

~~HENRY~~



DATA SCIENCE

SUBCONSULTAS, **VISTAS Y** FUNCIONES VENTANA



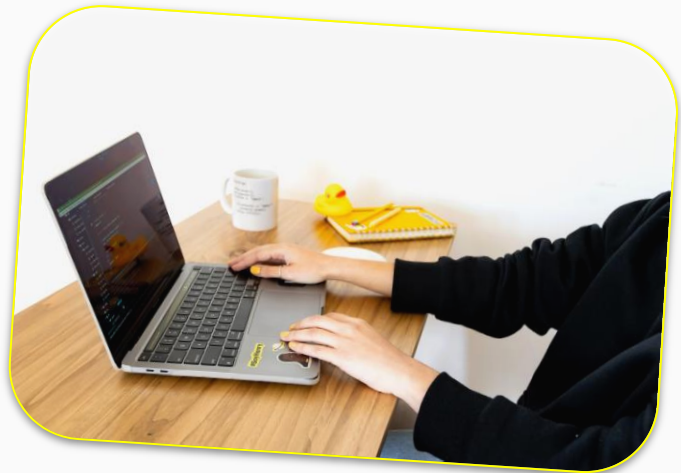


objetivos de la clase

- ✓ **Diferenciar** el concepto de Subconsultas y Vistas
- ✓ **Comprender** en qué casos son útiles las Funciones Ventana



Agenda



➤ Subconsultas

➤ Vistas

➤ Función ventanas



subconsultas





¿Qué son?

Una **consulta secundaria** es una consulta de selección que está contenida dentro de otra consulta.

La consulta de selección interna generalmente se usa para determinar los resultados de la consulta de selección externa. Consiste en utilizar los resultados de **una consulta dentro de otra**, que se considera la principal.





Es una sentencia **SELECT** anidada dentro de una instrucción **SELECT, SELECT...INTO, INSERT...INTO, DELETE, o UPDATE** o dentro de otra subconsulta.





Subconsultas

Una subconsulta tiene la misma sintaxis que una sentencia SELECT exceptuando que aparece encerrada entre paréntesis, no puede contener la cláusula ORDER BY, ni puede ser la **UNION** de varias sentencias SELECT.



PASO A PASO

Subconsultas

1 Las **subconsultas** son un proceso de selección interno, y se pueden utilizar en cualquier sentencia que permita una expresión

- 2**
- **SELECT**: Para calcular y crear un nuevo campo virtual a la consulta principal.
 - **FROM**: Para devolver una tabla secundaria calculada o un campo calculado con un contexto diferente.

- 3**
- **WHERE**: Para definir filtros compuestos calculados. Si se conociera el valor a calcular con la subconsulta, se utilizará ese valor.



Listas --> IN , NOT IN.

Valor único --> = , >= , <= , etc.





vistas





✦ ¿Qué son? ✦

Es un mecanismo que permite **almacenar de forma permanente** una consulta en SQL. A su vez esta consulta almacenada en la vista se puede acceder como si fuera una tabla, denominándose a la vista como una tabla virtual.

Cabe destacar, que lo que se almacena es la consulta, no los datos de sus resultados.



Las vistas se componen de **campos** y **filas** provenientes del resultado de la consulta, las cuales pueden venir de varias tablas



Al igual que con los otros objetos que forman parte de la base de datos se crean mediante la sentencia **CREATE** y se eliminan mediante **DROP**. No se les aplica **INSERT** o **UPDATE**, dado que se trata de la consulta, no los datos.

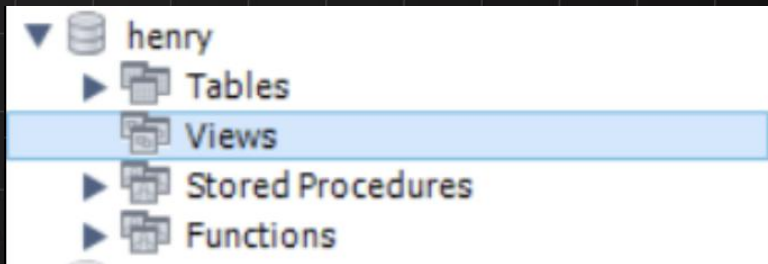


```
-- Crear una vista.  
CREATE VIEW primerosAlumnos AS  
SELECT idAlumno, fechaIngreso  
FROM alumnos  
WHERE fechaIngreso = ( SELECT MIN(fechaIngreso) AS fecha  
                        FROM alumnos)  
  
-- Obtener los resultados de una vista.  
SELECT *  
FROM primerosAlumnos  
  
-- Modificar una vista.  
ALTER VIEW primerosAlumnos AS  
SELECT idAlumno, CONCAT(apellido," ",nombre), fechaIngreso  
FROM alumnos  
WHERE fechaIngreso = ( SELECT MIN(fechaIngreso) AS fecha  
                        FROM alumnos)  
  
-- Eliminar una vista  
DROP VIEW primerosAlumnos
```



