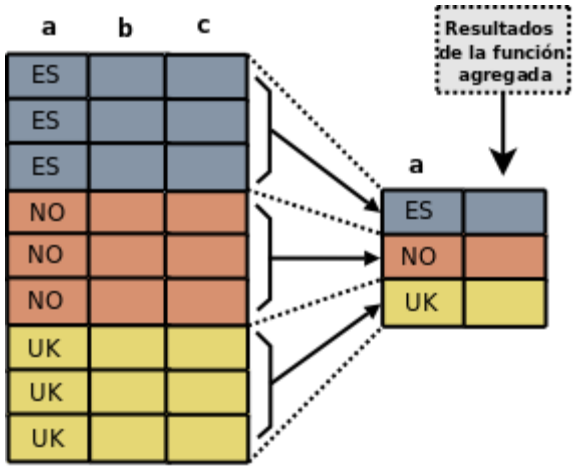
Funciones Ventana (Window Functions)

En este artículo vamos a dar una introducción a las *"funciones ventanas"* (Window functions), una nueva funcionalidad disponible a partir de PostgreSQL 8.4.

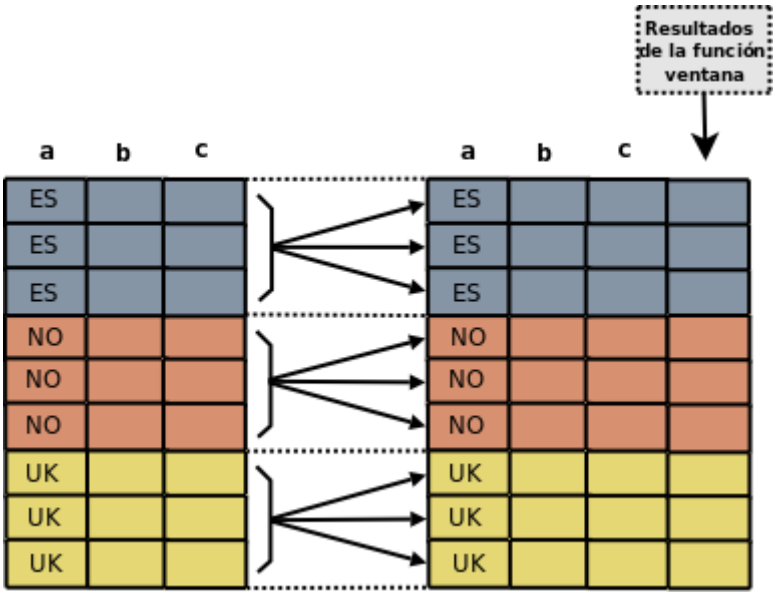
Esta funcionalidad fue introducida en el estandard SQL2003 y ampliada en SQL2008. Esta disponible en Oracle, SQL server, Sybase y DB2, pero en ninguna base de datos de código abierto exceptuando a PostgreSQL.

Las *"funciones ventanas"* hacen la vida más fácil cuando se necesitan realizar cierto tipo de consultas en donde queremos aplicar una función agregada a una partición o subconjunto de filas, desde cada fila que forma parte de un resultado. Las *"funciones ventana"*son similares hasta cierto punto a las típicas funciones agregadas pero con muchas mas posibilidades.

Con funciones agregadas normales y el uso de GROUP BY obtenemos un resultado con una fila por cada valor diferente del atributo usado en el GROUP BY. El siguiente gráfico muestra como funciona una consulta con funciones agregadas normales.



Sin embargo con el uso de las *"funciones ventanas"* la cosa cambia. En el siguiente gráfico podemos ver la diferencia cuando usamos estas funciones.



- Una función ventana es una función agregada aplicada a una partición ó subconjunto del resultado de una consulta
- Una función ventana se define utilizando la cláusula OVER despues de la función.
- Una función ventana devuelve un valor por cada fila del resultado de una consulta
- Para trabajar con ventanas se pueden utilizar las funciones agregadas normales y específicas disponibles en PostgreSQL. Más información sobre las funciones disponibles se encuentra disponible en ingles en 9.19. Window Functions y en 9.18. Aggregate Functions. También se pueden utilizar funciones agregadas creadas por nosotros.
- Una ventana está formada por una partición (PARTITION) y un *marco* (FRAME)
- Una ventana se define aplicando la cláusula OVER a la función agregada
- La cláusula OVER define la partición ó subconjunto que forma la ventana y un *marco*(frame) dentro de la partición
- Una partición se define con la cláusula PARTITION BY
- Si no utilizamos la cláusula PARTITION BY, todas las filas se consideran dentro de la misma ventana
- Un *marco* se define con la cláusula ORDER BY y una cláusula "marco"
- Un *marco* permite definir los límites de una ventana
- La cláusula OVER puede contener la definición de una ventana ó el nombre de una ventana definida con la cláusula opcional WINDOW

