

Introducción al lenguaje SQL de PostgreSQL

Este tutorial supone una inmersión en el lenguaje SQL implementado en PostgreSQL. Si bien el lenguaje SQL tiene un estándar definido, por motivos históricos o prácticos cada motor de bases de datos tiene especificidades en sus implementaciones del lenguaje SQL. En este tutorial entraremos al detalle de esas especificaciones para proveer una guía que sea útil durante la gestión de datos en PostgreSQL. Este tutorial trata aquellos aspectos básicos del lenguaje SQL orientados a la manipulación de datos. Otras estructuras avanzadas serán tratadas en otros tutoriales específicos.

- [Consulta de datos en PostgreSQL](#)
 - [Clausula SELECT de PostgreSQL](#)
 - [Alias de columnas en PostgreSQL](#)
 - [Clausula ORDER BY en PostgreSQL](#)

Consulta de datos en PostgreSQL

Una de las tareas más comunes y en la que invertiremos mucho tiempo es en la consulta de datos. Cada vez que necesitemos extraer un conjunto de datos de una tabla, deberemos escribir una consulta que permita obtener los datos con los contenidos, estructura y formato necesarios. En este capítulo vamos a realizar una introducción a la consulta de datos.

Clausula SELECT de PostgreSQL

La clausula `SELECT` es una de las más utilizadas cuando se trabaja con datos. Esta permite extraer datos de una o varias tablas. Para ello la clausula `SELECT` puede enriquecerse con otras clausulas que nos permitirán filtrar los datos hasta conseguir el conjunto exacto que necesitamos. Por todo ello, podemos decir que la clausula `SELECT` es una de las más complejas que existen en PostgreSQL.

Dado que la clausula `SELECT` es una clausula compleja, iremos presentando sus diferentes componentes en diversos capítulos. En este primer capítulo nos centraremos en la sintaxis básica de la clausula `SELECT`.

Sintaxis

La sintaxis básica de la clausula `SELECT` es la siguiente:

```
SELECT
  columnas
FROM
  tabla;
```

Como ves la sentencia `SELECT` tiene una sintaxis muy intuitiva que puede ser traducida fácilmente al lenguaje natural: **selecciona** la lista de **columnas de la tabla**. Ahora veamos con más detalle los componentes:

- `SELECT`. Es el nombre de la clausula e indica al motor de bases de datos que deseamos ejecutar una consulta.
- `columnas`. Es una lista de todas columnas que queremos recuperar de la tabla `tabla`. La lista de columnas puede ser uno o varios nombres de columna separados por una coma (,). También podemos usar el carácter asterisco (*), que significa 'todas' y nos permite seleccionar todas las columnas sin tener que definir los nombres.
- `FROM`. Es una clausula dentro de `SELECT` que define una lista de las tablas desde donde queremos recuperar datos. Esta clausula es opcional y puede ser omitida cuando no se van a recuperar datos de ninguna tabla.
- `tabla`. Es la lista propiamente dicha de tablas sobre las cuales queremos recuperar datos. La lista puede ser de una o más tablas separadas por una coma.

Recuerda que todas las sentencias PostgreSQL deben finalizar por un punto y coma (;).

Para entender mejor la clausula `SELECT` vamos a ver un ejemplo práctico.

```
SELECT * FROM employees;
```

La sentencia anterior devuelve una lista con todos los registros contenidos en la tabla `employees`:

	<code>employee_id</code> [PK] smallint	<code>last_name</code> character varying (20)	<code>first_name</code> character varying (10)
1	1	Davolio	Nancy
2	2	Fuller	Andrew
3	3	Leverling	Janet
4	4	Peacock	Margaret
5	5	Buchanan	Steven
6	6	Suyama	Michael
7	7	King	Robert
8	8	Callahan	Laura
9	9	Dodsworth	Anne

En este caso, como hemos usado el asterisco (*) en la selección de columnas, el resultado devuelto por PostgreSQL contiene todas las columnas de la tabla.