Vistas

Al crear una vista, esta queda alojada en la base de datos correspondiente y se pueden ver en la interfaz del gestor de base de datos.





Una vista actúa como **filtro** de las tablas subyacentes a las que se hace referencia en ella.

La consulta que define la vista puede provenir de una o de varias tablas, o bien de otras vistas de la **base de datos** actual u otras bases de datos.



Asimismo, es posible utilizar las consultas distribuidas para definir vistas que utilicen datos de orígenes heterogéneos. Esto puede resultar de utilidad, por ejemplo, si desea **combinar datos** de estructura similar que proceden de distintos servidores, cada uno de los cuales almacena los datos para una región distinta de la organización.



Ventajas

- **Permite** centrar, simplificar y personalizar la forma de mostrar la información a cada usuario.



- Se usa como mecanismo de seguridad, el cual permite a los usuarios obtener acceso a la información proveniente de la vista sin acceder a otras opciones.
- **Proporciona** una sintaxis simple para acceder a los resultados de la vista



Función ventana

Puede entenderse como un conjunto de registros y una función que se ejecuta sobre los mismos y cumple determinadas condiciones. Para cada registro se debe ejecutar una función en esta ventana.

Las funciones ventana evita los JOIN de una tabla consigo misma:



-- Promedio de ventas por Fecha:

```
SELECT  Fecha,  
        AVG(Precio * Cantidad) AS Promedio_Ventas  
FROM venta  
GROUP BY Fecha;
```

-- Unimos el promedio de ventas por fecha con las ventas por fecha:

```
SELECT  v.Fecha,  
        v.Precio * v.Cantidad AS Venta,  
        v2.Promedio_Ventas  
FROM    venta v JOIN ( SELECT  Fecha,  
                                AVG(Precio * Cantidad) AS Promedio_Ventas  
                                FROM venta  
                                GROUP BY Fecha) v2  
ON (v.Fecha = v2.Fecha);
```

¿Cómo podría una función ventana ayudar en este caso?



```
SELECT  v.Fecha,  
        v.Precio * v.Cantidad AS Venta,  
        AVG(v.Precio * v.Cantidad) OVER (PARTITION BY v.Fecha) AS Promedio_Ventas  
FROM venta v;
```



```
SELECT v.Fecha,  
       v.Precio * v.Cantidad AS Venta,  
       AVG(v.Precio * v.Cantidad) OVER (PARTITION BY v.Fecha) AS Promedio_Ventas  
FROM venta v;
```

Fecha	Venta	Promedio_Ventas
2015-01-01	2355.000	6014.4992308
2015-01-01	4710.000	6014.4992308
2015-01-01	321.000	6014.4992308
2015-01-01	321.000	6014.4992308
2015-01-01	642.000	6014.4992308
2015-01-01	1638.120	6014.4992308
2015-01-01	1638.120	6014.4992308
2015-01-01	1638.120	6014.4992308
2015-01-01	1638.120	6014.4992308
2015-01-01	2364.000	6014.4992308
2015-01-01	3546.000	6014.4992308
2015-01-02	356.000	1705.6317647
2015-01-02	178.000	1705.6317647
2015-01-02	534.000	1705.6317647
2015-01-02	356.000	1705.6317647
2015-01-02	1030.000	1705.6317647
2015-01-02	1030.000	1705.6317647
2015-01-02	1030.000	1705.6317647
2015-01-02	1545.000	1705.6317647
2015-01-02	1030.000	1705.6317647
2015-01-02	515.000	1705.6317647
2015-01-02	515.000	1705.6317647

La función ventana se puede descomponer en las siguientes partes:



- **OVER:** Define una ventana o conjunto de filas que debe utilizar una función ventana, incluyendo cualquier orden. No está restringido por sí mismo, e incluye todas las filas. Se pueden usar múltiples cláusulas OVER en una sola consulta, cada una con su propio particionamiento y ordenación si es necesario.
- **Cláusula de partición (PARTITION BY):** Las ventanas se agrupan según esos campos y las funciones de ventana se ejecutan en diferentes grupos.
- **Orden por cláusula (ORDER BY):** Según los campos a ordenar, la función de ventana enumerará los registros según el orden de clasificación. Se puede utilizar junto con la cláusula de partición o solo.



