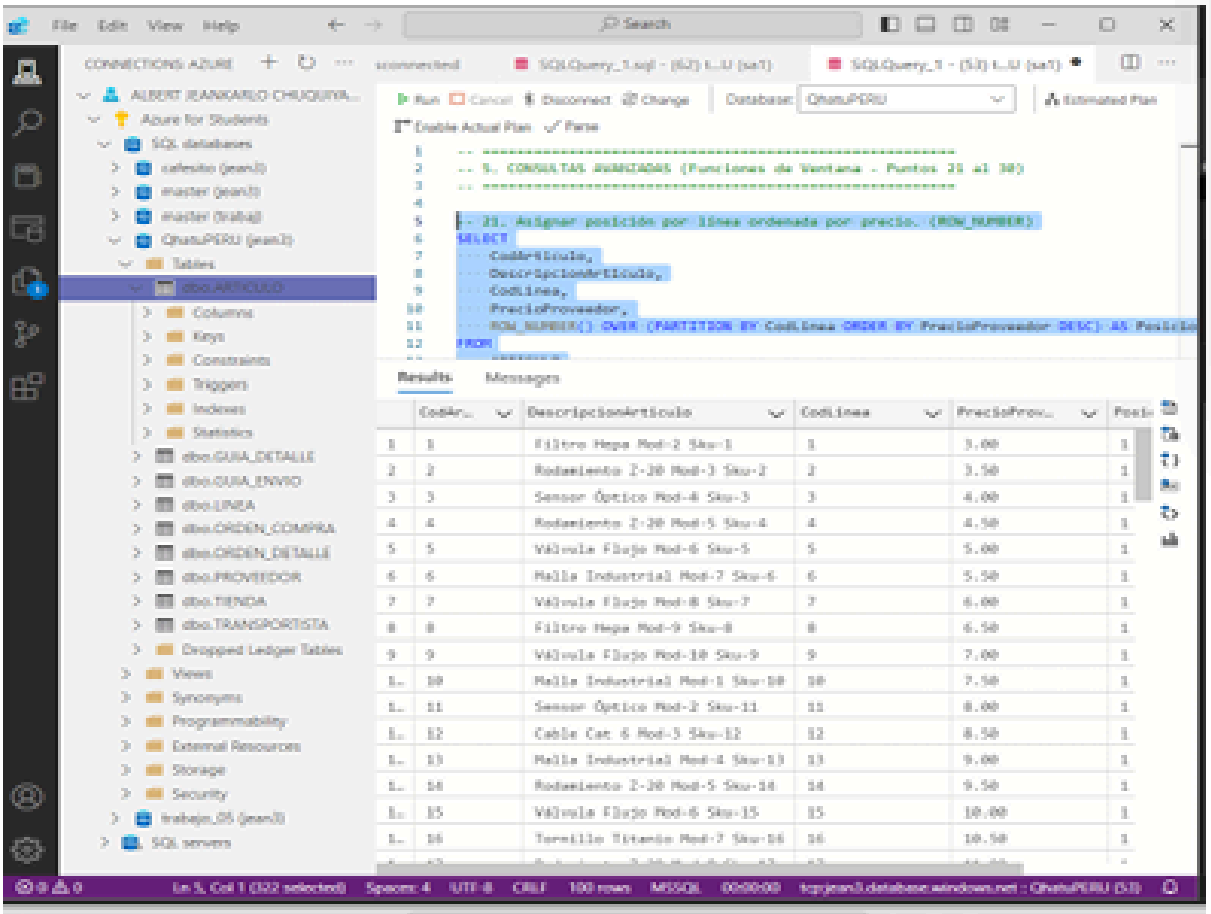
21. Asignar posición por línea ordenada por precio.
22. Calcular costo por orden y su RANK (RankCosto)
23. Mostrar TotalDia y AcumuladoVentas ordenado por fecha.
24. Calcular promedio móvil para stock.
25. Mostrar PrecioAnteriorMismoProveedor usando LAG.
26. Añadir columna CantidadPorLinea a cada artículo.
27. Mostrar MontoProveedor y PorcentajeDelTotal.
28. Mostrar solo los 3 artículos más caros por línea.
29. Mostrar transportista y su DenseRank por TotalEnviado.
30. Mostrar por guía la suma acumulada por tienda hasta esa guía (ordenada por FechaSalida)

21. Asignar posición por línea ordenada por precio.

Enunciado: Asignar una posición (número de fila) a cada artículo, partiendo desde 1, dentro de su propia línea de producto, ordenado por el precio del proveedor de forma descendente.

Consulta SQL:

```
-- 21. Asignar posición por línea ordenada por precio.
(ROW_NUMBER)
SELECT
    CodArticulo,
    DescripcionArticulo,
    CodLinea,
    PrecioProveedor,
    ROW_NUMBER() OVER (PARTITION BY CodLinea ORDER BY
PrecioProveedor DESC) AS PosicionPorPrecio
FROM
    ARTICULO
ORDER BY
    CodLinea, PosicionPorPrecio;
GO
```

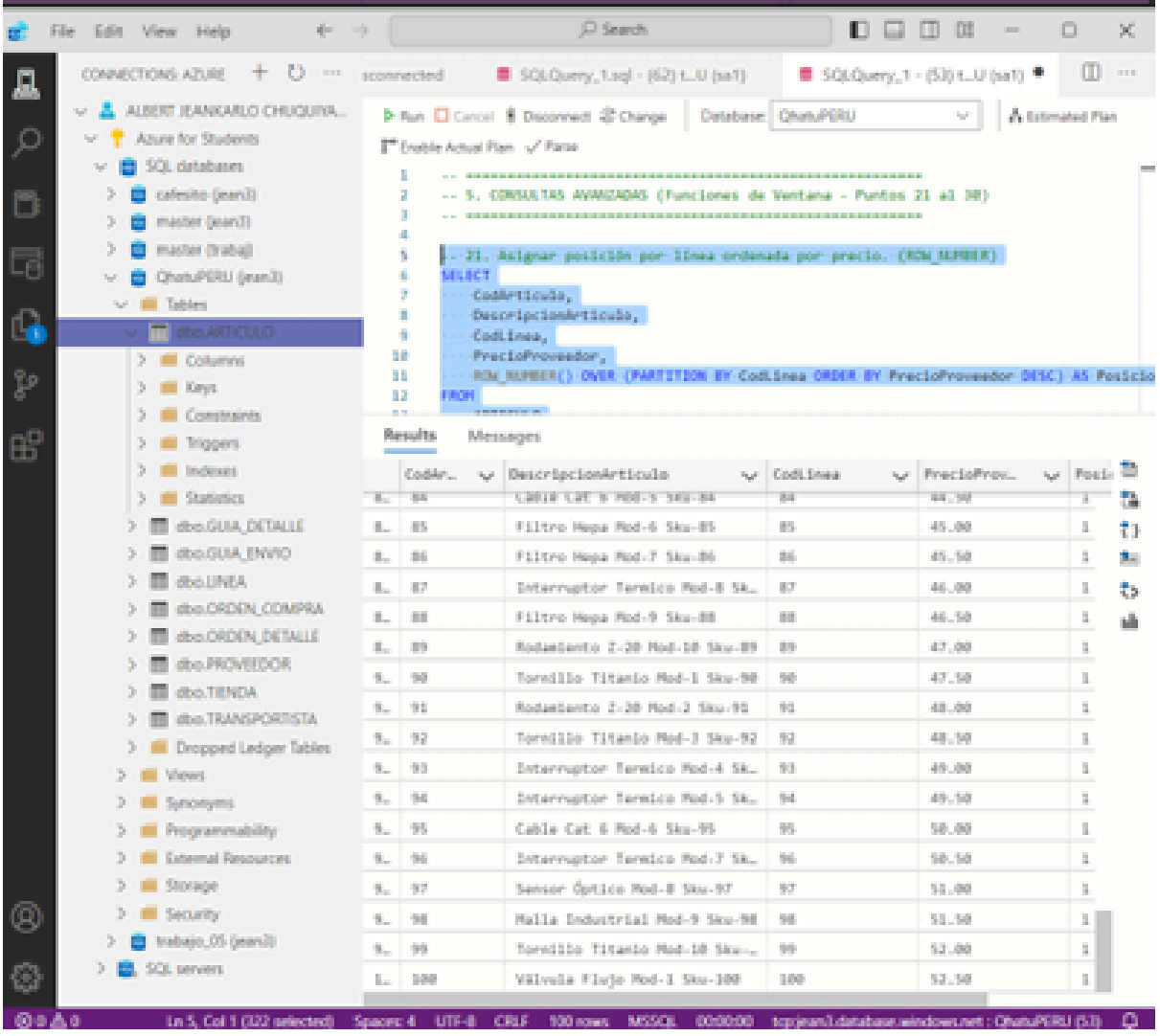


```
-- ============================================================
-- 5. CONSULTAS AVANZADAS (Funciones de Ventana - Puntos 21 al 30)
-- ============================================================

-- 21. Asignar posición por línea ordenada por precio. (ROW_NUMBER)
SELECT
    CodArticulo,
    DescripcionArticulo,
    CodLinea,
    PrecioProveedor,
    ROW_NUMBER() OVER (PARTITION BY CodLinea ORDER BY PrecioProveedor DESC) AS Posicio
FROM
```

Results Messages

	CodAr...	DescripcionArticulo	CodLinea	PrecioProv...	Posic
1	1	Filtro Hepa Mod-2 Sku-1	1	3.00	1
2	2	Rodamiento Z-20 Mod-3 Sku-2	2	3.50	1
3	3	Sensor Óptico Mod-4 Sku-3	3	4.00	1
4	4	Rodamiento Z-20 Mod-5 Sku-4	4	4.50	1
5	5	Válvula Flujo Mod-6 Sku-5	5	5.00	1
6	6	Malla Industrial Mod-7 Sku-6	6	5.50	1
7	7	Válvula Flujo Mod-8 Sku-7	7	6.00	1
8	8	Filtro Hepa Mod-9 Sku-8	8	6.50	1
9	9	Válvula Flujo Mod-10 Sku-9	9	7.00	1
1...	10	Malla Industrial Mod-1 Sku-10	10	7.50	1
1...	11	Sensor Óptico Mod-2 Sku-11	11	8.00	1
1...	12	Cable Cat 6 Mod-3 Sku-12	12	8.50	1
1...	13	Malla Industrial Mod-4 Sku-13	13	9.00	1
1...	14	Rodamiento Z-20 Mod-5 Sku-14	14	9.50	1
1...	15	Válvula Flujo Mod-6 Sku-15	15	10.00	1
1...	16	Tornillo Titanio Mod-7 Sku-16	16	10.50	1



```
-- ============================================================
-- 5. CONSULTAS AVANZADAS (Funciones de Ventana - Puntos 21 al 30)
-- ============================================================

-- 21. Asignar posición por línea ordenada por precio. (ROW_NUMBER)
SELECT
    CodArticulo,
    DescripcionArticulo,
    CodLinea,
    PrecioProveedor,
    ROW_NUMBER() OVER (PARTITION BY CodLinea ORDER BY PrecioProveedor DESC) AS Posicio
FROM
```

Results Messages

	CodAr...	DescripcionArticulo	CodLinea	PrecioProv...	Posic
8...	84	Cable Cat 6 Mod-5 Sku-84	84	44.50	1
8...	85	Filtro Hepa Mod-6 Sku-85	85	45.00	1
8...	86	Filtro Hepa Mod-7 Sku-86	86	45.50	1
8...	87	Interruptor Termico Mod-8 Sk...	87	46.00	1
8...	88	Filtro Hepa Mod-9 Sku-88	88	46.50	1
8...	89	Rodamiento Z-20 Mod-10 Sku-89	89	47.00	1
9...	90	Tornillo Titanio Mod-1 Sku-90	90	47.50	1
9...	91	Rodamiento Z-20 Mod-2 Sku-91	91	48.00	1
9...	92	Tornillo Titanio Mod-3 Sku-92	92	48.50	1
9...	93	Interruptor Termico Mod-4 Sk...	93	49.00	1
9...	94	Interruptor Termico Mod-5 Sk...	94	49.50	1
9...	95	Cable Cat 6 Mod-6 Sku-95	95	50.00	1
9...	96	Interruptor Termico Mod-7 Sk...	96	50.50	1
9...	97	Sensor Óptico Mod-8 Sku-97	97	51.00	1
9...	98	Malla Industrial Mod-9 Sku-98	98	51.50	1
9...	99	Tornillo Titanio Mod-10 Sku-...	99	52.00	1
1...	100	Válvula Flujo Mod-1 Sku-100	100	52.50	1