Selección de determinadas columnas

Como vimos en el ejemplo anterior al usar el asterisco (*) se seleccionan todas las columnas de la tabla. Veamos ahora un ejemplo en el que vamos a seleccionar solo unas determinadas columnas.

```
SELECT last_name, first_name, title_of_courtesy FROM employees;
```

Como vemos en la lista de columnas hemos definido qué columnas queremos recuperar, en este ejemplo son las columnas `last_name`, `first_name` y `title_of_courtesy`. El resultado que obtenemos es el siguiente:

	employee_id [PK] smallint	last_name character varying (20)	first_name character varying (10)	title character varying (30)	title_of_courtesy character varying (25)
1	1	Davolio	Nancy	Sales Representative	Ms.
2	2	Fuller	Andrew	Vice President, Sales	Dr.
3	3	Leverling	Janet	Sales Representative	Ms.
4	4	Peacock	Margaret	Sales Representative	Mrs.
5	5	Buchanan	Steven	Sales Manager	Mr.
6	6	Suyama	Michael	Sales Representative	Mr.
7	7	King	Robert	Sales Representative	Mr.
8	8	Callahan	Laura	Inside Sales Coordinator	Ms.
9	9	Dodsworth	Anne	Sales Representative	Ms.

En este caso, como hemos usado el asterisco (*) en la selección de columnas, el resultado devuelto por PostgreSQL contiene todas las columnas de la tabla.

Selección de determinadas columnas

Como vimos en el ejemplo anterior al usar el asterisco (*) se seleccionan todas las columnas de la tabla. Veamos ahora un ejemplo en el que vamos a seleccionar solo unas determinadas columnas.

```
SELECT last_name, first_name, title_of_courtesy FROM employees;
```

Como vemos en la lista de columnas hemos definido qué columnas queremos recuperar, en este ejemplo son las columnas `last_name`, `first_name` y `title_of_courtesy`. El resultado que obtenemos es el siguiente:

	last_name character varying (20)	first_name character varying (10)	title_of_courtesy character varying (25)
1	Davolio	Nancy	Ms.
2	Fuller	Andrew	Dr.
3	Leverling	Janet	Ms.
4	Peacock	Margaret	Mrs.
5	Buchanan	Steven	Mr.
6	Suyama	Michael	Mr.
7	King	Robert	Mr.
8	Callahan	Laura	Ms.
9	Dodsworth	Anne	Ms.

Selección usando expresiones

Toda la potencia de la cláusula `SELECT` reside en la potencia del lenguaje SQL de PostgreSQL. Una de estas características es el uso de expresiones para estructurar y formatear las columnas de

manera que obtengamos un conjunto de datos en la forma que nosotros deseemos. De este modo tenemos una flexibilidad total para obtener los datos y no tener que ceñirnos a la estructura que tienen en la base de datos.

Como ejemplo práctico vamos a suponer que queremos recuperar la lista de todos los empleados, pero queremos recuperar solo una columna que contenga el título de cortesía, el nombre y el apellido de cada uno:

```
SELECT title_of_courtesy || ' ' || first_name || ' ' || last_name FROM employees;
```

Observa que en la lista de columnas hemos definido una sola columna que está formada por la concatenación de los resultados de las tres columnas. La concatenación se define con doble tubo `||`. En este caso estamos concatenando el valor de dos columnas con un espacio en blanco, lo cual va a construir una cadena formateada, como puedes apreciar a continuación.

	?column? text
1	Ms. Nancy Davolio
2	Dr. Andrew Fuller
3	Ms. Janet Leverling
4	Mrs. Margaret Peacock
5	Mr. Steven Buchanan
6	Mr. Michael Suyama
7	Mr. Robert King
8	Ms. Laura Callahan
9	Ms. Anne Dodsworth

Como habíamos avanzado anteriormente, la cláusula `FROM` es opcional y puede ser omitida cuando no se consultan datos sobre una tabla. Un ejemplo es el uso en la siguiente sentencia:

```
SELECT ( 20 + 10) * 2;
```

Cuyo resultado sera:

	?column? integer
1	60

Alias de columnas en PostgreSQL

Los alias de columnas te permiten definir un nombre temporal para las columnas y las expresiones. Los alias definidos en una consulta duran el tiempo que dure la ejecución de la consulta y no modifican ninguna columna ni dato.