La función ventana se puede descomponer en las siguientes partes:

- **Cláusula de marco:** El marco es un subconjunto de la partición actual. La cláusula se usa para definir las reglas del subconjunto y generalmente se usa como una ventana deslizante.
 - ❑ **UNBOUNDED** significa ir todo el camino al límite en la dirección especificada por **PRECEDING** o **FOLLOWING** (comienzo o final)
 - ❑ **CURRENT ROW** indica inicio o final en la fila actual en la partición
 - ❑ **ROWS BETWEEN** permite definir un rango de filas entre dos puntos



Ejemplos función ventana





Se requiere visualizar una tabla con el acumulado de la venta por fecha, en este caso, se hace uso del marco "ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW" para tomar dentro de la partición, las filas desde el inicio de la misma hasta la fila actual:

```
SELECT  v.Fecha,  
        v.Precio * v.Cantidad AS Venta,  
        SUM(v.Precio * v.Cantidad) OVER (PARTITION BY v.Fecha ROWS BETWEEN UNBOUNDED PRECEDING AND  
CURRENT ROW) AS Total_Ventas  
FROM venta v;
```



Notar, que para este caso, se consigue un resultado similar con la cláusula "ORDER BY"



```
SELECT v.Fecha,  
       v.Precio * v.Cantidad AS Venta,  
       SUM(v.Precio * v.Cantidad) OVER (PARTITION BY v.Fecha ORDER BY v.IdVenta) AS Total_Ventas  
FROM venta v;
```

Fecha	Venta	Total_Ventas
2015-01-10	1637.680	161342.300
2015-01-10	1637.680	162979.980
2015-01-10	1637.680	164617.660
2015-01-10	17407.280	182024.940
2015-01-12	2475.000	2475.000
2015-01-12	3712.500	6187.500
2015-01-12	3712.500	9900.000
2015-01-12	1024.000	10924.000
2015-01-12	1024.000	11948.000
2015-01-12	2509.760	14457.760
2015-01-12	3764.640	18222.400
2015-01-12	2509.760	20732.160
2015-01-12	706.640	21438.800
2015-01-12	387684....	409122.800
2015-01-13	2978.000	2978.000
2015-01-13	2978.000	5956.000
2015-01-13	1062.000	7018.000
2015-01-13	708.000	7726.000
2015-01-13	7672.500	15398.500
2015-01-13	7672.500	23071.000
2015-01-13	5115.000	28186.000
2015-01-13	5115.000	33301.000
2015-01-13	2557.500	35858.500



Se requiere visualizar por fechas, el ranking de las ventas ordenadas de mayor a menor, notemos el uso de las cláusulas PARTITION BY y ORDER BY con el agregado de DESC, para las ventas mayores sean las que tengan mejor ranking.



```
SELECT RANK() OVER (PARTITION BY v.Fecha ORDER BY v.Precio * v.Cantidad DESC) AS Ranking_Venta,  
       v.Fecha,  
       v.IdCliente,  
       v.Precio,  
       v.Cantidad,  
       (v.Precio * Cantidad) as Venta  
FROM venta v;
```

Ranking_Venta	Fecha	IdCliente	Precio	Cantidad	Venta
25	2015-01-10	838	457.820	2	915.640
26	2015-01-10	540	543.180	1	543.180
26	2015-01-10	540	543.180	1	543.180
1	2015-01-12	2765	1254.880	3	3764.640
2	2015-01-12	328	1237.500	3	3712.500
2	2015-01-12	328	1237.500	3	3712.500
4	2015-01-12	2765	1254.880	2	2509.760
4	2015-01-12	2765	1254.880	2	2509.760
6	2015-01-12	328	1237.500	2	2475.000
7	2015-01-12	1476	512.000	2	1024.000
7	2015-01-12	1476	512.000	2	1024.000
9	2015-01-12	783	353.320	2	706.640
1	2015-01-13	1370	2802.800	3	8408.400
2	2015-01-13	548	4141.000	2	8282.000



