SQL Avanzado

WITH recursive

- La cláusula WITH RECURSIVE permite encontrar la **clausura transitiva** de una consulta
- Dada una tabla T que es *input* de una consulta, permite que el resultado de la misma:

$$T_{new} \leftarrow subquery(T)$$

Sea utilizado en el lugar de T para volver a ejecutar la misma consulta.

Esto se repite hasta encontrar un punto fijo. Ej: T = subquery(T)

WITH RECURSIVE $T[(A_1, A_2, ..., A_n)]$
AS (<initial_value_query>
UNION <subquery>)
<query with T>;

WITH recursivo 1: Pseudocódigo

$T_0 = initial\_value\_query(R_1, R_2, ...)$;
$T_{new} = T_0$;
do
  $T = T_{new}$;
  $T_{new} = T_0 \cup subquery(T, R_1, R_2, ...)$;
while $T_{new} \neq T$;
return $query(T, ...)$;

- **Importante:** la consulta <initial_value_query> no puede depender de T
- Tanto en WITH como en WITH RECURSIVE puede definirse más de una tabla auxiliar antes de la consulta

Ejemplo

Dada la relación Vuelos(codVuelo, ciudadDesde, ciudadHasta) que indica todos los vuelos que ofrece una aerolínea, encuentre todas las ciudades que son 'alcanzables' desde París, independientemente de la cantidad de escalas que sea necesario hacer.

```
WITH RECURSIVE DestinosAlcanzables(ciudad)
AS (VALUES ('Paris')
    UNION
    SELECT v.ciudadHasta AS ciudad
    FROM DestinosAlcanzables d, Vuelos v
    WHERE d.ciudad = v.ciudadDesde
)
SELECT ciudad FROM DestinosAlcanzables;
```

Funciones de Ventana (Window functions)

Estas permiten aplicar un procesamiento final a los resultados de una consulta, siguiendo estos pasos:

1. Dividiendo en grupos (**particiones**)
2. Ordenando internamente cada partición
3. Cruzando información entre las filas de cada partición

A cada atributo **SELECT** de una consulta se le puede aplicar una función de ventana distinta, o ninguna.

UNICA PARTICION

En estos casos se considera a todo el resultado como una única partición (osea no se usa la palabra clave PARTITION)

El esquema básico para esta situación es:

SELECT .., [$f(A_i)$ | $w([A_i], ..)$] **OVER** ({ **ORDER BY** A_j [ASC | DESC] }), ..
FROM ...

→ Esto ordena el resultado de la consulta por el atributo A_j y para cada fila imprime el resultado de una función de agregación **f(A_i)**, o de una función de ventana **w([A_i],..)**

Ejemplos de funciones de ventana w([Ai],..):

- **RANK():** Nos devuelve el ranking de cada fila, de acuerdo al valor del atributo de ordenamiento de la partición
- **ROW_NUMBER():** nos devuelve el numero de orden de la fila en la partición
- **LAG(A_i, offset):** nos devuelve el valor que toma el atributo A_i en una fila a distancia *offset* hacia atrás de la actual

Ejemplo

Supongamos que en la siguiente tabla de la final de 100m llanos queremos ordenar a los atletas de acuerdo con sus tiempos e indicar su posición final en una columna:

FINAL_2009

nombre_atleta	país_origen	marca_seg
Daniel Bailey	Antigua y Barbuda	9,93
Tyson Gay	Estados Unidos	9,71
Marc Burns	Trinidad y Tobago	10,00
Usain Bolt	Jamaica	9,58
Darvis Patton	Estados Unidos	10,34
Asafa Powell	Jamaica	9,84
Richard Thompson	Trinidad y Tobago	9,93
Dwain Chambers	Reino Unido	10,00

```
SELECT RANK() OVER (ORDER BY marca_seg) AS posición,
       nombre_atleta, país_origen, marca_seg
FROM Final_2009;
```

El resultado será:

posición	nombre_atleta	país_origen	marca_seg
1	Usain Bolt	Jamaica	9,58
2	Tyson Gay	Estados Unidos	9,71
3	Asafa Powell	Jamaica	9,84
4	Daniel Bailey	Antigua y Barbuda	9,93
4	Richard Thompson	Trinidad y Tobago	9,93
6	Dwain Chambers	Reino Unido	10,00
6	Marc Burns	Trinidad y Tobago	10,00
8	Darvis Patton	Estados Unidos	10,34