PostgreSQL dispone de una cláusula especial para definir los alias, pero su uso es opcional. Vemos dos ejemplos de sintaxis.

```
SELECT column AS alias FROM tabla;
```

En este primer ejemplo hemos usado la cláusula `AS` para definir el alias.

```
SELECT column alias FROM tabla;
```

En este segundo ejemplo hemos omitido el uso de la cláusula `AS`. En todo caso ambas posibilidades son correctas y funcionan.

Los alias se usan generalmente para personalizar los nombres de las columnas en los resultados. Su uso principal es para definir un nombre de columna más intuitivo o también para definir un nombre al resultado de una expresión. Solamente hay que tomar la precaución de no definir un alias con el mismo nombre de una columna que exista dentro de las tablas consultadas ya que se producirá un error.

Veamos algunos ejemplos a continuación.

Asignación de un alias a una columna cualquiera

En el capítulo anterior mostramos un ejemplo para recuperar unas ciertas columnas de una tabla. Supongamos que ahora queremos ejecutar la misma consulta pero, ahora, queremos que los nombres de las columnas en el resultado estén en castellano y no en inglés. Lo lograríamos con una sentencia como esta:

```
SELECT
    last_name Nombre,
    first_name apellido,
    title_of_courtesy "Título de cortesía"
FROM employees;
```

Esta consulta produce el siguiente resultado:

	nombre character varying (20) 🔒	apellido character varying (10) 🔒	Título de cortesía character varying (25) 🔒
1	Davolio	Nancy	Ms.
2	Fuller	Andrew	Dr.
3	Leverling	Janet	Ms.
4	Peacock	Margaret	Mrs.
5	Buchanan	Steven	Mr.
6	Suyama	Michael	Mr.
7	King	Robert	Mr.
8	Callahan	Laura	Ms.
9	Dodsworth	Anne	Ms.

Si observas los encabezados de las columnas del resultado, muestran el alias, pero hay un detalle. En la primera columna, a la que le hemos asignado el alias *Nombre*, aparece en el resultado como *nombre*. Esto se debe a que PostgreSQL no tiene en cuenta si lo que esta escrito lo esta en mayúsculas o en minúsculas.


Ahora observa la tercera columna, definimos el alias usando comillas "*Título de cortesía*" y en el resultado se muestra como *Título de cortesía*. Como puedes observar al usar comillas ("), hemos definido un literal de cadena que, PostgreSQL respeta y mantiene el texto tal y como lo definimos.

Asignación de un alias a una expresión

Como dijimos anteriormente uno de los usos de los alias es el de darle nombre al resultado de una expresión. Si tomamos el ejemplo del capítulo anterior y lo modificamos, quedaría así:

```
SELECT title_of_courtesy || ' ' || first_name || ' ' || last_name AS "Nombre Completo" FROM
employees;
```

Nos devuelve el siguiente resultado:

	Nombre Completo text 
1	Ms. Nancy Davolio
2	Dr. Andrew Fuller
3	Ms. Janet Leverling
4	Mrs. Margaret Peacock
5	Mr. Steven Buchanan
6	Mr. Michael Suyama
7	Mr. Robert King
8	Ms. Laura Callahan
9	Ms. Anne Dodsworth

Como puedes ver, ahora el encabezado de la columna muestra un nombre más intuitivo y no el nombre autogenerado por PostgreSQL, que no tiene ningún tipo de contexto con los datos.

Clausula ORDER BY en PostgreSQL

Los resultados devueltos por una consulta con `SELECT` generalmente no guardan un orden específico. La clausula `ORDER BY` ofrece la posibilidad de ordenar los resultados de una consulta siguiendo unos criterios definidos.

La sintaxis de la clausula `ORDER BY` es como sigue:

```
SELECT
    lista_de_columnas
FROM
    tabla
ORDER BY
    expresión_1 [ASC | DESC] [NULLS FIRST | NULLS LAST],
    ...
    expresión_n [ASC | DESC] [NULLS FIRST | NULLS LAST]
```

Como puedes ver, para cada expresión de ordenación, se puede definir el orden, `ASC` para ascendente y `DESC` para descendente. Si se omite el orden, entonces se usará `ASC` por defecto. Cada una de las expresiones de ordenación debe separarse de una coma (,), pues se trata de una lista.

Las opciones `[NULLS FIRST | NULLS LAST]` hacen referencia al modo en que serán tratados los valores nulos.

A continuación se muestran algunos ejemplos de uso de la clausula `ORDER BY`.