La función RANK() tiene la particularidad de que utiliza un salto o gap entre los registros, para notar lo que pasa en la consulta anterior, con el puesto 4 del ranking. El cuarto aparece dos veces, debido a que son ventas con el mismo valor, y luego el siguiente valor de ranking es el 4. Con el mismo 4 y con el 7, vuelve a pasar lo mismo:

Ranking_Venta	Fecha	IdCliente	Precio	Cantidad	Venta
25	2015-01-10	838	457.820	2	915.640
26	2015-01-10	540	543.180	1	543.180
26	2015-01-10	540	543.180	1	543.180
1	2015-01-12	2765	1254.880	3	3764.640
2	2015-01-12	328	1237.500	3	3712.500
2	2015-01-12	328	1237.500	3	3712.500
4	2015-01-12	2765	1254.880	2	2509.760
4	2015-01-12	2765	1254.880	2	2509.760
6	2015-01-12	328	1237.500	2	2475.000
7	2015-01-12	1476	512.000	2	1024.000
7	2015-01-12	1476	512.000	2	1024.000
9	2015-01-12	783	353.320	2	706.640
1	2015-01-13	1370	2802.800	3	8408.400
2	2015-01-13	548	4141.000	2	8282.000



Quizás querramos ver esto de otra manera, para lo cual, deberíamos usar la función DENSE_RANK():



```
SELECT DENSE_RANK( ) OVER (PARTITION BY v.Fecha ORDER BY v.Precio * v.Cantidad DESC) AS Ranking_Venta,  
       v.Fecha,  
       v.IdCliente,  
       v.Precio,  
       v.Cantidad,  
       (v.Precio * Cantidad) as Venta  
FROM venta v;
```







Ranking_Venta	Fecha	IdCliente	Precio	Cantidad	Venta
12	2015-01-10	838	457.820	2	915.640
13	2015-01-10	540	543.180	1	543.180
13	2015-01-10	540	543.180	1	543.180
1	2015-01-12	2765	1254.880	3	3764.640
2	2015-01-12	328	1237.500	3	3712.500
2	2015-01-12	328	1237.500	3	3712.500
3	2015-01-12	2765	1254.880	2	2509.760
3	2015-01-12	2765	1254.880	2	2509.760
4	2015-01-12	328	1237.500	2	2475.000
5	2015-01-12	1476	512.000	2	1024.000
5	2015-01-12	1476	512.000	2	1024.000
6	2015-01-12	783	353.320	2	706.640
1	2015-01-13	1370	2802.800	3	8408.400



Qué pasa si ahora, se quieren mostrar solamente las 3 ventas más altas por cada fecha. Para esto, sí vamos a acudir a una subconsulta, ya que no es posible utilizar la cláusula WHERE con las funciones ventana, debido a que la función ventana, ejecuta en el último paso, justo antes del ORDER BY, esto es parte de lo que le da su buena performance.

```
SELECT *
FROM ( SELECT DENSE_RANK() OVER (PARTITION BY v.Fecha ORDER BY v.Precio * v.Cantidad DESC) AS Ranking_Venta,
        v.Fecha,
        v.IdCliente,
        v.Precio,
        v.Cantidad,
        (v.Precio * Cantidad) as Venta
      FROM venta v) ventas
WHERE Ranking_Venta < 4;
```



Ranking_Venta	Fecha	IdCliente	Precio	Cantidad	Venta
1	2015-01-01	1868	2355.000	2	4710.000
2	2015-01-01	1315	1182.000	3	3546.000
3 	2015-01-01	1432	906.180	3	2718.540
1	2015-01-02	3146	2807.640	3	8422.920
2	2015-01-02	3146	2807.640	2	5615.280
2	2015-01-02	3146	2807.640	2	5615.280
3 	2015-01-02	186	1765.000	3	5295.000
1 	2015-01-03	1447	815.000	2	1630.000
1	2015-01-05	2689	1182.000	3	3546.000
1	2015-01-05	2689	1182.000	3	3546.000
2	2015-01-05	2689	1182.000	1	1182.000
3	2015-01-05	885	369.820	3	1109.460
3 	2015-01-05	885	369.820	3	1109.460



