https://www.postgresqltutorial.com/

Sección 1. Consulta de datos

SELECT de PostgreSQL

Resumen : en este tutorial, aprenderá a usar la instrucción **SELECT básica de PostgreSQL** para consultar datos de una tabla.

> Tenga en cuenta que si no sabe cómo ejecutar una consulta en la base de datos PostgreSQL utilizando la herramienta de línea de comandos **psql** o la herramienta GUI **pgAdmin** , puede consultar el tutorial de conexión a la base de datos PostgreSQL .

Una de las tareas más comunes, cuando trabaja con la base de datos, es consultar los datos de las tablas mediante la declaración SELECT.

La declaración SELECTes una de las declaraciones más complejas en PostgreSQL. Tiene muchas cláusulas que puede usar para formar una consulta flexible.

Debido a su complejidad, lo dividiremos en muchos tutoriales más cortos y fáciles de entender para que pueda aprender sobre cada cláusula más rápido.

La declaración SELECT tiene las siguientes cláusulas:

- Seleccione filas distintas usando el operador DISTINCT.
- Ordenar filas usando la cláusula ORDER BY.
- Filtrar filas usando la cláusula WHERE.
- Seleccione un subconjunto de filas de una tabla usando la cláusula LIMIT o FETCH.
- Agrupe las filas en grupos usando la cláusula GROUP BY.
- Filtrar grupos usando la cláusula HAVING.
- Únase a otras tablas mediante cláusulas combinaciones como INNER JOIN, LEFT JOIN, FULL OUTER JOIN, CROSS JOIN.
- Realice operaciones de ajuste usando UNION, INTERSECTy EXCEPT.

En este tutorial, se centrará en las cláusulas SELECT y FROM.

SELECT Sintaxis de sentencias de PostgreSQL

Comencemos con la forma básica de la SELECT declaración que recupera datos de una sola tabla.

A continuación se ilustra la sintaxis de la SELECT instrucción:

```
SELECT
    select_list
FROM
    table_name;
```

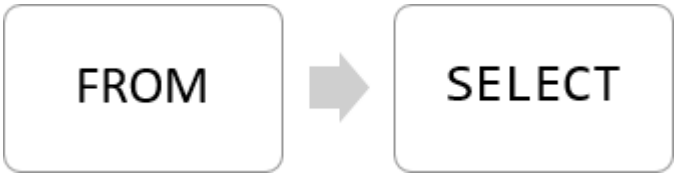
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Examinemos la SELECT declaración con más detalle:

- Primero, especifique una lista de selección que puede ser una columna o una lista de columnas en una tabla de la que desea recuperar datos. Si especifica una lista de columnas, debe colocar una coma (,) entre dos columnas para separarlas. Si desea seleccionar datos de todas las columnas de la tabla, puede usar un asterisco (*) abreviado en lugar de especificar todos los nombres de columna. La lista de selección también puede contener expresiones o valores literales.
- En segundo lugar, especifique el nombre de la tabla desde la que desea consultar los datos después de la FROM palabra clave.

La FROM cláusula es opcional. Si no consulta datos de ninguna tabla, puede omitir la FROMcláusula en la SELECTinstrucción.

PostgreSQL evalúa la FROM cláusula antes de la SELECT cláusula en la SELECT declaración:



Tenga en cuenta que las palabras clave de SQL no distinguen entre mayúsculas y minúsculas. Significa que SELECTes equivalente a selecto Select. Por convención, usaremos todas las palabras clave de SQL en mayúsculas para que las consultas sean más fáciles de leer.

SELECT Ejemplos de PostgreSQL

Echemos un vistazo a algunos ejemplos del uso de SELECT la declaración de PostgreSQL.

Usaremos la siguiente customer tabla en la base de datos de ejemplo para la demostración.