La cláusula OVER se define como:

```
OVER(
 [PARTITION BY expresion [, ...]]
 [ORDER BY expression [ASC|DESC|USING operator][NULLS{FIRST|LAST}][, ...]]
 [clausula_frame]
)
```

La [clausula_frame] puede contener:

```
RANGE UNBOUNDED PRECEDING
```

```
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING

ROWS UNBOUNDED PRECEDING

ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
```

La cláusula OVER también se puede definir como:

```
OVER nombre_de_la_ventana
```

En este caso habrá que utilizar una cláusula WINDOW para definir la ventana *nombre_de_la_ventana*

```
WINDOW nombre_de_la_ventana AS (definicion_de_la_ventana) [, ...]
```

Y la *definicion_de_la_ventana* se definiria como:

```
[nombre_de_una_ventana_existente]

[PARTITION BY expresion [, ...]]

[ORDER BY expression [ASC|DESC|USING operator][NULLS{FIRST|LAST}][, ...]]

[clausula_frame]
```

Bueno, hasta ahora solo hemos enumerado conceptos y definiciones que probablemente sean un poco difíciles de asimilar la primera vez que trateis este tema. No hay nada mejor que unos ejemplos prácticos para ver como se utilizan las *"funciones ventana"*.

Para nuestros ejemplos crearemos la tabla *empleado* con información sobre el departamento, el salario y la edad de cada empleado:

```
CREATE TABLE empleado (
  empid integer,
  departamento text,
  salario integer,
  edad integer,
  primary key (empid)
);


INSERT INTO empleado (empid,departamento,salario,edad)

VALUES (1,'ventas',3000,24);

INSERT INTO empleado (empid,departamento,salario,edad)
```

```
VALUES (2,'ventas',3200,26);
INSERT INTO empleado (empid,departamento,salario,edad)
VALUES (3,'ventas',3500,35);
INSERT INTO empleado (empid,departamento,salario,edad)
VALUES (4,'distribucion',2000,22);
INSERT INTO empleado (empid,departamento,salario,edad)
VALUES (5,'distribucion',2100,42);
INSERT INTO empleado (empid,departamento,salario,edad)
VALUES (6,'distribucion',2400,40);
INSERT INTO empleado (empid,departamento,salario,edad)
VALUES (7,'produccion',2800,41);
INSERT INTO empleado (empid,departamento,salario,edad)
VALUES (8,'produccion',2400,29);
INSERT INTO empleado (empid,departamento,salario,edad)
VALUES (9,'produccion',1900,19);
INSERT INTO empleado (empid,departamento,salario,edad)
VALUES (10,'produccion',3000,45);
INSERT INTO empleado (empid,departamento,salario,edad)
VALUES (11,'produccion',3000,40);

SELECT
  empid,
  departamento,
  salario,
  edad
FROM empleado;

 empid | departamento | salario | edad
-------+--------------+---------+------
     1 | ventas       |    3000 |   24
     2 | ventas       |    3200 |   26
     3 | ventas       |    3500 |   35
     4 | distribucion |    2000 |   22
```

```
     5 | distribucion |    2100 |   42
     6 | distribucion |    2400 |   40
     7 | produccion   |    2800 |   41
     8 | produccion   |    2400 |   29
     9 | produccion   |    1900 |   19
    10 | produccion   |    3000 |   45
    11 | produccion   |    3000 |   40
(11 rows)
```