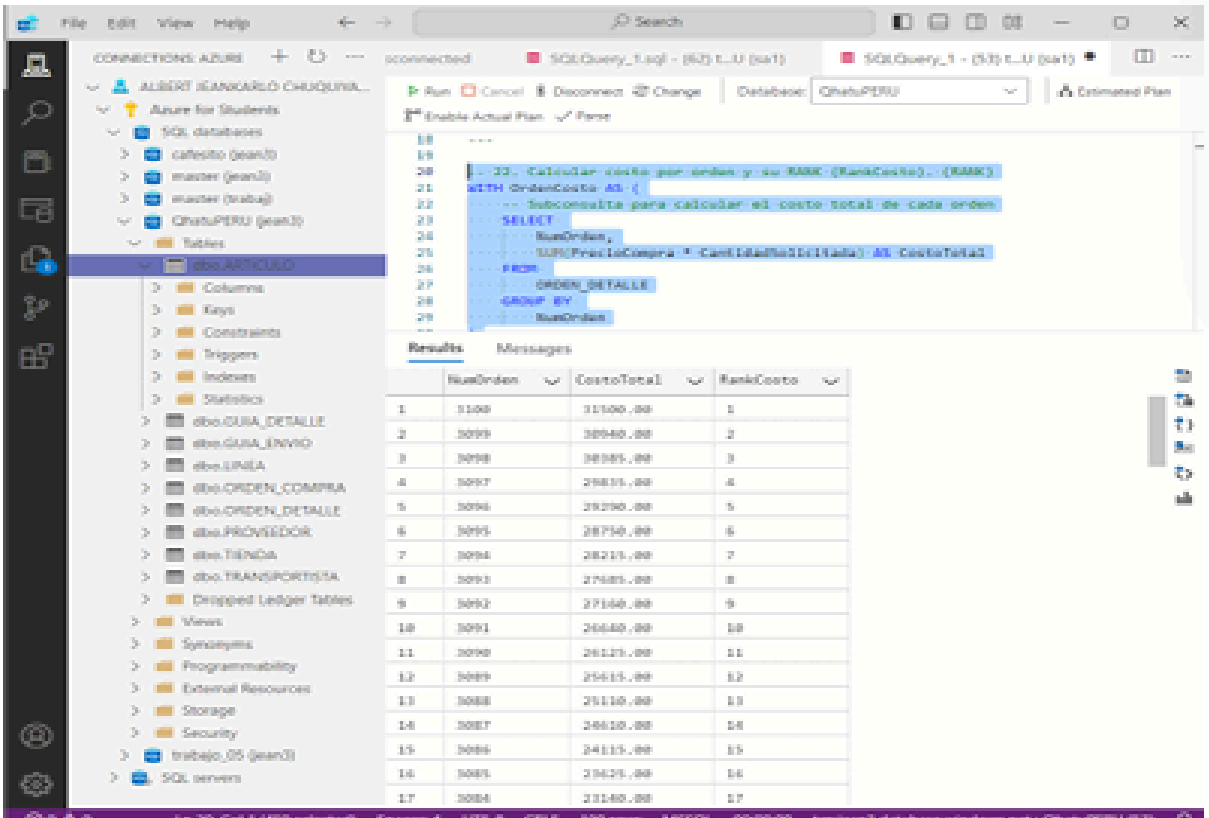
Explicación: Se utiliza la función **ROW_NUMBER()** dentro de la cláusula OVER. El **PARTITION BY CodLinea** agrupa los artículos por línea, y el **ORDER BY PrecioProveedor DESC** asigna la posición dentro de cada grupo, dando el número 1 al artículo más caro de esa línea.

22. Calcular costo por orden y su RANK (RankCosto).

Enunciado: Calcular el costo total de cada orden de compra y luego asignar un rango de clasificación (RANK) a esas órdenes basado en dicho costo.

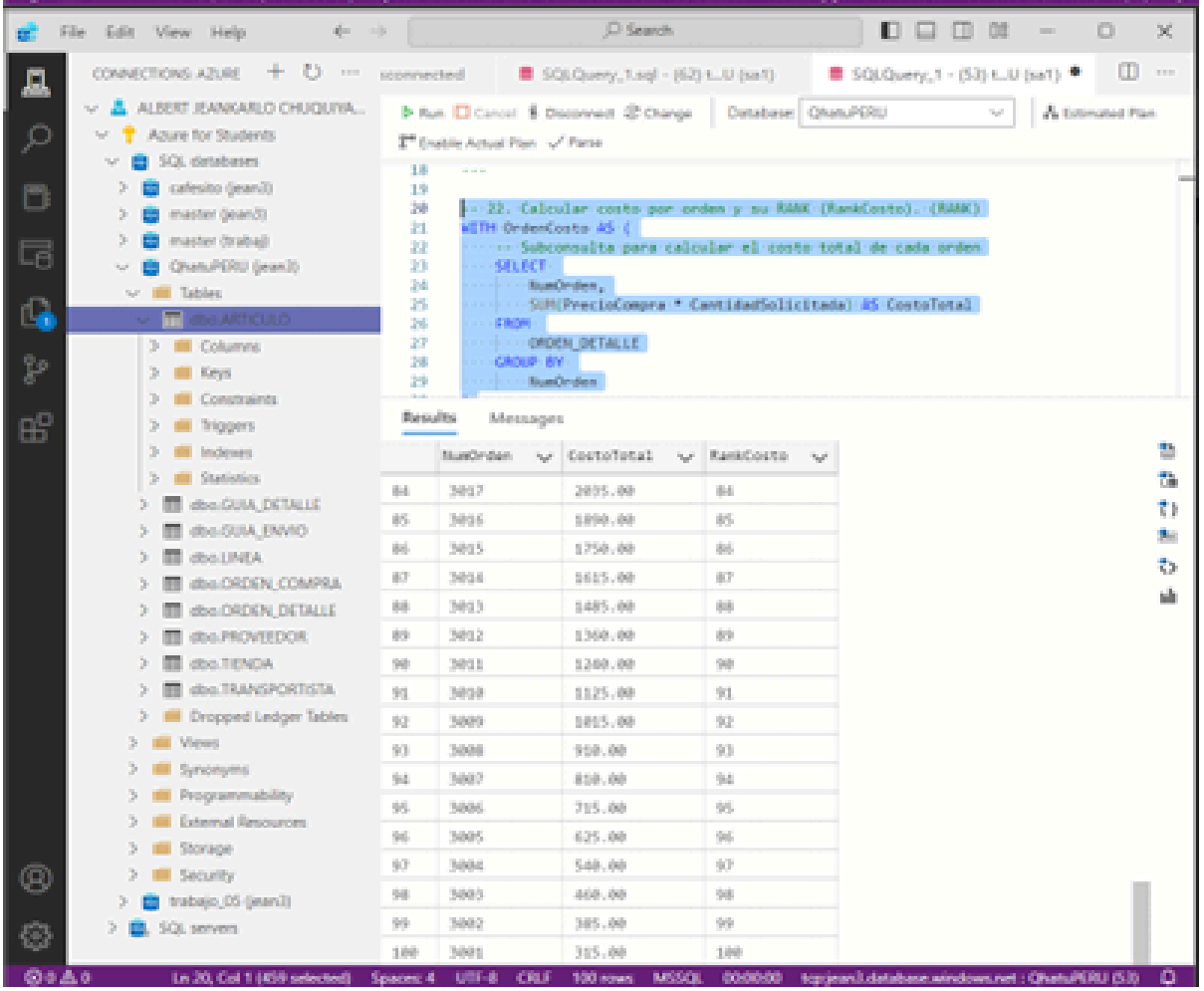
Consulta SQL:

```
-- 22. Calcular costo por orden y su RANK (RankCosto). (RANK)
WITH OrdenCosto AS (
    -- Subconsulta para calcular el costo total de cada orden
    SELECT
        NumOrden,
        SUM(PrecioCompra * CantidadSolicitada) AS CostoTotal
    FROM
        ORDEN_DETALLE
    GROUP BY
        NumOrden
)
SELECT
    NumOrden,
    CostoTotal,
    RANK() OVER (ORDER BY CostoTotal DESC) AS RankCosto
FROM
    OrdenCosto
ORDER BY
    RankCosto;
GO
```



```sql
-- 22. Calcular costo por orden y su RANK (RankCosto). (RANK)
WITH OrdenCosto AS (
    -- Subconsulta para calcular el costo total de cada orden
    SELECT
        NumOrden,
        SUM(PrecioCompra * CantidadSolicitada) AS CostoTotal
    FROM
        ORDEN_DETALLE
    GROUP BY
        NumOrden
```

	NumOrden	CostoTotal	RankCosto
1	3100	31500.00	1
2	3099	30940.00	2
3	3098	30385.00	3
4	3097	29835.00	4
5	3096	29290.00	5
6	3095	28750.00	6
7	3094	28215.00	7
8	3093	27685.00	8
9	3092	27160.00	9
10	3091	26640.00	10
11	3090	26125.00	11
12	3089	25615.00	12
13	3088	25110.00	13
14	3087	24610.00	14
15	3086	24115.00	15
16	3085	23625.00	16
17	3084	23140.00	17



```sql
-- 22. Calcular costo por orden y su RANK (RankCosto). (RANK)
WITH OrdenCosto AS (
    -- Subconsulta para calcular el costo total de cada orden
    SELECT
        NumOrden,
        SUM(PrecioCompra * CantidadSolicitada) AS CostoTotal
    FROM
        ORDEN_DETALLE
    GROUP BY
        NumOrden
```

	NumOrden	CostoTotal	RankCosto
84	3017	2035.00	84
85	3016	1890.00	85
86	3015	1750.00	86
87	3014	1615.00	87
88	3013	1485.00	88
89	3012	1360.00	89
90	3011	1240.00	90
91	3010	1125.00	91
92	3009	1015.00	92
93	3008	910.00	93
94	3007	810.00	94
95	3006	715.00	95
96	3005	625.00	96
97	3004	540.00	97
98	3003	460.00	98
99	3002	385.00	99
100	3001	315.00	100

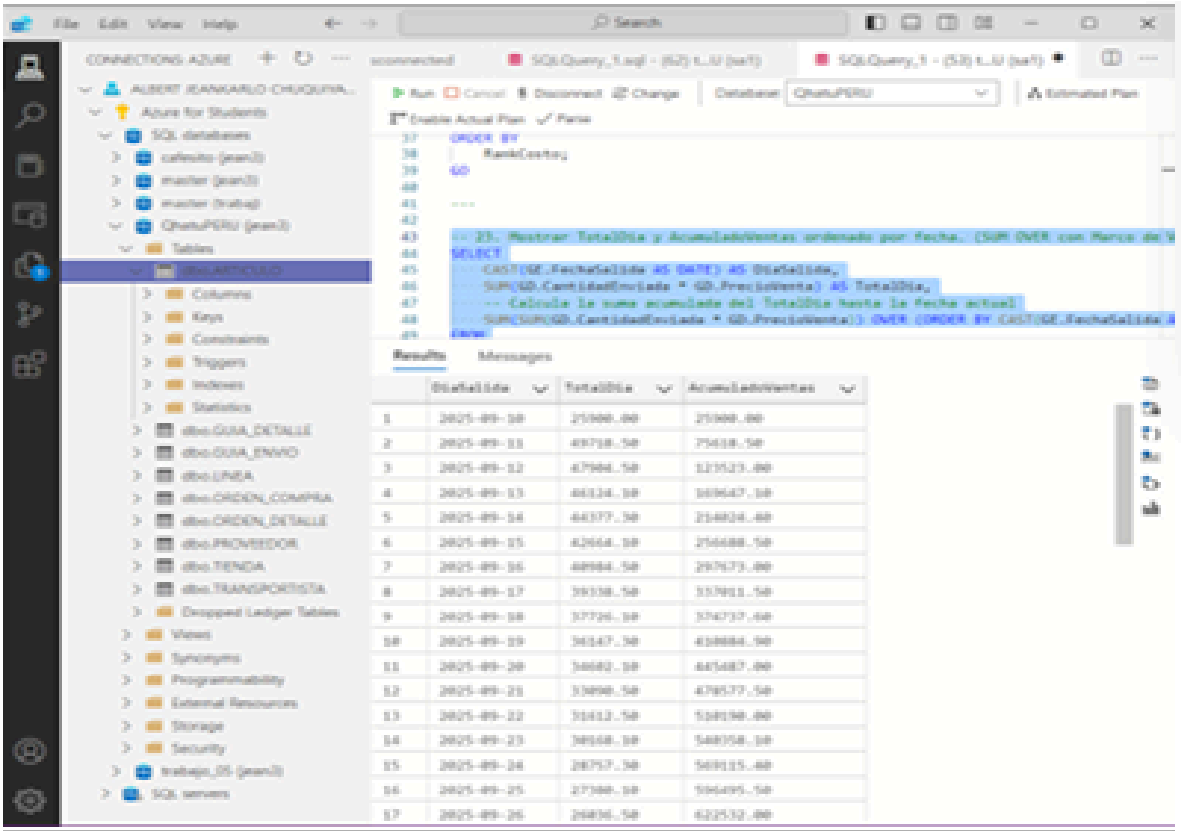
Explicación: Primero, una Expresión de Tabla Común (CTE) calcula el costo total por NumOrden. Luego, la función **RANK()** se aplica sobre ese costo total para clasificar todas las órdenes. Los empates reciben el mismo rango, y el siguiente rango salta posiciones.