En caso de empate, si queremos que quede 5 y 6, podemos usar DENSE_RANK()

Otra solución:

- También podríamos devolver la diferencia de tiempo que cada atleta tuvo con el que llegó antes que él:

```
SELECT RANK() OVER (ORDER BY marca_seg) AS posición,
       nombre_atleta, país_origen, marca_seg,
       marca_seg – LAG(marca_seg, 1) OVER (ORDER BY marca_seg) AS diferencia
FROM Final_2009;
```

FINAL_2009

posición	nombre_atleta	país_origen	marca_seg	diferencia
1	Usain Bolt	Jamaica	9,58	
2	Tyson Gay	Estados Unidos	9,71	0,13
3	Asafa Powell	Jamaica	9,84	0,13
4	Daniel Bailey	Antigua y Barbuda	9,93	0,09
4	Richard Thompson	Trinidad y Tobago	9,93	0,00
6	Dwain Chambers	Reino Unido	10,00	0,07
6	Marc Burns	Trinidad y Tobago	10,00	0,00
8	Darvis Patton	Estados Unidos	10,34	0,34

Observaciones

- Para cada columna a la que queremos aplicar una función de ventana, debemos repetir la estructura de **OVER (ORDER BY …)** a la que llamamos: **ventana**
- A diferencia del GROUP BY, el OVER (ORDER BY …) **no agrupa**, con lo cual no cambiara la cantidad de filas en el resultado.
- La función de ventana **se aplica antes** del ordenamiento que pueda hacerse en la clausula ORDER BY al final de la consulta.

- Ejemplo: la siguiente consulta reordena a los atletas por su diferencia en forma decreciente, pero sin alterar los rankings:

```
SELECT RANK() OVER (ORDER BY marca_seg) AS posición,
    nombre_atleta, país_origen, marca_seg,
    marca_seg – LAG(marca_seg, 1) OVER (ORDER BY marca_seg) AS diferencia
FROM Final_2009
ORDER BY diferencia DESC;
```

Otras funciones de agregación y Frames

- Si bien el ORDER BY dentro del OVER() es opcional, al no indicarlo los datos podrán llegar en cualquier orden, lo que no suele ser el comportamiento deseado
- Además, cuando usamos funciones de agregación en la columna (SUM(), COUNT(), ...), el ORDER BY de dentro del OVER() tiene un comportamiento más disruptivo:
  - ✓ Si se emplea OVER() a secas, entonces la función de agregación se calcula sobre toda la partición, con lo que no difiere de lo que haría un GROUP BY, salvo porque nos repite el valor agregado en todas las filas.
  - ✓ Si se emplea OVER(ORDER BY...), entonces la función de agregación en cada fila se calcula sobre un **frame (marco)** que se extiende desde la primera fila de la partición ordenada hasta la actual. Es decir que los resultados devueltos son resultados acumulados (parciales) dentro de la partición.

Definición de ventanas

Si vamos a utilizar una misma ventana muchas veces en la consulta podemos definirla una 'unica vez con la cláusula **WINDOW**

Ejemplo:

```
SELECT RANK() OVER ventana_marca_seg AS posición,
    nombre_atleta, país_origen, marca_seg,
    marca_seg − LAG(marca_seg, 1) OVER ventana_marca_seg AS diferencia
FROM Final_2009
WINDOW ventana_marca_seg AS (ORDER BY marca_seg)
ORDER BY diferencia DESC;
```

MULTIPLES PARTICIONES

En estos esquemas, antes de aplicar cada ventana se puede agrupar el resultado por el valor de un conjunto de atributos

A cada uno de estos grupos (conjunto de filas del resultado) se lo denomina **partición**

El esquema general en este caso es:

SELECT .., [$f(A_i)$ | $w([A_i], ..)$] **OVER** (**PARTITION BY** A_k
{ **ORDER BY** A_j [ASC | DESC] }), ..
FROM ...