Ahora vamos a definir nuestra primera *"función ventana"*. A la consulta anterior le vamos a añadir una columna con el salario medio del departamento donde el empleado trabaja. Para ellos utilizamos la funcion avg() con la cláusula OVER() y definimos la partición por departamentos.

```
SELECT
  empid,
  departamento,
  salario,
  edad,
  avg(salario) OVER (PARTITION BY departamento) AS salario_medio
FROM empleado ;


 empid | departamento | salario | edad |     salario_medio
-------+--------------+---------+------+-----------------------
     4 | distribucion |    2000 |   22 | 2166.6666666666666667
     6 | distribucion |    2400 |   40 | 2166.6666666666666667
     5 | distribucion |    2100 |   42 | 2166.6666666666666667
    11 | produccion   |    3000 |   40 | 2620.0000000000000000
     8 | produccion   |    2400 |   29 | 2620.0000000000000000
     9 | produccion   |    1900 |   19 | 2620.0000000000000000
    10 | produccion   |    3000 |   45 | 2620.0000000000000000
     7 | produccion   |    2800 |   41 | 2620.0000000000000000
     3 | ventas       |    3500 |   35 | 3233.3333333333333333
     2 | ventas       |    3200 |   26 | 3233.3333333333333333
     1 | ventas       |    3000 |   24 | 3233.3333333333333333
```

```
(11 rows)
```

Particularmente me gusta más utilizar la cláusula WINDOW. La consulta anterior utilizando la clausula WINDOW se puede escribir de la siguiente manera:

```
SELECT
  empid,
  departamento,
  salario,
  edad,
  avg(salario) OVER ventana_departamento AS salario_medio
FROM empleado
WINDOW
  ventana_departamento AS (PARTITION BY departamento);
```

¿Qué os parece?, no es tan difícil cuando se ve en un ejemplo ¿no?. Vamos a seguir complicando las cosas. Además del salario medio queremos la edad media del departamento donde el empleado trabaja.

Los valores medios los voy a redondear con la función round() para no tener tantos decimales. Como las dos *funciones ventana* que calculan los valores medios utilizan la misma ventana, solo habrá que utilizar una sola cláusula WINDOW. Si no utilizaramos la cláusula WINDOW, tendriamos que definir la ventana dos veces con OVER().

```
SELECT
  empid,
  departamento,
  salario,
  edad,
  round(avg(salario) OVER ventana_departamento) AS sal_medio,
  round(avg(edad) OVER ventana_departamento) AS ed_media
FROM empleado
WINDOW
  ventana_departamento AS (PARTITION BY departamento);


 empid | departamento | salario | edad | sal_medio | ed_media
-------+--------------+---------+------+-----------+----------
     4 | distribucion |    2000 |   22 |      2167 |       35
     6 | distribucion |    2400 |   40 |      2167 |       35
```

```
     5 | distribucion |    2100 |   42 |      2167 |       35
    11 | produccion   |    3000 |   40 |      2620 |       35
     8 | produccion   |    2400 |   29 |      2620 |       35
     9 | produccion   |    1900 |   19 |      2620 |       35
    10 | produccion   |    3000 |   45 |      2620 |       35
     7 | produccion   |    2800 |   41 |      2620 |       35
     3 | ventas       |    3500 |   35 |      3233 |       28
     2 | ventas       |    3200 |   26 |      3233 |       28
     1 | ventas       |    3000 |   24 |      3233 |       28
(11 rows)
```

Una vez que sabemos como hallar el salario medio y la edad media en cada de departamento, podriamos calcular la diferencia entre lo que cobra cada empleado y la media del departamento y la diferencia entre la edad de cada empleado y la media del departamento.

```
SELECT
  empid,
  departamento,
  salario,
  edad,
  salario - round(avg(salario) OVER ventana_departamento) AS sal_diff,
  edad - round(avg(edad) OVER ventana_departamento) AS ed_diff
FROM empleado
WINDOW
  ventana_departamento AS (PARTITION BY departamento);


 empid | departamento | salario | edad | sal_diff | ed_diff
-------+--------------+---------+------+----------+---------
     4 | distribucion |    2000 |   22 |     -167 |     -13
     6 | distribucion |    2400 |   40 |      233 |       5
     5 | distribucion |    2100 |   42 |      -67 |       7
    11 | produccion   |    3000 |   40 |      380 |       5
     8 | produccion   |    2400 |   29 |     -220 |      -6
     9 | produccion   |    1900 |   19 |     -720 |     -16
    10 | produccion   |    3000 |   45 |      380 |      10
```

```
     7 | produccion   |    2800 |   41 |      180 |       6
     3 | ventas       |    3500 |   35 |      267 |       7
     2 | ventas       |    3200 |   26 |      -33 |      -2
     1 | ventas       |    3000 |   24 |     -233 |      -4
(11 rows)
```