23. Mostrar TotalDia y AcumuladoVentas ordenado por fecha.

Enunciado: Calcular el valor total de ventas de cada día (TotalDia) y mostrar el valor acumulado de ventas hasta esa fecha (AcumuladoVentas).

Consulta SQL:

```
-- 23. Mostrar TotalDia y AcumuladoVentas ordenado por fecha.
(SUM OVER con Marco de Ventana)
SELECT
    CAST(GE.FechaSalida AS DATE) AS DiaSalida,
    SUM(GD.CantidadEnviada * GD.PrecioVenta) AS TotalDia,
    -- Calcula la suma acumulada del TotalDia hasta la fecha actual
    SUM(SUM(GD.CantidadEnviada * GD.PrecioVenta)) OVER
(ORDER BY CAST(GE.FechaSalida AS DATE) ROWS BETWEEN
UNBOUNDED PRECEDING AND CURRENT ROW) AS
AcumuladoVentas
FROM
    GUIA_ENVIO GE
JOIN
    GUIA_DETALLE GD ON GE.NumGuia = GD.NumGuia
GROUP BY
    CAST(GE.FechaSalida AS DATE)
ORDER BY
    DiaSalida;
GO
```

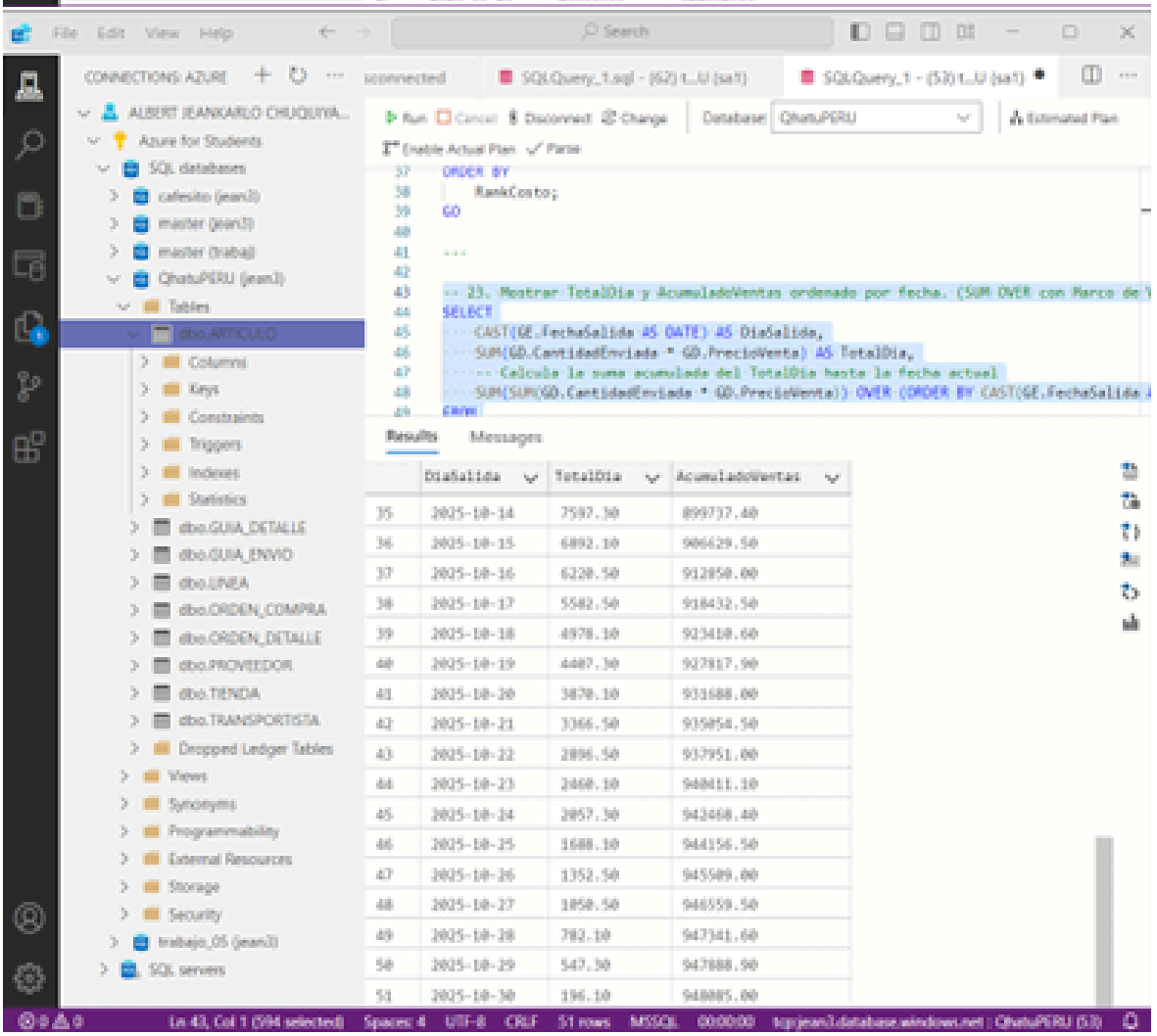


```
ORDER BY
    RankCosto;
GO

---

-- 23. Mostrar TotalDia y AcumuladoVentas ordenado por fecha. (SUM OVER con Marco de V
SELECT
    CAST(GE.FechaSalida AS DATE) AS DiaSalida,
    SUM(GD.CantidadEnviada * GD.PrecioVenta) AS TotalDia,
    -- Calcula la suma acumulada del TotalDia hasta la fecha actual
    SUM(SUM(GD.CantidadEnviada * GD.PrecioVenta)) OVER (ORDER BY CAST(GE.FechaSalida A
FROM
```

	DiaSalida	TotalDia	AcumuladoVentas
1	2025-09-10	251000.00	251000.00
2	2025-09-11	49718.50	754618.50
3	2025-09-12	47984.50	1235123.00
4	2025-09-13	46134.10	1639647.10
5	2025-09-14	44377.50	2144024.60
6	2025-09-15	42664.10	2566688.50
7	2025-09-16	40984.50	2979673.00
8	2025-09-17	39338.50	3357011.50
9	2025-09-18	37726.10	3747737.60
10	2025-09-19	36147.50	4108884.90
11	2025-09-20	34602.10	4475487.00
12	2025-09-21	33090.50	4785577.50
13	2025-09-22	31612.50	5169190.00
14	2025-09-23	30168.10	5489358.10
15	2025-09-24	28757.50	5691115.60
16	2025-09-25	27380.10	5964495.70
17	2025-09-26	26036.50	6225532.00



```
ORDER BY
    RankCosto;
GO

---

-- 23. Mostrar TotalDia y AcumuladoVentas ordenado por fecha. (SUM OVER con Marco de V
SELECT
    CAST(GE.FechaSalida AS DATE) AS DiaSalida,
    SUM(GD.CantidadEnviada * GD.PrecioVenta) AS TotalDia,
    -- Calcula la suma acumulada del TotalDia hasta la fecha actual
    SUM(SUM(GD.CantidadEnviada * GD.PrecioVenta)) OVER (ORDER BY CAST(GE.FechaSalida A
FROM
```

	DiaSalida	TotalDia	AcumuladoVentas
35	2025-10-14	7597.30	899737.40
36	2025-10-15	6892.10	906629.50
37	2025-10-16	6220.50	912850.00
38	2025-10-17	5582.50	918432.50
39	2025-10-18	4978.10	923410.60
40	2025-10-19	4407.30	927817.90
41	2025-10-20	3870.10	931688.00
42	2025-10-21	3366.50	935054.50
43	2025-10-22	2896.50	937951.00
44	2025-10-23	2460.10	940411.10
45	2025-10-24	2057.30	942468.40
46	2025-10-25	1688.10	944156.50
47	2025-10-26	1352.50	945509.00
48	2025-10-27	1050.50	946559.50
49	2025-10-28	782.10	947341.60
50	2025-10-29	547.30	947888.90
51	2025-10-30	196.10	948085.00

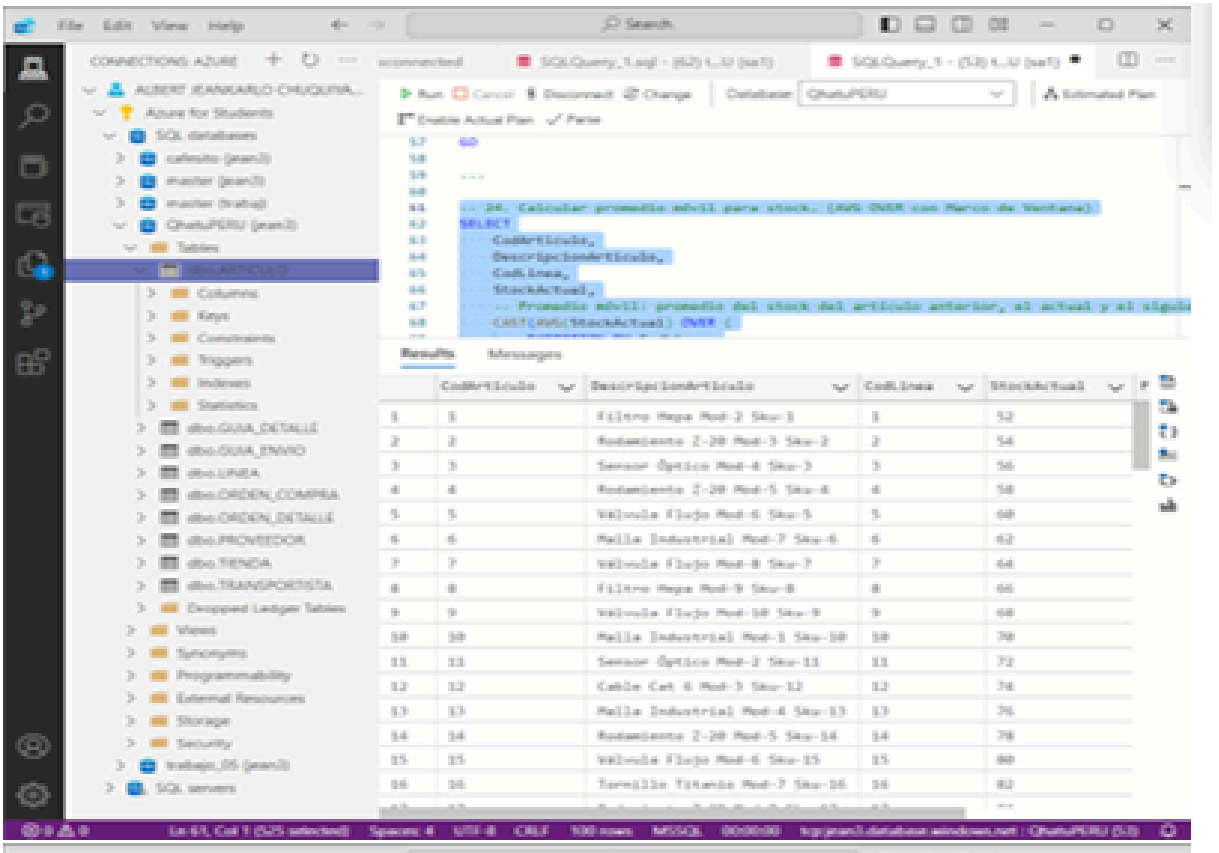
Explicación: Se agrupa la venta total por día. Luego, se utiliza una segunda función **SUM()** con OVER sin PARTITION BY y con un marco de ventana (ROWS BETWEEN...) para calcular un **total acumulado** global, sumando las ventas desde el inicio hasta la fila actual (la fecha actual).