Ejemplo 1:

- Supongamos que queremos rankear a los países de acuerdo a quiénes son los que más exportan de cada producto, indicando para cada producto cuál es el país que ocupa cada lugar del ranking, y ordenando el resultado final por producto, ranking y país.

EXPORTACIONES

país	producto	cantidad
Brasil	Trigo	720
Argentina	Trigo	440
USA	Maíz	520
Australia	Trigo	900
China	Sorgo	80
Argentina	Maíz	520
China	Trigo	780

```
SELECT producto,
       RANK() OVER (PARTITION BY producto ORDER BY cantidad DESC) AS posición,
       país, cantidad
FROM Exportaciones
ORDER BY producto, posición;
```

EXPORTACIONES

producto	posición	país	cantidad
Maíz	1	Argentina	520
Maíz	1	USA	520
Sorgo	1	China	80
Trigo	1	Australia	900
Trigo	2	China	780
Trigo	3	Brasil	720
Trigo	4	Argentina	440

Ejemplo 2

¿Qué pasa si queremos utilizar una función de agregación en vez de una función de ventana?

- Disponemos de todas las operaciones realizadas por los clientes de un banco, ordenadas por fecha, incluyendo el saldo inicial de cada uno:

OPERACIONES

fecha	cliente	concepto	monto
2020-04-01	003	Saldo inicial	$10.000
2020-04-01	004	Saldo inicial	$5.000
2020-04-02	003	Depósito en efectivo	$2.000
2020-04-15	004	Farmacia	-$2.100
2020-04-18	003	Adidas	-$3.500
2020-04-23	003	Transferencia a Jorge Mussi	-$2.000
2020-04-28	004	Transferencia de Emilce Vega	$2.800

- Quisieramos generar un resumen ordenado por cliente y fecha en donde además de las operaciones realizadas se indique el saldo del cliente en todo momento.

```
SELECT cliente, fecha, concepto, monto,
       SUM(monto) OVER (PARTITION BY cliente ORDER BY fecha) AS saldo
FROM Operaciones
ORDER BY cliente, fecha;
```

cliente	fecha	concepto	monto	saldo
003	2020-04-01	Saldo inicial	$10.000	$10.000
003	2020-04-02	Depósito en efectivo	$2.000	$12.000
003	2020-04-18	Adidas	-$3.500	$8.500
003	2020-04-23	Transferencia a Jorge Mussi	-$2.000	$6.500
004	2020-04-01	Saldo inicial	$5.000	$5.000
004	2020-04-15	Farmacia	-$2.100	$2.900
004	2020-04-28	Transferencia de Emilce Vega	$2.800	$5.700

La función SUM() utiliza en cada fila un marco (frame) que va desde el inicio del mes de ese cliente hasta la fecha actual, calculando así las “sumas parciales”

Ejemplo 3:

Supongamos que en una tabla tenemos el listado de toques de balón durante un partido de fútbol. Cada entrada de esta tabla indica el instante en que unfutbolista toca el balón.

Queremos encontrar al jugador que pasó más tiempo sin tocar el balón, indicando su nombre y cuál fue dicho tiempo.

TOQUES

timestamp	jugador
16:21:18.418	Lionel Messi
16:21:22.637	Giovanni Lo Celso
16:21:42.402	Dani Alves
16:21:46.535	Nicolás Otamendi
16:21:49.388	Lionel Messi
...	...

```sql
WITH Tiempos_Sin_Balon AS
    (SELECT jugador,
        timestamp - LAG(timestamp, 1)
        OVER (PARTITION BY jugador ORDER BY timestamp ASC) AS tiempo_sin_balón
    FROM Toques)
SELECT DISTINCT t1.jugador, t1.tiempo_sin_balón
FROM Tiempos_Sin_Balon t1
WHERE t1.tiempo_sin_balón = (SELECT MAX(t2.tiempo_sin_balón)
                             FROM Tiempos_Sin_Balon t2);
```

Usos de valores Nulos en BDDs

Tabla de alumnos

padron	nombre	apellido	nombre_inst_sec	cantidad_creditos
12300	Álvarez	Antonio	ORT	30
45600	Barano	Beatriz	ILSE	NULL
55555	De Marti	Darío	NULL	30
78900	Canga	Carla	ORT	40

Tabla de instituciones secundarias (inst_sec)

nombre	direccion
CNBA	Bolívar 263
ILSE	Libertad 555
ORT	Yatay 240

En la tabla de alumnos vemos dos valores NULLs: un alumno no tiene nombre de institución secundaria, por ej porque es un alumno extranjero.