Ordenar una consulta por una columna

Uno de los ejemplos mínimos del uso de la clausula `ORDER BY` es la ordenación de los resultados por una sola columna. Veamos el siguiente ejemplo:

```
SELECT
    ship_name,
```

```

    ship_address,
    ship_city
FROM
    orders
ORDER BY
    ship_name ASC;

```

El resultado devuelto por esta consulta esta ordenado por `ship_name` :

	ship_name character varying (40) 	ship_address character varying (60) 	ship_city character varying (15) 
1	Alfred's Futterkiste	Obere Str. 57	Berlin
2	Alfred's Futterkiste	Obere Str. 57	Berlin
3	Alfred's Futterkiste	Obere Str. 57	Berlin
4	Alfred's Futterkiste	Obere Str. 57	Berlin
5	Alfred's Futterkiste	Obere Str. 57	Berlin
6	Alfreds Futterkiste	Obere Str. 57	Berlin
7	Ana Trujillo Emparedados y helados	Avda. de la Constitución 2222	México D.F.
8	Ana Trujillo Emparedados y helados	Avda. de la Constitución 2222	México D.F.
9	Ana Trujillo Emparedados y helados	Avda. de la Constitución 2222	México D.F.
10	Ana Trujillo Emparedados y helados	Avda. de la Constitución 2222	México D.F.
11	Antonio Moreno Taquería	Mataderos 2312	México D.F.
12	Antonio Moreno Taquería	Mataderos 2312	México D.F.
13	Antonio Moreno Taquería	Mataderos 2312	México D.F.
14	Antonio Moreno Taquería	Mataderos 2312	México D.F.

La consulta a continuación es análoga a la anterior y produce el mismo resultado:

```

SELECT
    ship_name,
    ship_address,
    ship_city
FROM
    orders
ORDER BY
    ship_name;

```

Recordemos que si omitimos los complementos `ASC` y `DESC` en la clausula `ORDER BY` por defecto se usará `ASC` por defecto.

Del mismo modo, si definimos el ordenamiento en `DESC`, los resultados se ordenarán en sentido descendente:

```

SELECT
    ship_name,
    ship_address,
    ship_city
FROM
    orders
ORDER BY
    ship_name DESC;

```

La consulta anterior produce el siguiente resultado:

	ship_name character varying (40) 🔒	ship_address character varying (60) 🔒	ship_city character varying (15) 🔒
1	Wolski Zajazd	ul. Filtrowa 68	Warszawa
2	Wolski Zajazd	ul. Filtrowa 68	Warszawa
3	Wolski Zajazd	ul. Filtrowa 68	Warszawa
4	Wolski Zajazd	ul. Filtrowa 68	Warszawa
5	Wolski Zajazd	ul. Filtrowa 68	Warszawa
6	Wolski Zajazd	ul. Filtrowa 68	Warszawa
7	Wolski Zajazd	ul. Filtrowa 68	Warszawa
8	Wilman Kala	Keskuskatu 45	Helsinki
9	Wilman Kala	Keskuskatu 45	Helsinki
10	Wilman Kala	Keskuskatu 45	Helsinki
11	Wilman Kala	Keskuskatu 45	Helsinki
12	Wilman Kala	Keskuskatu 45	Helsinki
13	Wilman Kala	Keskuskatu 45	Helsinki
14	Wilman Kala	Keskuskatu 45	Helsinki

Ordenar una consulta por múltiples columnas

La cláusula `ORDER BY` puede ordenar los resultados por múltiples columnas. Para ello la cláusula `ORDER BY` acepta una lista de columnas para ordenar. Veamos un ejemplo:

```

SELECT
    ship_name,
    ship_address,
    ship_city,
    shipped_date