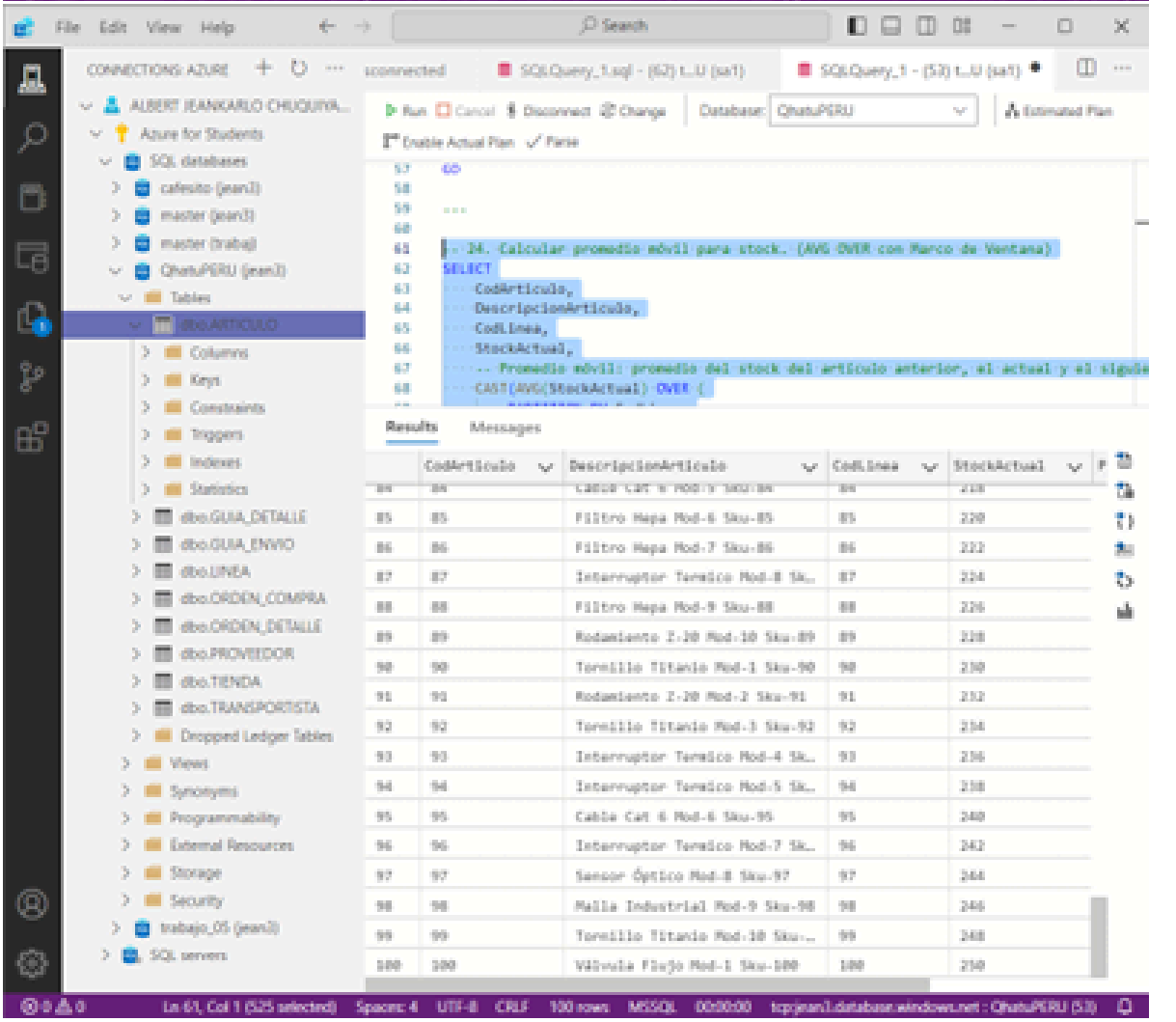
24. Calcular promedio móvil para stock.

Enunciado: Para cada artículo, calcular el promedio móvil de StockActual, promediando el stock del artículo anterior, el artículo actual y el artículo siguiente dentro de su misma línea.

Consulta SQL:

```
-- 24. Calcular promedio móvil para stock. (AVG OVER con Marco de Ventana)
SELECT
    CodArticulo,
    DescripcionArticulo,
    CodLinea,
    StockActual,
    -- Promedio móvil: promedio del stock del artículo anterior, el actual y el siguiente dentro de la línea
    CAST(AVG(StockActual) OVER (
        PARTITION BY CodLinea
        ORDER BY CodArticulo
        ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING
    ) AS DECIMAL(10,2)) AS PromedioMovilStock
FROM
    ARTICULO
ORDER BY
    CodLinea, CodArticulo;
GO
```





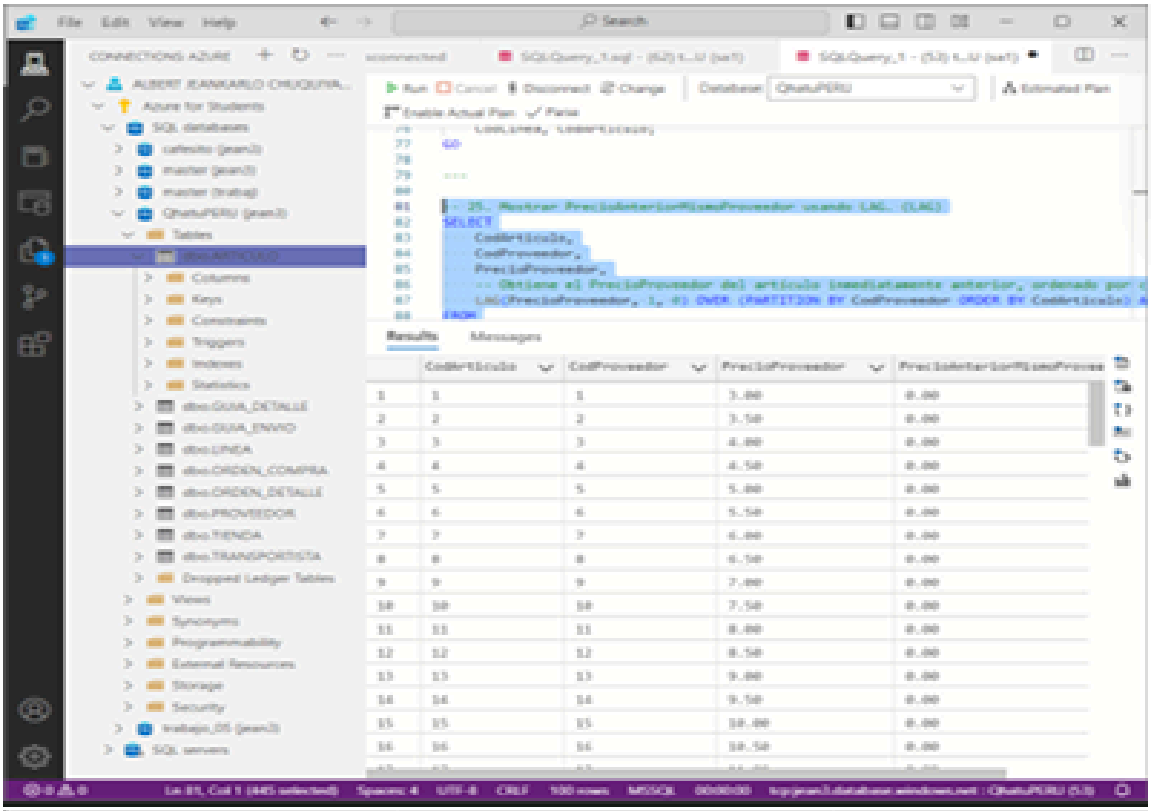
Explicación: Se utiliza la función **AVG()** con OVER. El PARTITION BY CodLinea asegura que el promedio se calcule por línea. El marco de ventana **ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING** define el "móvil", que incluye un registro anterior, el actual, y un registro siguiente para el cálculo del promedio.

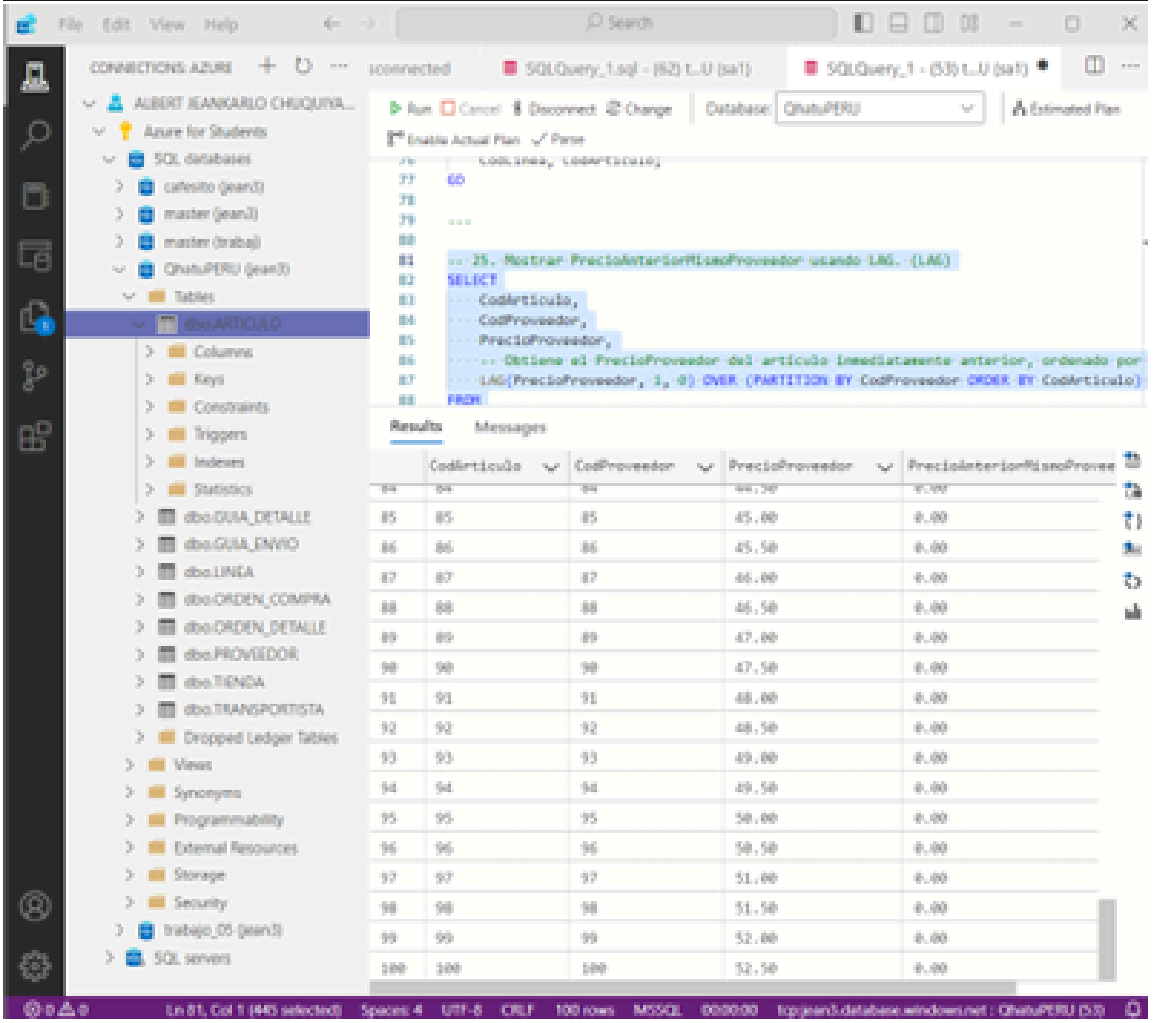
25. Mostrar PrecioAnteriorMismoProveedor usando LAG.