En la tabla de instituciones secundarias tenemos dadas de alta 3 instituciones.

1er problema: el conjunto complemento

2. *(Problemas con el conjunto complemento)* Genere dos consultas que devuelvan el padrón de los alumnos recibidos de la institución secundaria ILSE (primer consulta) y recibidos de otra institución secundaria (segunda consulta), sin saber a priori que la columna nombre_inst_sec permite nulos.

Consulta 1:

```
1  SELECT * FROM alumnos
2  WHERE nombre_inst_sec = 'ILSE'
```

Data Output | Explain | Messages | Notifications

	padron [PK] integer	nombre character varying (30)	apellido character varying (30)	nombre_inst_sec character varying (30)	cantidad_creditos integer
1	45600	Barano	Beatriz	ILSE	[null]

Consulta 2: cambiamos el = por un distinto:

```
1  SELECT * FROM alumnos
2  WHERE nombre_inst_sec <> 'ILSE'
```

Data Output | Explain | Messages | Notifications

	padron [PK] integer	nombre character varying (30)	apellido character varying (30)	nombre_inst_sec character varying (30)	cantidad_creditos integer
1	12300	Álvarez	Antonio	ORT	30
2	78900	Canga	Carla	ORT	40

Ahora, el que tenia colegio NULL no aparece en ningún lado, nunca lo voy a poder devolver. Son consultas complementarias y no están devolviendo el conjunto completo. Los nulos, se toman como desconocidos y siempre se van a evaluar como falsos.

El tema es que si yo desconozco la institución, no puedo decir que ese colegio es ILSE. Pero tampoco puedo decir que es distinta de ILSE para la 2da consulta, porque la realidad es que no se. Cualquier comparación con desconocidos da desconocido y por ende falso.

En este caso, el NULL debería devolverse en los != al ILSE: así que modificamos la consulta 2 para que incluya los nulos. Para comparar con nulos se usa **la palabra clave IS NULL**

```
SELECT * FROM alumnos
WHERE nombre_inst_sec <> 'ILSE' OR nombre_inst_sec IS NULL
```

Data Output | Explain | Messages | Notifications

	padron [PK] integer	nombre character varying (30)	apellido character varying (30)	nombre_inst_sec character varying (30)	cantidad_creditos integer
1	12300	Álvarez	Antonio	ORT	30
2	55555	De Marti	Darío	[null]	30
3	78900	Canga	Carla	ORT	40

Y si necesitara que NO fuera nulo seria IS NOT NULL

Función COALESCE

4. *(Función COALESCE)* Utilizando la función COALESE, devuelva un listado con el padron y la institución secundaria de todos los alumnos. Para aquellos alumnos sin institución secundaria, devolver el nombre 'SIN_INST'.

Es una función que recibe varios parámetros y devuelve el 1ro de esos parámetros que sea distinto de NULL para reemplazar los nulls por valores fijos/valores de otras columnas. Me puede servir para hacer un listado de alumnos, pero por ej para que en vez de que me devuelva ese NULL de la consulta anterior, me devuelva algún valor fijo. Y si no encuentra ninguno, devuelve NULL.:

En nuestro ej le pasamos como 1er paramentro nombre_inti_sec y como segundo un valor fijo "SIN_INST"

```
SELECT padron, nombre, apellido
, COALESCE(nombre_inst_sec, 'SIN_INST')
, cantidad_creditos FROM alumnos
```

De esta manera, lo que hace esta función es que si el 1er parámetro NO es null, devuelve ese valor, sino, va al 2do parámetro, que como acá es fijo, nunca es null, así que lo devuelve:

```
SELECT padron, nombre, apellido
, COALESCE(nombre_inst_sec, 'SIN_INST') AS nombre_inst_sec
, cantidad_creditos FROM alumnos
```