Ahora vamos a hallar el salario medio y la edad media de toda la empresa, sin hacer distinción entre departamentos. Para ello definimos la cláusula WINDOW vacia. De esta manera la ventana definida es todo el resultado:

```
SELECT
  empid,
  departamento,
  salario,
  edad,
  round(avg(salario) OVER ventana_empresa) AS sal_medio,
  round(avg(edad) OVER ventana_empresa) AS ed_media
FROM empleado
WINDOW
  ventana_empresa AS ();


 empid | departamento | salario | edad | sal_medio | ed_media
-------+--------------+---------+------+-----------+----------
     1 | ventas       |    3000 |   24 |      2664 |       33
     2 | ventas       |    3200 |   26 |      2664 |       33
     3 | ventas       |    3500 |   35 |      2664 |       33
     4 | distribucion |    2000 |   22 |      2664 |       33
     5 | distribucion |    2100 |   42 |      2664 |       33
     6 | distribucion |    2400 |   40 |      2664 |       33
     7 | produccion   |    2800 |   41 |      2664 |       33
     8 | produccion   |    2400 |   29 |      2664 |       33
     9 | produccion   |    1900 |   19 |      2664 |       33
    10 | produccion   |    3000 |   45 |      2664 |       33
    11 | produccion   |    3000 |   40 |      2664 |       33
(11 rows)
```

Equivalentemente a uno de los ejemplos anteriores, la diferencia entre lo que cobra cada empleado y la media de la empresa y la diferencia entre la edad de cada empleado y la media de la empresa se obtendria así:

```
SELECT
  empid,
  departamento,
  salario,
  edad,
  salario - round(avg(salario) OVER ventana_empresa) AS sal_diff,
  edad - round(avg(edad) OVER ventana_empresa) AS ed_diff
FROM empleado
WINDOW
  ventana_empresa AS ();


 empid | departamento | salario | edad | sal_diff | ed_diff
-------+--------------+---------+------+----------+---------
     1 | ventas       |    3000 |   24 |      336 |      -9
     2 | ventas       |    3200 |   26 |      536 |      -7
     3 | ventas       |    3500 |   35 |      836 |       2
     4 | distribucion |    2000 |   22 |     -664 |     -11
     5 | distribucion |    2100 |   42 |     -564 |       9
     6 | distribucion |    2400 |   40 |     -264 |       7
     7 | produccion   |    2800 |   41 |      136 |       8
     8 | produccion   |    2400 |   29 |     -264 |      -4
     9 | produccion   |    1900 |   19 |     -764 |     -14
    10 | produccion   |    3000 |   45 |      336 |      12
    11 | produccion   |    3000 |   40 |      336 |       7
(11 rows)
```

Ahora vamos a poner un ejemplo con dos ventanas. Una la vamos a utilizar para hallar en que posición (ranking) se encuentra cada empleado por departamentos, en relación al salario que cobra, y la otra para hallar en que posición (ranking) se encuentra cada empleado por departamentos, en relación a la edad que tiene. Utilizaremos la función dense_rank().

```
SELECT
```

```
  empid,
  departamento,
  salario,
  edad,
  dense_rank() OVER ventana_departamento_salario AS sal_pos,
  dense_rank() OVER ventana_departamento_edad AS ed_pos
FROM empleado
WINDOW
  ventana_departamento_salario AS
  (PARTITION BY departamento ORDER BY salario DESC),
  ventana_departamento_edad AS
  (PARTITION BY departamento ORDER BY edad DESC);


 empid | departamento | salario | edad | sal_pos | ed_pos
-------+--------------+---------+------+---------+--------
     5 | distribucion |    2100 |   42 |       2 |      1
     6 | distribucion |    2400 |   40 |       1 |      2
     4 | distribucion |    2000 |   22 |       3 |      3
    10 | produccion   |    3000 |   45 |       1 |      1
     7 | produccion   |    2800 |   41 |       2 |      2
    11 | produccion   |    3000 |   40 |       1 |      3
     8 | produccion   |    2400 |   29 |       3 |      4
     9 | produccion   |    1900 |   19 |       4 |      5
     3 | ventas       |    3500 |   35 |       1 |      1
     2 | ventas       |    3200 |   26 |       2 |      2
     1 | ventas       |    3000 |   24 |       3 |      3
(11 rows)
```