Enunciado: Mostrar el precio del proveedor (PrecioProveedor) del artículo actual junto con el precio del artículo inmediatamente anterior, siempre y cuando sea del mismo proveedor.

Consulta SQL:

```
-- 25. Mostrar PrecioAnteriorMismoProveedor usando LAG. (LAG)
SELECT
    CodArticulo,
    CodProveedor,
    PrecioProveedor,
    -- Obtiene el PrecioProveedor del artículo inmediatamente
anterior, ordenado por código de artículo, dentro del mismo
proveedor
    LAG(PrecioProveedor, 1, 0) OVER (PARTITION BY
CodProveedor ORDER BY CodArticulo) AS
PrecioAnteriorMismoProveedor
FROM
    ARTICULO
ORDER BY
    CodProveedor, CodArticulo;
GO
```





Explicación: Se usa la función de ventana **LAG()** para acceder al valor de una fila anterior. El PARTITION BY CodProveedor asegura que solo se compare con artículos del mismo proveedor, mientras que el ORDER BY CodArticulo define cuál es el "anterior" lógico. El 1 indica la diferencia de 1 fila, y el 0 es el valor por defecto si no hay fila anterior.

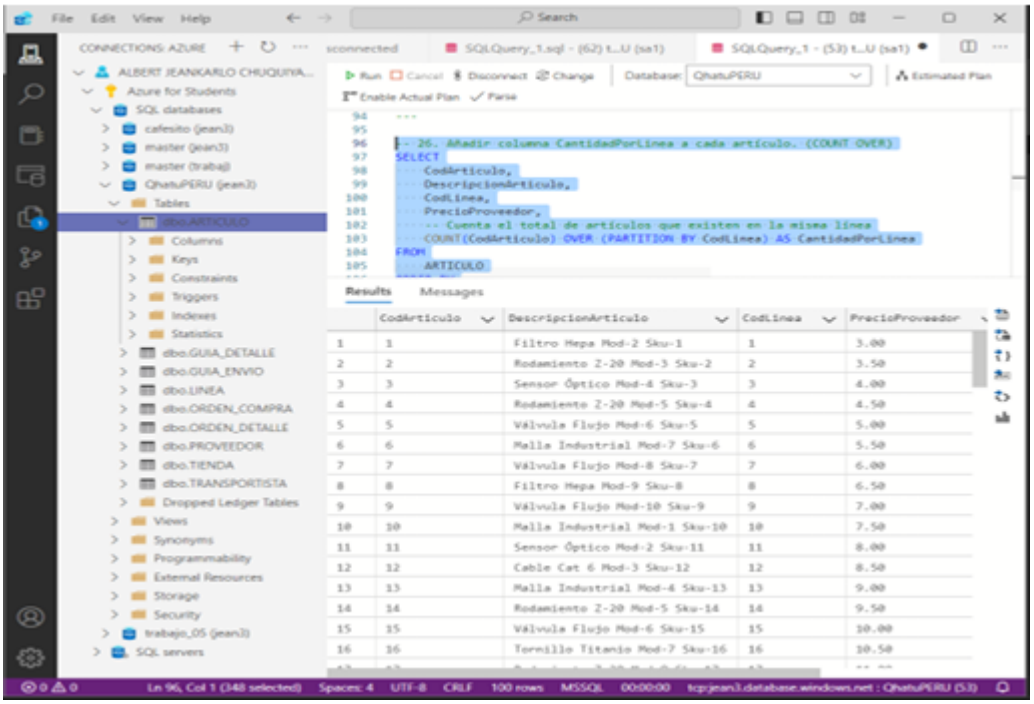
26. Añadir columna CantidadPorLinea a cada artículo.

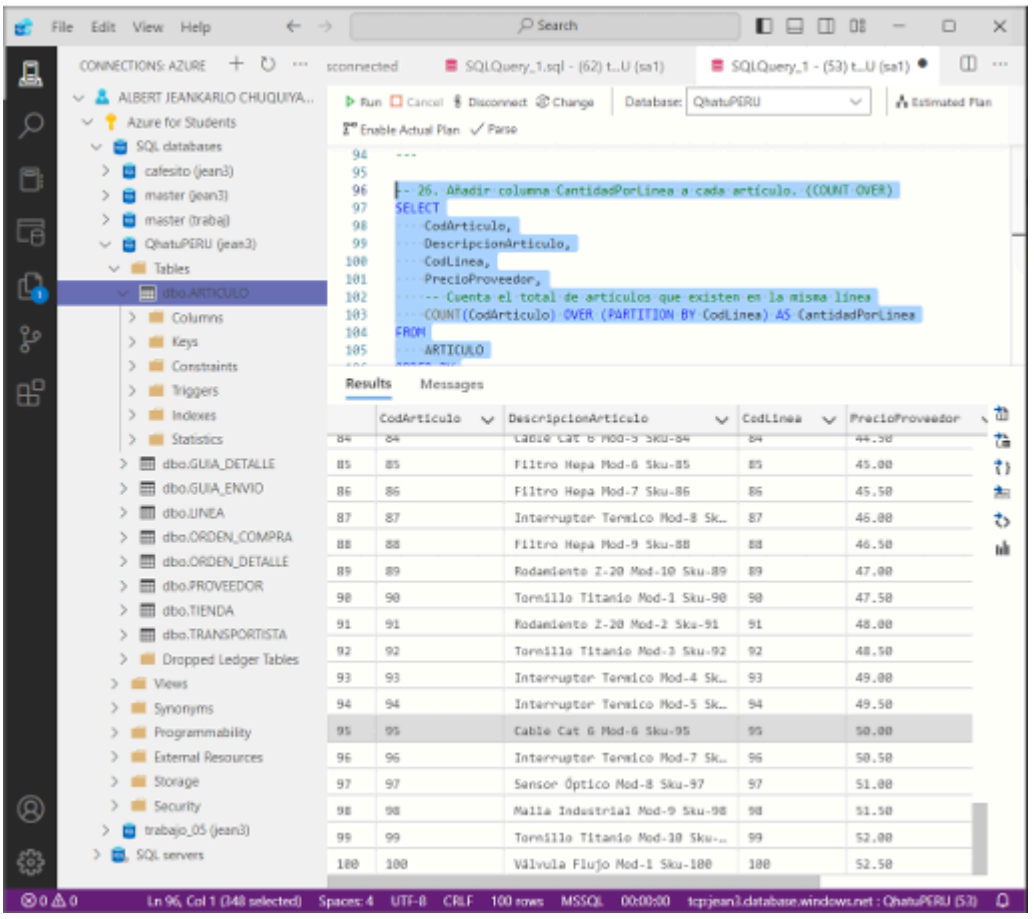
Enunciado: Para cada artículo, añadir una columna que muestre el conteo total de artículos que existen en su misma línea de producto.

Consulta SQL:

SQL

```
-- 26. Añadir columna CantidadPorLinea a cada artículo. (COUNT
OVER)
SELECT
    CodArticulo,
    DescripcionArticulo,
    CodLinea,
    PrecioProveedor,
    -- Cuenta el total de artículos que existen en la misma línea
    COUNT(CodArticulo) OVER (PARTITION BY CodLinea) AS
CantidadPorLinea
FROM
    ARTICULO
ORDER BY
    CodLinea;
GO
```


-- 26. Añadir columna CantidadPorLinea a cada artículo. (COUNT OVER)
SELECT
CodArticulo,
DescripcionArticulo,
CodLinea,
PrecioProveedor,
-- Cuenta el total de artículos que existen en la misma línea
COUNT(CodArticulo) OVER (PARTITION BY CodLinea) AS CantidadPorLinea
FROM
ARTICULO
Results
Messages
CodArticulo
DescripcionArticulo
CodLinea
PrecioProveedor
1 1 Filtro Hepa Mod-2 Sku-1 1 3.00
2 2 Rodamiento Z-20 Mod-3 Sku-2 2 3.50
3 3 Sensor Óptico Mod-4 Sku-3 3 4.00
4 4 Rodamiento Z-20 Mod-5 Sku-4 4 4.50
5 5 Válvula Flujo Mod-6 Sku-5 5 5.00
6 6 Malla Industrial Mod-7 Sku-6 6 5.50
7 7 Válvula Flujo Mod-8 Sku-7 7 6.00
8 8 Filtro Hepa Mod-9 Sku-8 8 6.50
9 9 Válvula Flujo Mod-10 Sku-9 9 7.00
10 10 Malla Industrial Mod-1 Sku-10 10 7.50
11 11 Sensor Óptico Mod-2 Sku-11 11 8.00
12 12 Cable Cat 6 Mod-3 Sku-12 12 8.50
13 13 Malla Industrial Mod-4 Sku-13 13 9.00
14 14 Rodamiento Z-20 Mod-5 Sku-14 14 9.50
15 15 Válvula Flujo Mod-6 Sku-15 15 10.00
16 16 Tornillo Titanio Mod-7 Sku-16 16 10.50


-- 26. Añadir columna CantidadPorLinea a cada artículo. (COUNT OVER)
SELECT
CodArticulo,
DescripcionArticulo,
CodLinea,
PrecioProveedor,
-- Cuenta el total de artículos que existen en la misma línea
COUNT(CodArticulo) OVER (PARTITION BY CodLinea) AS CantidadPorLinea
FROM
ARTICULO
Results
Messages
CodArticulo
DescripcionArticulo
CodLinea
PrecioProveedor
85 85 Filtro Hepa Mod-6 Sku-85 85 45.00
86 86 Filtro Hepa Mod-7 Sku-86 86 45.50
87 87 Interruptor Termico Mod-8 Sk... 87 46.00
88 88 Filtro Hepa Mod-9 Sku-88 88 46.50
89 89 Rodamiento Z-20 Mod-10 Sku-89 89 47.00
90 90 Tornillo Titanio Mod-1 Sku-90 90 47.50
91 91 Rodamiento Z-20 Mod-2 Sku-91 91 48.00
92 92 Tornillo Titanio Mod-3 Sku-92 92 48.50
93 93 Interruptor Termico Mod-4 Sk... 93 49.00
94 94 Interruptor Termico Mod-5 Sk... 94 49.50
95 95 Cable Cat 6 Mod-6 Sku-95 95 50.00
96 96 Interruptor Termico Mod-7 Sk... 96 50.50
97 97 Sensor Óptico Mod-8 Sku-97 97 51.00
98 98 Malla Industrial Mod-9 Sku-98 98 51.50
99 99 Tornillo Titanio Mod-10 Sku-... 99 52.00
100 100 Válvula Flujo Mod-1 Sku-100 100 52.50

Explicación: Se utiliza la función de agregación **COUNT()** con la cláusula OVER. El **PARTITION BY CodLinea** instruye a SQL a contar los artículos dentro de cada línea de forma independiente, repitiendo el resultado del conteo en cada fila de esa partición.