Data Output | Explain | Messages | Notifications

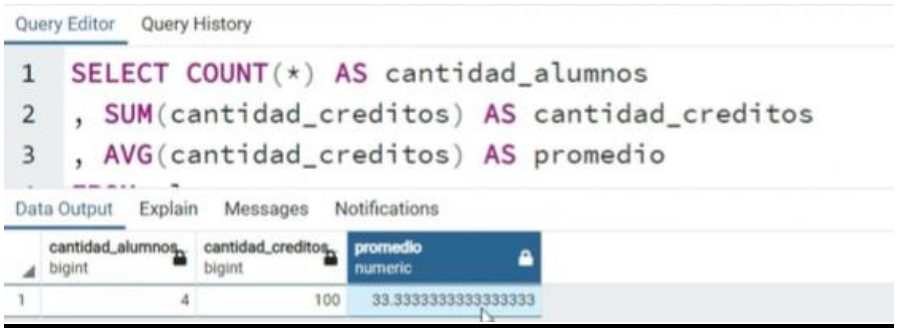
	padron [PK] integer	nombre character varying (30)	apellido character varying (30)	nombre_inst_sec character varying	cantidad_creditos integer
1	12300	Álvarez	Antonio	ORT	30
2	45600	Barano	Beatriz	ILSE	[null]
3	55555	De Marti	Darío	SIN_INST	30
4	78900	Canga	Carla	ORT	40

Le pusimos tmb un alias a la columna xq sino le cambia en nombre.

Como se trabajan los nulos con las funciones de agregación

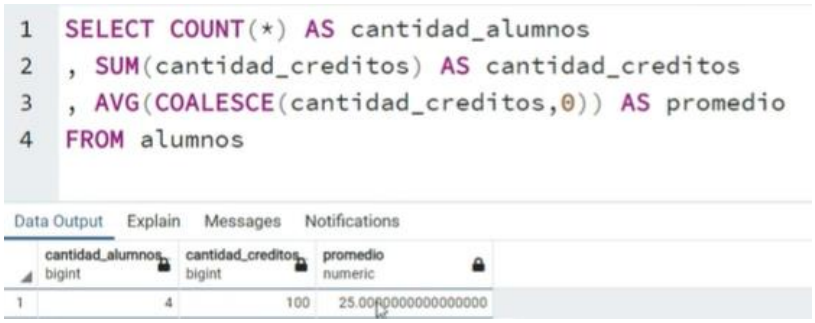
5. *(Problemas matemáticos con valores nulos)* En una consulta, devuelva (en tres columnas distintas) la cantidad total de alumnos, la suma total de créditos obtenidos y el promedio de créditos por alumno.
   a) ¿Cuál es el problema que hay si no se consideran valores nulos?
   b) ¿Cómo puede solucionarse?

- Cuantos alumnus hay: COUNT *
- Cantidad de creditos: SUM(cantidad de créditos)
- Promedio: AVG(cantidad de créditos)
- Le ponemos un alias a cada una



Pero acá el problema es que **los nulos se ignoran en la función de agregación.**

Para salvar eso, le metemos un COALELSE para que la cantidad nula la considere como un valor fijo (0 por ejemplo):



Problemas de ordenamiento

6. *(Problemas de ordenamiento)* Obtenga un listado de los alumnos, ordenados en forma descendente por la institucion de la que egresaron. Utilizar la opción NULLS FIRST/LAST para cambiar el orden.

Si yo lo tiro así nomás, los Nulos se me ponen en cualquier lado. Podría elegir al principio o al final pero porque sí.

Si yo quisiera que los nulos queden en un lugar especifico pongo NULLS FIRST/LAST para decirle mándamelos siempre al principio o siempre al final.

```sql
SELECT * FROM alumnos ORDER BY nombre_inst_sec DESC NULLS LAST
```

Data Output | Explain | Messages | Notifications

	padron [PK] integer	nombre character varying (30)	apellido character varying (30)	nombre_inst_sec character varying (30)	cantidad_creditos integer
1	12300	Álvarez	Antonio	ORT	30
2	78900	Canga	Carla	ORT	40
3	45600	Barano	Beatriz	ILSE	[null]
4	55555	De Marti	Darío	[null]	30

Comportamientos distintos de EXIST y NOT IN

7. *(Comportamientos distintos de **EXISTS** y **NOT IN**)* Utilizando NOT EXISTS, devuelva con una consulta las instituciones secundarias de las cuales no egresó ningún alumno. Resuelva lo mismo pero utilizando NOT IN en vez de NOT EXISTS.