¿Y si solamente queremos, por ejemplo, los datos del departamento de producción?. Podemos utilizar una cláusula WHERE para acotar el resultado.

```
SELECT
  empid,
  departamento,
  salario,
```

```
  edad,
  dense_rank() OVER ventana_departamento_salario AS sal_pos,
  dense_rank() OVER ventana_departamento_edad AS ed_pos
FROM empleado
WHERE
  departamento = 'produccion'
WINDOW
  ventana_departamento_salario AS
  (PARTITION BY departamento ORDER BY salario DESC),
  ventana_departamento_edad AS
  (PARTITION BY departamento ORDER BY edad DESC);


 empid | departamento | salario | edad | sal_pos | ed_pos
-------+--------------+---------+------+---------+--------
    10 | produccion   |    3000 |   45 |       1 |      1
     7 | produccion   |    2800 |   41 |       2 |      2
    11 | produccion   |    3000 |   40 |       1 |      3
     8 | produccion   |    2400 |   29 |       3 |      4
     9 | produccion   |    1900 |   19 |       4 |      5
(5 rows)
```

Podriamos haber conseguido el mismo resultado escribiendo la consulta anterior de las siguientes maneras:

Sin usar la clausula PARTITION BY, ya que el resultado solo tiene datos de un departamento.

```
SELECT
  empid,
  departamento,
  salario,
  edad,
  dense_rank() OVER ventana_departamento_salario AS sal_pos,
  dense_rank() OVER ventana_departamento_edad AS ed_pos
FROM empleado
WHERE
```

```
  departamento = 'produccion'
WINDOW
  ventana_departamento_salario AS
  (ORDER BY salario DESC),
  ventana_departamento_edad AS
  (ORDER BY edad DESC);

 empid | departamento | salario | edad | sal_pos | ed_pos
-------+--------------+---------+------+---------+--------
    10 | produccion   |    3000 |   45 |       1 |      1
     7 | produccion   |    2800 |   41 |       2 |      2
    11 | produccion   |    3000 |   40 |       1 |      3
     8 | produccion   |    2400 |   29 |       3 |      4
     9 | produccion   |    1900 |   19 |       4 |      5
(5 rows)
```

O utilizando un subselect:

```
SELECT *
FROM
(
 SELECT
   empid,
   departamento,
   salario,
   edad,
   dense_rank() OVER ventana_departamento_salario AS sal_pos,
   dense_rank() OVER ventana_departamento_edad AS ed_pos
 FROM empleado
 WINDOW
   ventana_departamento_salario AS
   (PARTITION BY departamento ORDER BY salario DESC),
   ventana_departamento_edad AS
   (PARTITION BY departamento ORDER BY edad DESC)
```

```
) AS subselect
WHERE
  departamento = 'produccion';


 empid | departamento | salario | edad | sal_pos | ed_pos
-------+--------------+---------+------+---------+--------
    10 | produccion   |    3000 |   45 |       1 |      1
     7 | produccion   |    2800 |   41 |       2 |      2
    11 | produccion   |    3000 |   40 |       1 |      3
     8 | produccion   |    2400 |   29 |       3 |      4
     9 | produccion   |    1900 |   19 |       4 |      5
(5 rows)
```

Aunque no todas estas consultas son igual de eficientes. Las dos primeras tienen un tiempo de ejecución similar en torno a los 0.094 ms, pero la del subselect tiene un tiempo de ejecución de 0.137 ms, un 40% más lenta que las dos primeras.

La consulta equivalente para toda la empresa seria:

```
SELECT
  empid,
  departamento,
  salario,
  edad,
  dense_rank() OVER ventana_empresa_salario AS sal_pos,
  dense_rank() OVER ventana_empresa_edad AS ed_pos
FROM empleado
WINDOW
  ventana_empresa_salario AS (ORDER BY salario DESC),
  ventana_empresa_edad AS (ORDER BY edad  DESC);


 empid | departamento | salario | edad | sal_pos | ed_pos
-------+--------------+---------+------+---------+--------
    10 | produccion   |    3000 |   45 |       3 |      1
     5 | distribucion |    2100 |   42 |       6 |      2
     7 | produccion   |    2800 |   41 |       4 |      3
```

```
     6 | distribucion |    2400 |   40 |       5 |      4
    11 | produccion   |    3000 |   40 |       3 |      4
     3 | ventas       |    3500 |   35 |       1 |      5
     8 | produccion   |    2400 |   29 |       5 |      6
     2 | ventas       |    3200 |   26 |       2 |      7
     1 | ventas       |    3000 |   24 |       3 |      8
     4 | distribucion |    2000 |   22 |       7 |      9
     9 | produccion   |    1900 |   19 |       8 |     10
(11 rows)
```