Al listado anterior, agregamos el requerimiento de que sea para la sucursal con id = 12 y además que se muestre el porcentaje acumulado, haciendo uso de la función PERCENT_RANK():

```
SELECT *
FROM ( SELECT DENSE_RANK() OVER (PARTITION BY v.Fecha ORDER BY v.Precio * v.Cantidad DESC) AS Ranking_Venta,
        v.Fecha,
        v.IdCliente,
        v.Precio,
        v.Cantidad,
        (v.Precio * Cantidad) as Venta
      FROM venta v) ventas
WHERE Ranking_Venta < 4;
```



Tener en cuenta que $\text{PERCENT_RANK} = (\text{ranking} - 1) / (\text{cantidad filas} - 1)$

Ranking_Venta	Ranking_Venta_Porcentaje	Fecha	IdCliente	Precio	Cantidad	Venta
1	0	2015-01-13	548	4141.000	2	8282.000
1	0	2015-01-13	548	4141.000	2	8282.000
1	0	2015-01-13	548	4141.000	2	8282.000
1	0	2015-01-13	548	4141.000	2	8282.000
1	0	2015-01-15	969	1819.180	3	5457.540
2	0.3333333333333333	2015-01-15	969	1819.180	2	3638.360
2	0.3333333333333333	2015-01-15	969	1819.180	2	3638.360
2	0.3333333333333333	2015-01-15	969	1819.180	2	3638.360
1	0	2015-01-16	512	560.000	2	1120.000
1	0	2015-01-20	1127	373.000	3	1119.000
2	0.5	2015-01-20	1127	373.000	2	746.000
3	1	2015-01-20	1127	373.000	1	373.000
1	0	2015-01-28	2975	1765.000	3	5295.000
2	1	2015-01-28	2975	1765.000	2	3530.000



Se requiere ver el listado de clientes numerado, particionando por Localidad.
Para esto, es posible hacer uso de la función ROW_NUMBER()

```
SELECT ROW_NUMBER() OVER(PARTITION BY c.IdLocalidad) AS row_id,  
       c.Nombre_Y_Apellido,  
       c.Domicilio,  
       c.Edad,  
       c.IdLocalidad  
FROM cliente c;
```

row_id	Nombre_Y_Apellido	Domicilio	Edad	IdLocalidad
1	Caballero, Mario Ernesto	El Parque Y San Luis S/n	47	1
1	Luis Maria Repiso	Arroyo Paycarabi S/N Paraje Islas 2â° Seccion	58	2
1	Gerardo Pedro Rey	Rio La Barquita S/N Paraje Islas 3â° Seccion	37	3
1	Beatriz Scapusio	Autopista Ricchieri Y Ruta Nâ°205 S/N	63	4
2	Acosta, Ernesto Cecilio	Ing. Huergo E/Pinzon Y Colon S/N Transradio	32	4
3	Plana, Ernesto Alejandro	Pinzon Esq. Del Progreso 1896 Las Torres	36	4
4	Pacheco, Roberto Marcelo	Copacabana E/ Necochea Y M. Del Plata 1455 ...	43	4
5	Rey, Silvia Beatriz	Fortunato Lopez 929	39	4
6	Castro Salinas, Pedro Juan	German Palleros 866 E. Echeverria	40	4
7	Rodriguez, Carlos Alberto	Pluton Esq. Saturno 4476 L. De Beheram	47	4
8	Garcia, Luis Hector	Sierra De Ambato E/ Vernet Y Benavidez 179	40	4
9	María Del Carmen Corujo	Guamini E/ Salta Y Av. De Mayo 5348 Las Torres	45	4
10	Jose Cosentino	Colonia Monte Grande Y Sta.Magdalena S/N Ba...	23	4
1	Mariela Sasson	236 (E/ 496 Y 498) Ruta 2 Km. 52 S/N La Rueda	61	5
2	Sione, Sebastian	207 E/516 Y 516 Bis S/n	25	5