De la tabla que teníamos, deberíamos devolver CNBA.

Para usar not exists, si quiero los que no se egresaron, no tiene que existir el nombre de inst. sec en la tabla de alumnos:

```sql
SELECT * FROM inst_sec i
WHERE NOT EXISTS (
    SELECT * FROM alumnos a WHERE a.nombre_inst_sec = i.nombre
)
```

a = tabla alumnos, i = tabla instituciones

pero esto suele tener una peor performance. Es mejor tener subconsultas independientes.

Podría también cambiar el not exist por NOT IN:

```sql
SELECT * FROM inst_sec i
WHERE i.nombre NOT IN (
    SELECT nombre_inst_sec FROM alumnos
)
```

Haceme una subconsulta independiente que devuelva todos los nombres de instituciones de la tabla, y devolver solo las instituciones cuyo nombre NO esta en la subconsulta.

Pero esto no va a funcionar… por el NULL. Va a mirar la isnt que aparecen en alumnos y se va a preguntar. ¿El CNBA, está en este conjunto? Y no puede responder ni si ni no por el NULL; responde no sé, porque tengo un valor que yo no conozco.

Para salvar este caso, colocamos in WHERE is NOT NULL.

```sql
SELECT * FROM inst_sec i
WHERE i.nombre NOT IN (
    SELECT nombre_inst_sec FROM alumnos
    WHERE nombre_inst_sec IS NOT NULL
)
```

Filtros en Outer Joins

8. *(Filtros en OUTER JOINS)* Haga una consulta que devuelva para cada institución secundaria la cantidad de alumnos que egresaron de ella. Revise que CNBA tenga el valor correcto. Modifique la consulta para que cuente únicamente alumnos con padron mayor a 40000.

Dice que al resultado del inner join le agregue las filas de la tabla que no se hayan juntado con las de la otra tabla.

```
1  SELECT *
2  FROM inst_sec i LEFT OUTER JOIN alumnos a ON (i.nombre = a.nombre_inst_sec)
```

Data Output | Explain | Messages | Notifications

	nombre character varying (30)	direccion character varying (30)	padron integer	nombre character varying (30)	apellido character varying (30)	nombre_inst_sec character varying (30)	cantidad_creditos integer
1	ORT	Yatay 240	12300	Álvarez	Antonio	ORT	30
2	ILSE	Libertad 555	45600	Barano	Beatriz	ILSE	[null]
3	ORT	Yatay 240	78900	Canga	Carla	ORT	40
4	CNBA	Bolivar 263	[null]	[null]	[null]	[null]	[null]

En las columnas que corresponde a alumnos me devuelve valores nulos xq no pudo matchear nada.

```
1  SELECT i.nombre, COUNT(*) AS cantidad
2  FROM inst_sec i LEFT OUTER JOIN alumnos a ON (i.nombre = a.nombre_inst_sec)
3  GROUP BY i.nombre
```

Data Output | Explain | Messages | Notifications

	nombre [PK] character varying (30)	count bigint
1	CNBA	1
2	ORT	2
3	ILSE	1

Ahora, agrego el count. El problema acá es que igualmente me está contabilizando el CNBA como si se hubiera egresado alguien, porque el count así a secas me cuenta filas. Entonces, debería meter adentro del count algún campo de la tabla alumnos, porque el CNBA va a tener todos esos campos en NULL, y ahí si no los va a contar.

```
1  SELECT i.nombre, COUNT(a.padron) AS cantidad
2  FROM inst_sec i LEFT OUTER JOIN alumnos a ON (i.nombre = a.nombre_inst_sec)
3  GROUP BY i.nombre
```

Data Output | Explain | Messages | Notifications

	nombre [PK] character varying (30)	cantidad bigint
1	CNBA	0
2	ORT	2
3	ILSE	1