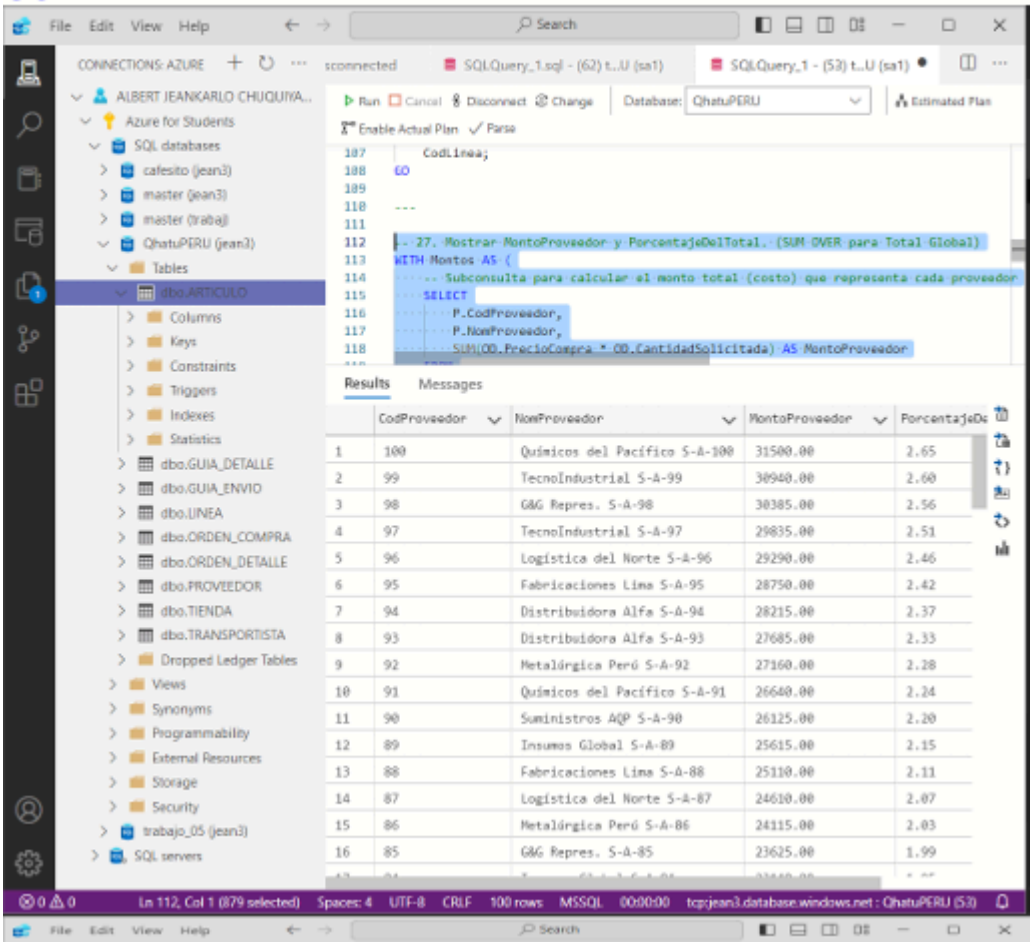
27. Mostrar MontoProveedor y PorcentajeDelTotal.

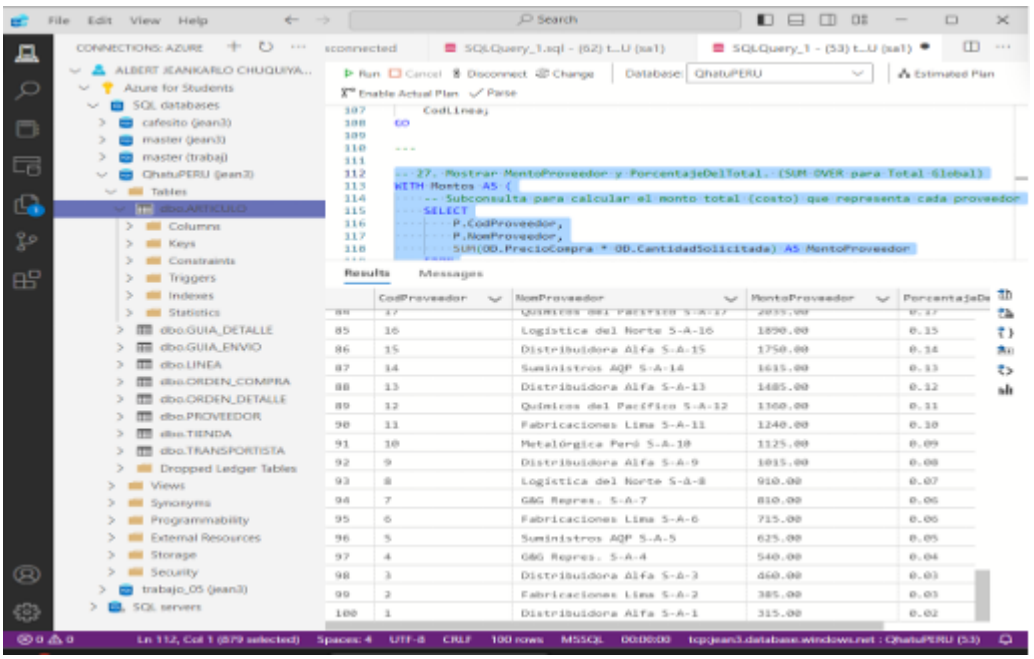
Enunciado: Calcular el monto total de compra por cada proveedor (MontoProveedor) y mostrar qué porcentaje representa ese monto respecto al monto total global de todas las compras.

Consulta SQL:

```
-- 27. Mostrar MontoProveedor y PorcentajeDelTotal. (SUM OVER para Total Global)
WITH Montos AS (
    -- Subconsulta para calcular el monto total (costo) que representa cada proveedor
    SELECT
        P.CodProveedor,
        P.NomProveedor,
        SUM(OD.PrecioCompra * OD.CantidadSolicitada) AS MontoProveedor
    FROM
        PROVEEDOR P
    JOIN
        ARTICULO A ON P.CodProveedor = A.CodProveedor
    JOIN
        ORDEN_DETALLE OD ON A.CodArticulo = OD.CodArticulo
    GROUP BY
        P.CodProveedor, P.NomProveedor
)
SELECT
    CodProveedor,
    NomProveedor,
    MontoProveedor,
    -- Se divide el MontoProveedor entre la suma total de todos los montos de la tabla, usando SUM() OVER ()
    CAST(MontoProveedor / SUM(MontoProveedor) OVER () * 100 AS DECIMAL(5,2)) AS PorcentajeDelTotal
FROM
    Montos
ORDER BY
```

```
    MontoProveedor DESC;
GO
```





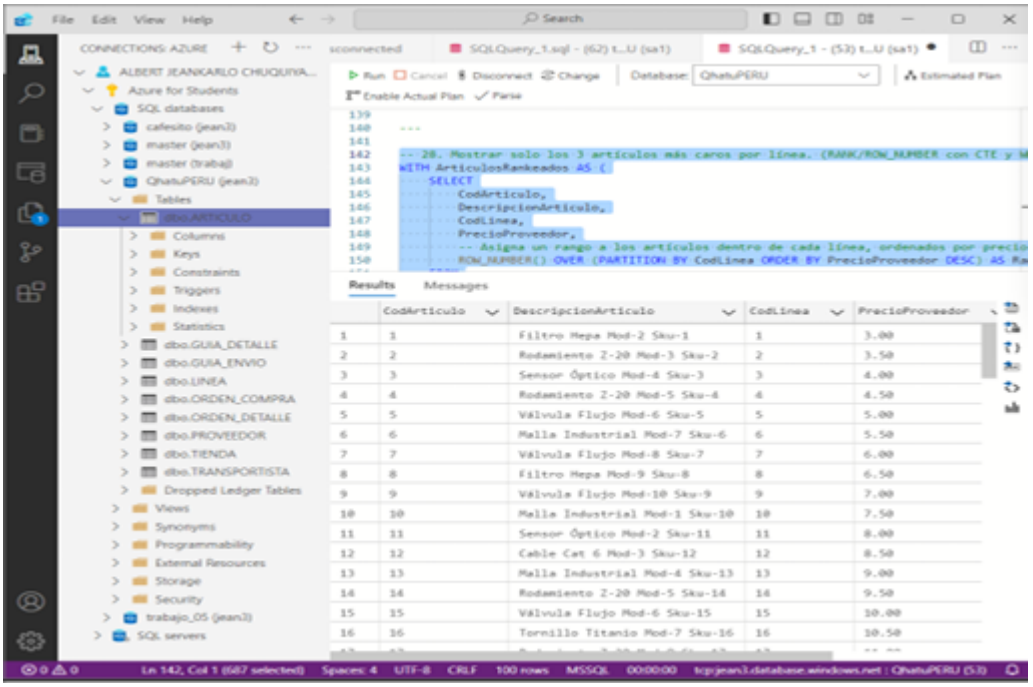
Explicación: La CTE calcula el MontoProveedor. Luego, se usa la función **SUM(MontoProveedor) OVER ()** para obtener la suma total de todos los montos (sin partición), que es el denominador. El monto del proveedor se divide entre este total y se multiplica por 100 para obtener el porcentaje.

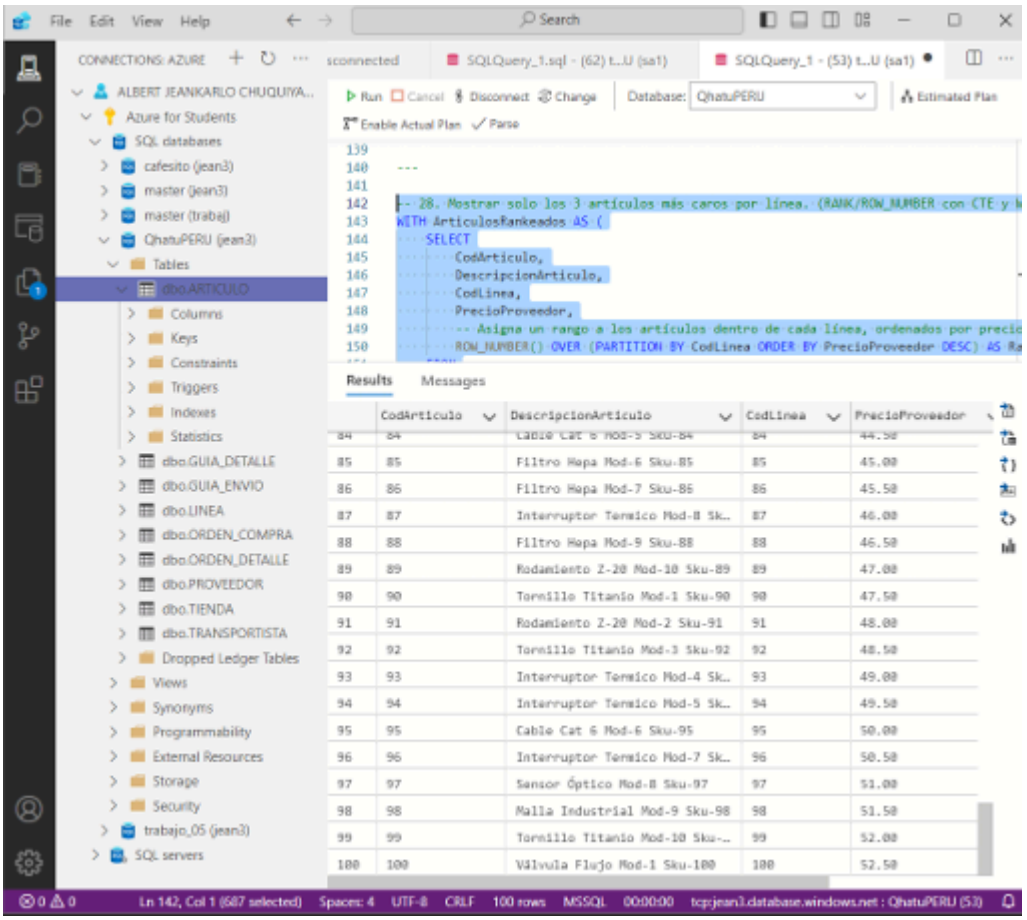
28. Mostrar solo los 3 artículos más caros por línea.

Enunciado: Mostrar el código, descripción y precio de solo los 3 artículos más caros dentro de cada línea de producto.