```

```
FROM
    orders
ORDER BY
    ship_name DESC,
    shipped_date ASC;
```

Cuyo resultado es como sigue:

	ship_name character varying (40) 🔒	ship_address character varying (60) 🔒	ship_city character varying (15) 🔒	shipped_date date 🔒
1	Wolski Zajazd	ul. Filtrowa 68	Warszawa	1996-12-09
2	Wolski Zajazd	ul. Filtrowa 68	Warszawa	1997-08-01
3	Wolski Zajazd	ul. Filtrowa 68	Warszawa	1997-12-31
4	Wolski Zajazd	ul. Filtrowa 68	Warszawa	1998-02-13
5	Wolski Zajazd	ul. Filtrowa 68	Warszawa	1998-03-03
6	Wolski Zajazd	ul. Filtrowa 68	Warszawa	1998-04-17
7	Wolski Zajazd	ul. Filtrowa 68	Warszawa	1998-05-01
8	Wilman Kala	Keskuskatu 45	Helsinki	1997-08-06
9	Wilman Kala	Keskuskatu 45	Helsinki	1997-09-19
10	Wilman Kala	Keskuskatu 45	Helsinki	1997-10-14
11	Wilman Kala	Keskuskatu 45	Helsinki	1998-02-09
12	Wilman Kala	Keskuskatu 45	Helsinki	1998-02-12
13	Wilman Kala	Keskuskatu 45	Helsinki	1998-03-04
14	Wilman Kala	Keskuskatu 45	Helsinki	1998-04-10

En este ejemplo, hemos ordenado, en primer lugar todos los resultados por la columna `ship_name` en orden descendente. A continuación, todos los resultados que tienen el mismo valor, son ordenados a su vez por la columna `shipped_date` en orden ascendente.

Esto significa que la cláusula `ORDER BY`, realiza la ordenación por grupos de igual valor en la columnas, según el orden en el que aparecen en la lista de izquierda a derecha.

Ordenar una consulta usando expresiones

La cláusula `ORDER BY` puede también ordenar el resultado de una expresión. Un ejemplo puede ser el uso de la función `LENGTH()` y ordenar por la longitud de la cadena:

```
SELECT
    ship_name,
    ship_address,
    ship_city,
    LENGTH(ship_name) longitud
FROM
```

```
orders
ORDER BY
    longitud ASC;
```

El resultado obtenido es el siguiente:

	ship_name character varying (40) 🔒	ship_address character varying (60) 🔒	ship_city character varying (15) 🔒	longitud integer 🔒
1	Bon app'	12, rue des Bouchers	Marseille	8
2	Bon app'	12, rue des Bouchers	Marseille	8
3	Bon app'	12, rue des Bouchers	Marseille	8
4	Bon app'	12, rue des Bouchers	Marseille	8
5	Bon app'	12, rue des Bouchers	Marseille	8
6	Bon app'	12, rue des Bouchers	Marseille	8
7	Bon app'	12, rue des Bouchers	Marseille	8
8	Bon app'	12, rue des Bouchers	Marseille	8
9	Bon app'	12, rue des Bouchers	Marseille	8
10	Bon app'	12, rue des Bouchers	Marseille	8
11	Bon app'	12, rue des Bouchers	Marseille	8
12	Bon app'	12, rue des Bouchers	Marseille	8
13	Bon app'	12, rue des Bouchers	Marseille	8
14	Bon app'	12, rue des Bouchers	Marseille	8

Ordenación de columnas con valores nulos

Dentro de PostgreSQL un valor nulo representa el estado de una celda cuyo valor no ha sido definido o es desconocido en un momento dado. La opción de tratamiento de valores nulos permite definir el modo en el que queremos que los valores nulos sean tratados. Tenemos dos opciones, `NULLS FIRST` ordenará los resultados con valores nulos primero y `NULLS LAST` los posicionará al final de los resultados. En el caso que la opción de nulos se omita, por defecto se usa `NULLS FIRST`.

Vamos a crear una tabla e insertar unos registros para entender mejor la ordenación de nulos:


```
CREATE TABLE demo_nulos (numero INT);

INSERT INTO demo_nulos(numero) VALUES (11),(22),(33),(44),(null);
```

Veamos el resultado de un primer ejemplo, en el que ordenamos los resultados de tal forma que los nulos sean posicionados al final:


```
SELECT numero FROM demo_nulos ORDER BY numero ASC NULLS LAST;
```

Cuyo resultado será:

	numero integer 
1	11
2	22
3	33
4	44
5	[null]

Ahora vamos a modificar la consulta para posicionar los nulos al inicio de los resultados:

```
SELECT numero FROM demo_nulos ORDER BY numero ASC NULLS FIRST;
```

	numero integer 
1	[null]
2	11
3	22
4	33
5	44