Agreguemos ahora, el primer y el último nombre de los clientes, haciendo uso de las funciones FIRST_VALUE() y LAST_VALUE()

```
SELECT ROW_NUMBER() OVER(PARTITION BY c.IdLocalidad) AS row_id,  
       FIRST_VALUE(Nombre_Y_Apellido) OVER(PARTITION BY c.IdLocalidad) AS primer_nombre,  
       LAST_VALUE(Nombre_Y_Apellido) OVER(PARTITION BY c.IdLocalidad) AS ultimo_nombre,  
       c.Nombre_Y_Apellido,  
       c.Domicilio,  
       c.Edad,  
       c.IdLocalidad  
FROM cliente c;
```

row_id	primer_nombre	ultimo_nombre	Nombre_Y_Apellido	Domicilio	Edad	IdLocalidad
1	Caballero, Mario Ernesto	Caballero, Mario Ernesto	Caballero, Mario Ernesto	El Parque Y San Luis S/n	47	1
1	Luis Maria Repiso	Luis Maria Repiso	Luis Maria Repiso	Arroyo Paycarabi S/N Paraje Islas 28° Seccion	58	2
1	Gerardo Pedro Rev	Gerardo Pedro Rev	Gerardo Pedro Rev	Rio La Barquita S/N Paraje Islas 33° Seccion	37	3
1	Beatriz Scapusio	Jose Cosentino	Beatriz Scapusio	Autopista Ricchieri Y Ruta Nã°205 S/N	63	4
2	Beatriz Scapusio	Jose Cosentino	Acosta, Ernesto Cecilio	Ing. Huergo E/Pinzon Y Colon S/N Transradio	32	4
3	Beatriz Scapusio	Jose Cosentino	Plana, Ernesto Alejandro	Pinzon Esq. Del Progreso 1896 Las Torres	36	4
4	Beatriz Scapusio	Jose Cosentino	Pacheco, Roberto Marcelo	Copacabana E/ Necochea Y M. Del Plata 1455 ...	43	4
5	Beatriz Scapusio	Jose Cosentino	Rey, Silvia Beatriz	Fortunato Lopez 929	39	4
6	Beatriz Scapusio	Jose Cosentino	Castro Salinas, Pedro Juan	German Palleros 866 E. Echeverria	40	4
7	Beatriz Scapusio	Jose Cosentino	Rodriguez, Carlos Alberto	Pluton Esq. Saturno 4476 L. De Beheram	47	4
8	Beatriz Scapusio	Jose Cosentino	Garcia, Luis Hector	Sierra De Ambato E/ Vernet Y Benavidez 179	40	4
9	Beatriz Scapusio	Jose Cosentino	María Del Carmen Corujo	Guamini E/ Salta Y Av. De Mayo 5348 Las Torres	45	4
10	Beatriz Scapusio	Jose Cosentino	Jose Cosentino	Colonia Monte Grande Y Sta.Magdalena S/N Ba...	23	4
1	Mariela Sasson	Sione, Sebastian	Mariela Sasson	236 (E/ 496 Y 498) Ruta 2 Km. 52 S/N La Ruëda	61	5
2	Mariela Sasson	Sione, Sebastian	Sione, Sebastian	207 E/ 516 Y 516 Bis S/n	25	5



A la consulta anterior, agregamos la necesidad de ver el *enésimo* nombre de los clientes, haciendo uso de la función `NTH_VALUE(<posición>)`:

```
SELECT ROW_NUMBER() OVER(PARTITION BY c.IdLocalidad) AS row_id,  
       FIRST_VALUE(Nombre_Y_Apellido) OVER(PARTITION BY c.IdLocalidad) AS primer_nombre,  
       LAST_VALUE(Nombre_Y_Apellido) OVER(PARTITION BY c.IdLocalidad) AS ultimo_nombre,  
       NTH_VALUE(Nombre_Y_Apellido, 3) OVER(PARTITION BY c.IdLocalidad) AS ultimo_nombre,  
       c.Nombre_Y_Apellido,  
       c.Domicilio,  
       c.Edad,  
       c.IdLocalidad  
FROM cliente c;
```