Consulta SQL:

```
-- 28. Mostrar solo los 3 artículos más caros por línea.
(RANK/ROW_NUMBER con CTE y WHERE)
WITH ArticulosRankeados AS (
    SELECT
        CodArticulo,
        DescripcionArticulo,
        CodLinea,
        PrecioProveedor,
        -- Asigna un rango a los artículos dentro de cada línea,
ordenados por precio (más caro primero)
        ROW_NUMBER() OVER (PARTITION BY CodLinea ORDER
BY PrecioProveedor DESC) AS RankPrecioLinea
    FROM
        ARTICULO
)
SELECT
    CodArticulo,
    DescripcionArticulo,
    CodLinea,
    PrecioProveedor,
    RankPrecioLinea
FROM
    ArticulosRankeados
WHERE
    RankPrecioLinea <= 3
ORDER BY
    CodLinea, RankPrecioLinea;
GO
```


File Edit View Help
Search
CONNECTIONS: AZURE
ALBERT JEANKARLO CHUQUIYA...
Azure for Students
SQL databases
cafesito (jean3)
master (jean3)
master (trabaj)
QhatuPERU (jean3)
Tables
dbo.ARTICULO
Columns
Keys
Constraints
Triggers
Indexes
Statistics
dbo.GUIA_DETALLE
dbo.GUIA_ENVIO
dbo.LINEA
dbo.ORDEN_COMPRA
dbo.ORDEN_DETALLE
dbo.PROVEEDOR
dbo.TIENDA
dbo.TRANSPORTISTA
Dropped Ledger Tables
Views
Synonyms
Programmability
External Resources
Storage
Security
trabajo_05 (jean3)
SQL servers
sconnected
SQLQuery_1.sql - (62) t...U (sa1)
SQLQuery_1 - (53) t...U (sa1)
Run
Cancel
Disconnect
Change
Database: QhatuPERU
Estimated Plan
Enable Actual Plan
Parse
-- 28. Mostrar solo los 3 artículos más caros por línea. (RANK/ROW_NUMBER con CTE y W
WITH ArticulosRankeados AS (
SELECT
CodArticulo,
DescripcionArticulo,
CodLinea,
PrecioProveedor,
-- Asigna un rango a los artículos dentro de cada línea, ordenados por precio
ROW_NUMBER() OVER (PARTITION BY CodLinea ORDER BY PrecioProveedor DESC) AS Ra
Results
Messages
CodArticulo
DescripcionArticulo
CodLinea
PrecioProveedor
1 1 Filtro Hepa Mod-2 Sku-1 1 3.00
2 2 Rodamiento Z-20 Mod-3 Sku-2 2 3.50
3 3 Sensor Óptico Mod-4 Sku-3 3 4.00
4 4 Rodamiento Z-20 Mod-5 Sku-4 4 4.50
5 5 Válvula Flujo Mod-6 Sku-5 5 5.00
6 6 Malla Industrial Mod-7 Sku-6 6 5.50
7 7 Válvula Flujo Mod-8 Sku-7 7 6.00
8 8 Filtro Hepa Mod-9 Sku-8 8 6.50
9 9 Válvula Flujo Mod-10 Sku-9 9 7.00
10 10 Malla Industrial Mod-1 Sku-10 10 7.50
11 11 Sensor Óptico Mod-2 Sku-11 11 8.00
12 12 Cable Cat 6 Mod-3 Sku-12 12 8.50
13 13 Malla Industrial Mod-4 Sku-13 13 9.00
14 14 Rodamiento Z-20 Mod-5 Sku-14 14 9.50
15 15 Válvula Flujo Mod-6 Sku-15 15 10.00
16 16 Tornillo Titanio Mod-7 Sku-16 16 10.50
Ln 142, Col 1 (687 selected) Spaces: 4 UTF-8 CRLF 100 rows MSSQL 00:00:00 tcp:jean3.database.windows.net : QhatuPERU (53)


File Edit View Help
Search
CONNECTIONS: AZURE
ALBERT JEANKARLO CHUQUIYA...
Azure for Students
SQL databases
cafesito (jean3)
master (jean3)
master (trabaj)
QhatuPERU (jean3)
Tables
dbo.ARTICULO
Columns
Keys
Constraints
Triggers
Indexes
Statistics
dbo.GUIA_DETALLE
dbo.GUIA_ENVIO
dbo.LINEA
dbo.ORDEN_COMPRA
dbo.ORDEN_DETALLE
dbo.PROVEEDOR
dbo.TIENDA
dbo.TRANSPORTISTA
Dropped Ledger Tables
Views
Synonyms
Programmability
External Resources
Storage
Security
trabajo_05 (jean3)
SQL servers
sconnected
SQLQuery_1.sql - (62) t...U (sa1)
SQLQuery_1 - (53) t...U (sa1)
Run
Cancel
Disconnect
Change
Database: QhatuPERU
Estimated Plan
Enable Actual Plan
Parse
-- 28. Mostrar solo los 3 artículos más caros por línea. (RANK/ROW_NUMBER con CTE y W
WITH ArticulosRankeados AS (
SELECT
CodArticulo,
DescripcionArticulo,
CodLinea,
PrecioProveedor,
-- Asigna un rango a los artículos dentro de cada línea, ordenados por precio
ROW_NUMBER() OVER (PARTITION BY CodLinea ORDER BY PrecioProveedor DESC) AS Ra
Results
Messages
CodArticulo
DescripcionArticulo
CodLinea
PrecioProveedor
85 85 Filtro Hepa Mod-6 Sku-85 85 45.00
86 86 Filtro Hepa Mod-7 Sku-86 86 45.50
87 87 Interruptor Termico Mod-8 Sk... 87 46.00
88 88 Filtro Hepa Mod-9 Sku-88 88 46.50
89 89 Rodamiento Z-20 Mod-10 Sku-89 89 47.00
90 90 Tornillo Titanio Mod-1 Sku-90 90 47.50
91 91 Rodamiento Z-20 Mod-2 Sku-91 91 48.00
92 92 Tornillo Titanio Mod-3 Sku-92 92 48.50
93 93 Interruptor Termico Mod-4 Sk... 93 49.00
94 94 Interruptor Termico Mod-5 Sk... 94 49.50
95 95 Cable Cat 6 Mod-6 Sku-95 95 50.00
96 96 Interruptor Termico Mod-7 Sk... 96 50.50
97 97 Sensor Óptico Mod-8 Sku-97 97 51.00
98 98 Malla Industrial Mod-9 Sku-98 98 51.50
99 99 Tornillo Titanio Mod-10 Sku-... 99 52.00
100 100 Válvula Flujo Mod-1 Sku-100 100 52.50
Ln 142, Col 1 (687 selected) Spaces: 4 UTF-8 CRLF 100 rows MSSQL 00:00:00 tcp:jean3.database.windows.net : QhatuPERU (53)

Explicación: Se utiliza la función **ROW_NUMBER()** con PARTITION BY CodLinea para asignar un número de fila ordenado por precio. La consulta exterior (SELECT) filtra el resultado usando la cláusula **WHERE RankPrecioLinea <= 3** para seleccionar solo los 3 artículos con la mejor posición (los más caros) en cada línea.

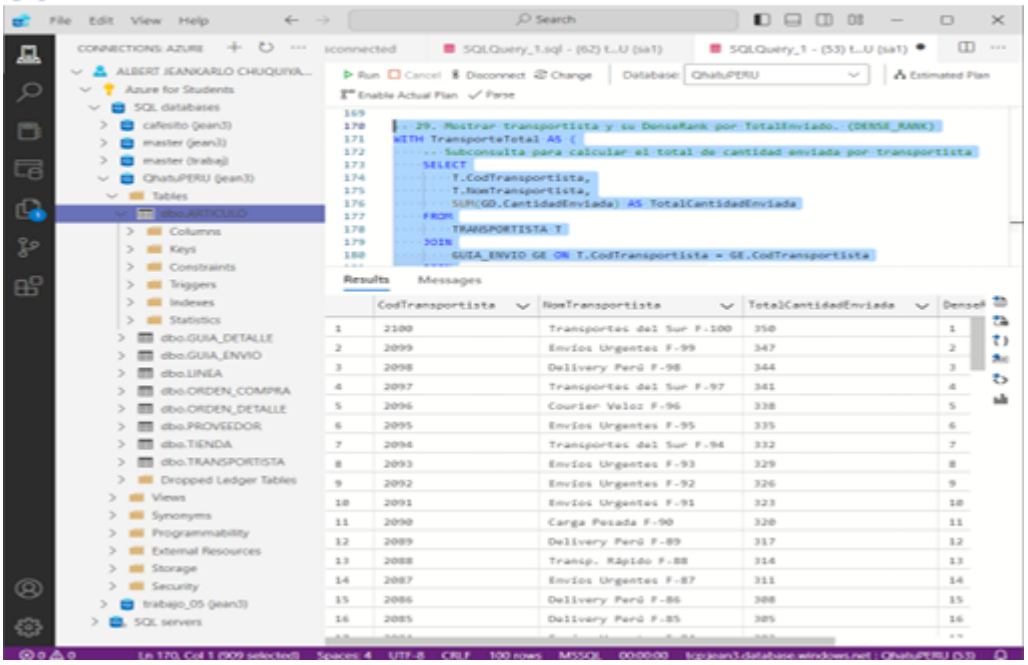
29. Mostrar transportista y su DenseRank por TotalEnviado.

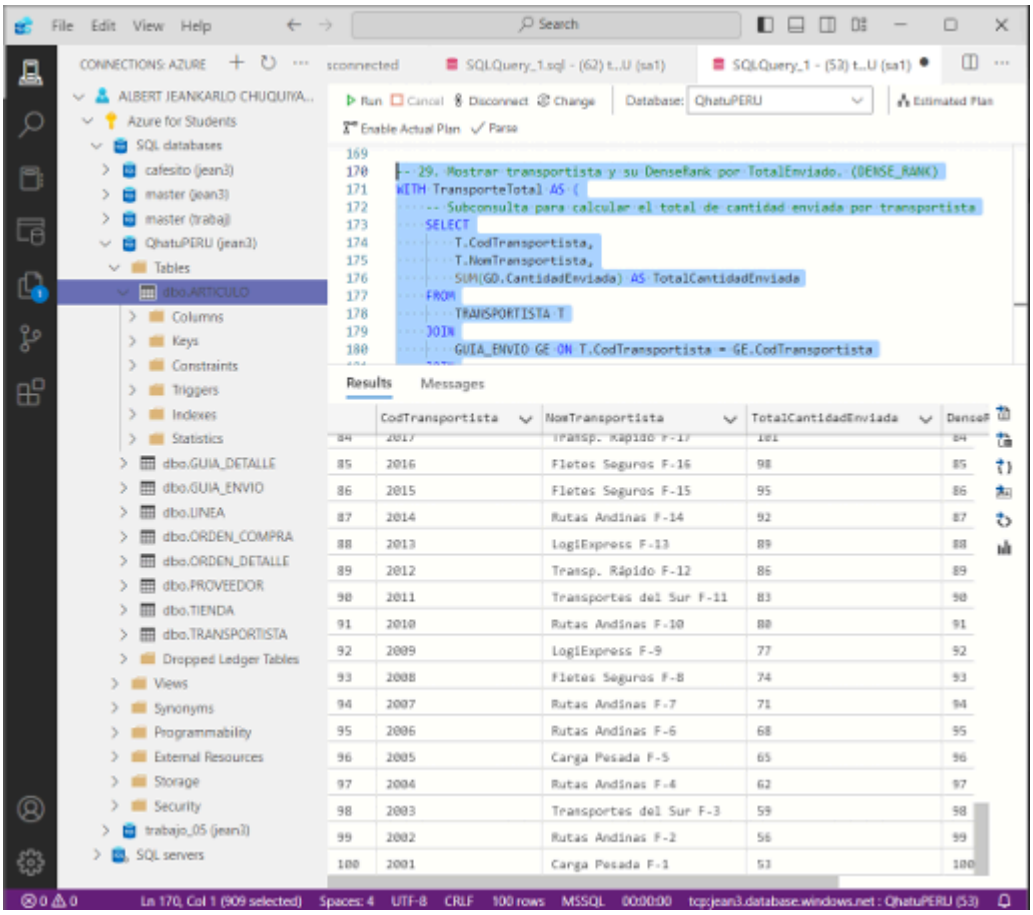
Enunciado: Calcular la cantidad total de artículos enviados por cada transportista y asignarles un rango de clasificación denso (DENSE_RANK) basado en esa cantidad.

Consulta SQL:

```
-- 29. Mostrar transportista y su DenseRank por TotalEnviado.
(DENSE_RANK)
WITH TransporteTotal AS (
    -- Subconsulta para calcular el total de cantidad enviada por
transportista
    SELECT
        T.CodTransportista,
        T.NomTransportista,
        SUM(GD.CantidadEnviada) AS TotalCantidadEnviada
    FROM
        TRANSPORTISTA T
    JOIN
        GUIA_ENVIO GE ON T.CodTransportista =
GE.CodTransportista
    JOIN
        GUIA_DETALLE GD ON GE.NumGuia = GD.NumGuia
    GROUP BY
        T.CodTransportista, T.NomTransportista
)
SELECT
    CodTransportista,
    NomTransportista,
    TotalCantidadEnviada,
    -- Asigna un rango denso al transportista según la cantidad total
enviada (empates tienen el mismo rango, no hay saltos)
    DENSE_RANK() OVER (ORDER BY TotalCantidadEnviada
DESC) AS DenseRankTotalEnviado
FROM
    TransporteTotal
```

```
ORDER BY
    DenseRankTotalEnviado;
GO
```





Explicación: La CTE calcula el total enviado por cada transportista. Luego, la función **DENSE_RANK()** se aplica para clasificar a los transportistas. A diferencia de RANK(), si hay empates, el siguiente rango no salta números, manteniendo una secuencia continua de clasificación.

30. Mostrar por guía la suma acumulada por tienda hasta esa guía (ordenada por FechaSalida).

Enunciado: Mostrar el valor total de cada guía de envío y calcular el valor acumulado total que esa guía ha representado para su respectiva tienda, ordenado por fecha de salida.

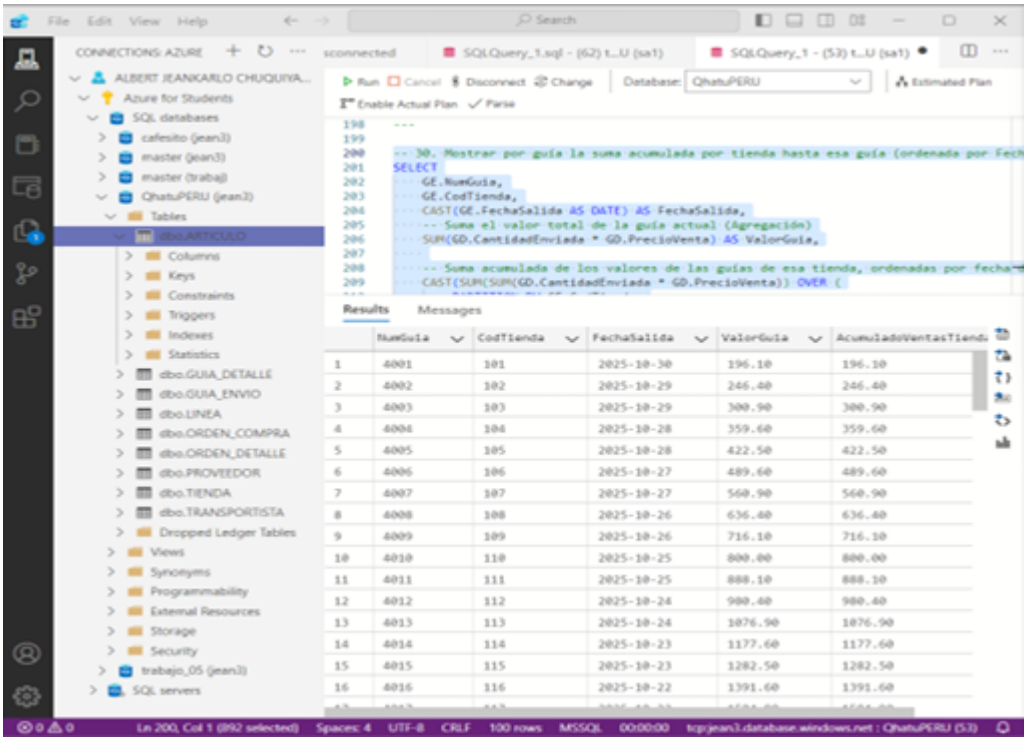
Consulta SQL:

```
-- 30. Mostrar por guía la suma acumulada por tienda hasta esa guía (ordenada por FechaSalida).
SELECT
    GE.NumGuia,
    GE.CodTienda,
    CAST(GE.FechaSalida AS DATE) AS FechaSalida,
    -- Suma el valor total de la guía actual
    SUM(GD.CantidadEnviada * GD.PrecioVenta) AS ValorGuia,
    -- Suma acumulada de los valores de las guías de esa tienda, ordenadas por fecha -- 30. Mostrar por guía la suma acumulada por tienda hasta esa guía (ordenada por FechaSalida).

SELECT
    GE.NumGuia,
    GE.CodTienda,
    CAST(GE.FechaSalida AS DATE) AS FechaSalida,
    -- Suma el valor total de la guía actual (Agregación)
    SUM(GD.CantidadEnviada * GD.PrecioVenta) AS ValorGuia,

    -- Suma acumulada de los valores de las guías de esa tienda, ordenadas por fecha de salida (Función de Ventana)
    CAST(SUM(SUM(GD.CantidadEnviada * GD.PrecioVenta))
OVER (
        PARTITION BY GE.CodTienda
        ORDER BY GE.FechaSalida
        ROWS BETWEEN UNBOUNDED PRECEDING AND
CURRENT ROW
    ) AS MONEY) AS AcumuladoVentasTienda
```

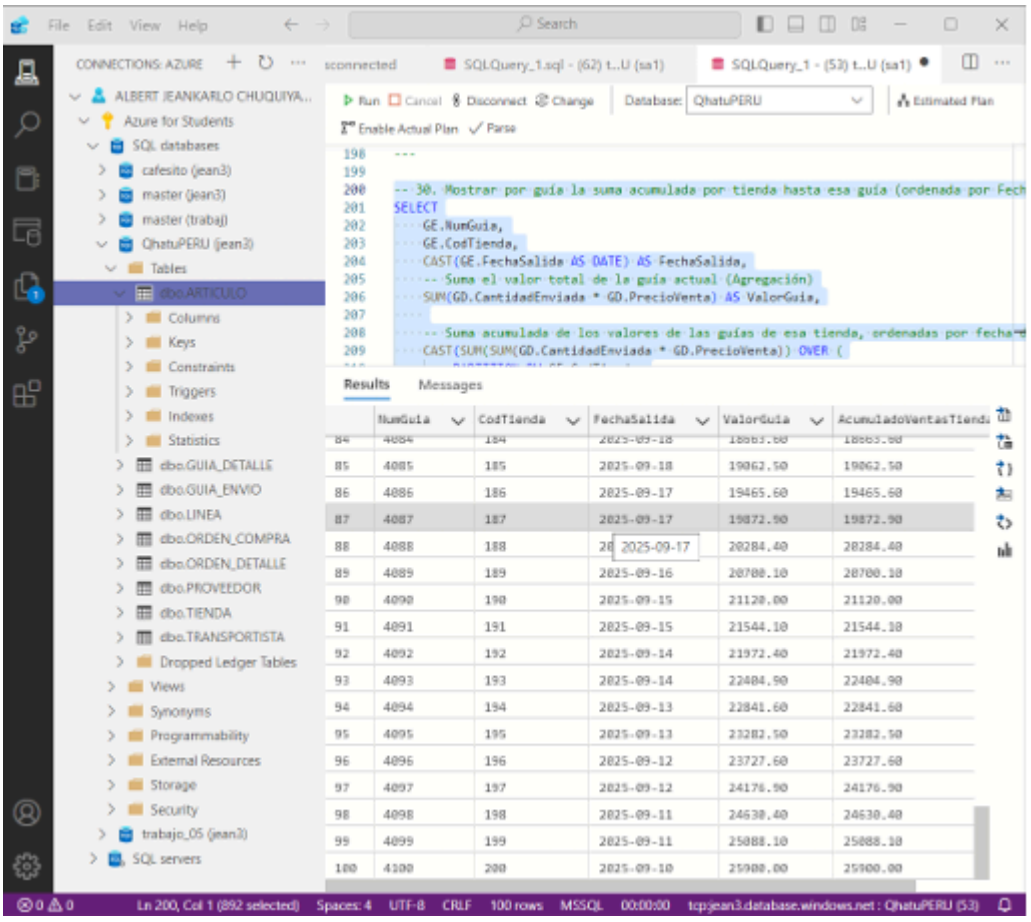
```
FROM
    GUIA_ENVIO GE
JOIN
    GUIA_DETALLE GD ON GE.NumGuia = GD.NumGuia
GROUP BY
    GE.NumGuia,
    GE.CodTienda,
    GE.FechaSalida -- <--- ¡CORRECCIÓN APLICADA AQUÍ!
ORDER BY
    GE.CodTienda, FechaSalida;
GO
de salida
    CAST(SUM(SUM(GD.CantidadEnviada * GD.PrecioVenta))
OVER (
        PARTITION BY GE.CodTienda
        ORDER BY GE.FechaSalida
        ROWS BETWEEN UNBOUNDED PRECEDING AND
CURRENT ROW
    ) AS MONEY) AS AcumuladoVentasTienda
FROM
    GUIA_ENVIO GE
JOIN
    GUIA_DETALLE GD ON GE.NumGuia = GD.NumGuia
GROUP BY
    GE.NumGuia, GE.CodTienda, CAST(GE.FechaSalida AS DATE)
ORDER BY
    GE.CodTienda, FechaSalida;
GO
```



```
-- 30. Mostrar por guía la suma acumulada por tienda hasta esa guía (ordenada por Fech
SELECT
    GE.NumGuia,
    GE.CodTienda,
    CAST(GE.FechaSalida AS DATE) AS FechaSalida,
    -- Suma el valor total de la guía actual (Agregación)
    SUM(GD.CantidadEnviada * GD.PrecioVenta) AS ValorGuia,

    -- Suma acumulada de los valores de las guías de esa tienda, ordenadas por fecha
    CAST(SUM(SUM(GD.CantidadEnviada * GD.PrecioVenta)) OVER (
```

	NumGuia	CodTienda	FechaSalida	ValorGuia	AcumuladoVentasTienda
1	4001	101	2025-10-30	196.10	196.10
2	4002	102	2025-10-29	246.40	246.40
3	4003	103	2025-10-29	300.90	300.90
4	4004	104	2025-10-28	359.60	359.60
5	4005	105	2025-10-28	422.50	422.50
6	4006	106	2025-10-27	489.60	489.60
7	4007	107	2025-10-27	560.90	560.90
8	4008	108	2025-10-26	636.40	636.40
9	4009	109	2025-10-26	716.10	716.10
10	4010	110	2025-10-25	800.00	800.00
11	4011	111	2025-10-25	888.10	888.10
12	4012	112	2025-10-24	980.40	980.40
13	4013	113	2025-10-24	1076.90	1076.90
14	4014	114	2025-10-23	1177.60	1177.60
15	4015	115	2025-10-23	1282.50	1282.50
16	4016	116	2025-10-22	1391.60	1391.60



```
-- 30. Mostrar por guía la suma acumulada por tienda hasta esa guía (ordenada por Fech
SELECT
    GE.NumGuia,
    GE.CodTienda,
    CAST(GE.FechaSalida AS DATE) AS FechaSalida,
    -- Suma el valor total de la guía actual (Agregación)
    SUM(GD.CantidadEnviada * GD.PrecioVenta) AS ValorGuia,

    -- Suma acumulada de los valores de las guías de esa tienda, ordenadas por fecha
    CAST(SUM(SUM(GD.CantidadEnviada * GD.PrecioVenta)) OVER (
```

	NumGuia	CodTienda	FechaSalida	ValorGuia	AcumuladoVentasTienda
85	4085	185	2025-09-18	19062.50	19062.50
86	4086	186	2025-09-17	19465.60	19465.60
87	4087	187	2025-09-17	19872.90	19872.90
88	4088	188	2025-09-17	20284.40	20284.40
89	4089	189	2025-09-16	20700.10	20700.10
90	4090	190	2025-09-15	21120.00	21120.00
91	4091	191	2025-09-15	21544.10	21544.10
92	4092	192	2025-09-14	21972.40	21972.40
93	4093	193	2025-09-14	22404.90	22404.90
94	4094	194	2025-09-13	22841.60	22841.60
95	4095	195	2025-09-13	23282.50	23282.50
96	4096	196	2025-09-12	23727.60	23727.60
97	4097	197	2025-09-12	24176.90	24176.90
98	4098	198	2025-09-11	24630.40	24630.40
99	4099	199	2025-09-11	25088.10	25088.10
100	4100	200	2025-09-10	25900.00	25900.00

Explicación: La consulta primero agrupa para obtener el ValorGuia. Luego, usa la función **SUM()** anidada con OVER. El **PARTITION BY CodTienda** reinicia el cálculo acumulado para cada tienda, y el marco de ventana **ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW** asegura que se sumen todas las guías de esa tienda desde la primera fecha hasta la fecha de la guía actual.