Ranking_Venta	Ranking_Venta_Porcentaje	Fecha	IdCliente	Precio	Cantidad	Venta
1	0	2015-01-13	548	4141.000	2	8282.000
1	0	2015-01-13	548	4141.000	2	8282.000
1	0	2015-01-13	548	4141.000	2	8282.000
1	0	2015-01-13	548	4141.000	2	8282.000
1	0	2015-01-15	969	1819.180	3	5457.540
2	0.3333333333333333	2015-01-15	969	1819.180	2	3638.360
2	0.3333333333333333	2015-01-15	969	1819.180	2	3638.360
2	0.3333333333333333	2015-01-15	969	1819.180	2	3638.360
1	0	2015-01-16	512	560.000	2	1120.000
1	0	2015-01-20	1127	373.000	3	1119.000
2	0.5	2015-01-20	1127	373.000	2	746.000
3	1	2015-01-20	1127	373.000	1	373.000
1	0	2015-01-28	2975	1765.000	3	5295.000
2	1	2015-01-28	2975	1765.000	2	3530.000



Se requiere ver un listado con el detalle de las ventas para cada cliente, que contenga la cantidad de días transcurridos entre operación y operación de venta, para lo cual, es útil la función LEAD() que trae el valor que contiene ese campo en el registro anterior, según la partición y el orden que se le de. La función LAG() obtiene el valor que contiene el registro siguiente:

```
SELECT  ROW_NUMBER() OVER(PARTITION BY v.IdCliente ORDER BY v.Fecha) AS operacion,  
        v.IdCliente,  
        LAG(v.Fecha) OVER(PARTITION BY v.IdCliente ORDER BY v.Fecha) AS Fecha_Anterior,  
        v.Fecha,  
        LEAD(v.Fecha) OVER(PARTITION BY v.IdCliente ORDER BY v.Fecha) AS Fecha_Siguiente,  
        DATEDIFF(LEAD(v.Fecha) OVER(PARTITION BY v.IdCliente ORDER BY v.Fecha), v.Fecha) AS Diferencia_Ste_Venta,  
        (v.Precio * v.Cantidad) AS Venta  
FROM venta v;
```




operacion	IdCliente	Fecha_Anterior	Fecha	Fecha_Siguiente	Diferencia_Ste_Venta	Venta
1	1	NULL	2015-10-10	2015-10-10	0	642.000
2	1	2015-10-10	2015-10-10	2015-10-10	0	642.000
3	1	2015-10-10	2015-10-10	2015-11-04	25	321.000
4	1	2015-10-10	2015-11-04	2016-06-06	215	1798.500
5	1	2015-11-04	2016-06-06	2016-12-30	207	996.000
6	1	2016-06-06	2016-12-30	2016-12-30	0	1494.000
7	1	2016-12-30	2016-12-30	2016-12-30	0	1494.000
8	1	2016-12-30	2016-12-30	2017-03-22	82	1494.000
9	1	2016-12-30	2017-03-22	2017-03-22	0	904.420
10	1	2017-03-22	2017-03-22	2018-06-19	454	904.420
11	1	2017-03-22	2018-06-19	2018-06-19	0	5482.000
12	1	2018-06-19	2018-06-19	2018-06-19	0	5482.000
13	1	2018-06-19	2018-06-19	2019-01-22	217	8223.000
14	1	2018-06-19	2019-01-22	2019-01-22	0	563.000



Con el listado anterior, ahora es necesario obtener el promedio de días que transcurren, por cliente, entre operación y operación de venta:

```
SELECT IdCliente,
       ROUND(AVG(Diferencia_Ste_Venta),0) AS Promedio_Dias
FROM (
    SELECT v.IdCliente,
           DATEDIFF(LEAD(v.Fecha) OVER(PARTITION BY v.IdCliente ORDER BY v.Fecha), v.Fecha) AS Diferencia
    FROM venta v) vta
GROUP BY IdCliente;
```

IdCliente	Promedio_Dias
1	69
2	70
3	57
4	69
5	91
6	109
7	102
8	96
9	62
10	311
11	83
12	60
13	188
14	107



Tambien, es posible definir las ventanas con un alias:

```
SELECT IdCliente,  
       ROUND(AVG(Diferencia_Ste_Venta),0) AS Promedio_Dias  
FROM (  
    SELECT v.IdCliente,  
           DATEDIFF(LEAD(v.Fecha) OVER w, v.Fecha) AS Diferencia_Ste_Venta  
    FROM venta v  
    WINDOW w AS (PARTITION BY v.IdCliente ORDER BY v.Fecha)) vta  
GROUP BY IdCliente;
```

HENRY



Próxima lecture
**De datos a
conocimiento**





**¡Muchas
gracias!**