1) Uso de SELECT la declaración de PostgreSQL para consultar datos de un ejemplo de columna

Este ejemplo usa la SELECT declaración para encontrar los nombres de todos los clientes de la customer tabla:

```
SELECT first_name FROM customer;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Aquí está la salida parcial:

	first_name character varying (45)
1	Jared
2	Mary
3	Patricia
4	Linda
5	Barbara
6	Elizabeth
7	Jennifer
8	Maria
9	Susan
10	Margaret

Observe que agregamos un punto y coma (;) al final de la SELECTdeclaración. El punto y coma no forma parte de la instrucción SQL. Se utiliza para señalar a PostgreSQL el final de una declaración SQL. El punto y coma también se usa para separar dos sentencias SQL.

2) SELECT Ejemplo de uso de la declaración de PostgreSQL para consultar datos de varias columnas

Supongamos que solo desea saber el nombre, el apellido y el correo electrónico de los clientes, puede especificar estos nombres de columna en la SELECTcláusula como se muestra en la siguiente consulta:

```
SELECT
    first_name,
    last_name,
    email
FROM
    customer;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	first_name character varying (45)	last_name character varying (45)	email character varying (50)
1	Jared	Ely	jared.ely@sakilacustomer.org
2	Mary	Smith	mary.smith@sakilacustomer.org
3	Patricia	Johnson	patricia.johnson@sakilacustomer.org
4	Linda	Williams	linda.williams@sakilacustomer.org
5	Barbara	Jones	barbara.jones@sakilacustomer.org
6	Elizabeth	Brown	elizabeth.brown@sakilacustomer.org
7	Jennifer	Davis	jennifer.davis@sakilacustomer.org
8	Maria	Miller	maria.miller@sakilacustomer.org
9	Susan	Wilson	susan.wilson@sakilacustomer.org
10	Margaret	Moore	margaret.moore@sakilacustomer.org
11	Dorothy	Taylor	dorothy.taylor@sakilacustomer.org

3) Uso de la declaración SELECT de PostgreSQL para consultar datos de todas las columnas de un ejemplo de tabla

La siguiente consulta usa la declaración SELECT para seleccionar datos de todas las columnas de la tabla customer:

```
SELECT * FROM customer;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	customer_id integer	store_id smallint	first_name character varying (45)	last_name character varying (45)	email character varying (50)	address_id smallint
1	524	1	Jared	Ely	jared.ely@sakilacustomer.org	530
2	1	1	Mary	Smith	mary.smith@sakilacustomer...	5
3	2	1	Patricia	Johnson	patricia.johnson@sakilacust...	6
4	3	1	Linda	Williams	linda.williams@sakilacusto...	7
5	4	2	Barbara	Jones	barbara.jones@sakilacusto...	8
6	5	1	Elizabeth	Brown	elizabeth.brown@sakilacust...	9
7	6	2	Jennifer	Davis	jennifer.davis@sakilacustom...	10
8	7	1	Maria	Miller	maria.miller@sakilacustome...	11
9	8	2	Susan	Wilson	susan.wilson@sakilacustom...	12
10	9	2	Margaret	Moore	margaret.moore@sakilacust...	13
11	10	1	Dorothy	Taylor	dorothy.taylor@sakilacusto...	14

En este ejemplo, usamos un asterisco (*) en la cláusula, SELECT que es una forma abreviada de todas las columnas. En lugar de enumerar todas las columnas en la cláusula SELECT, solo usamos el asterisco (*) para ahorrar algo de escritura.

Sin embargo, no es una buena práctica usar el asterisco (*) en la declaración SELECT cuando incrusta declaraciones SQL en el código de la aplicación como Python , Java , Node.js o PHP debido a las siguientes razones:

1. Rendimiento de la base de datos. Supongamos que tiene una tabla con muchas columnas y muchos datos, la SELECT declaración con el asterisco (*) abreviado seleccionará datos de todas las columnas de la tabla, que pueden no ser necesarios para la aplicación.
2. Rendimiento de la aplicación. La recuperación de datos innecesarios de la base de datos aumenta el tráfico entre el servidor de la base de datos y el servidor de aplicaciones. En consecuencia, sus aplicaciones pueden ser más lentas para responder y menos escalables.

Por estas razones, es una buena práctica especificar explícitamente los nombres de las columnas en la SELECT cláusula siempre que sea posible para obtener solo los datos necesarios de la base de datos.

Y solo debe usar la abreviatura de asterisco (*) para las consultas ad-hoc que examinan datos de la base de datos.

4) Uso de la declaración SELECT de PostgreSQL con el ejemplo de expresiones

El siguiente ejemplo usa la SELECT declaración para devolver los nombres completos y los correos electrónicos de todos los clientes:

```
SELECT
   first_name || ' ' || last_name,
   email
FROM
   customer;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Producción:

	?column? text	email character varying (50)
1	Jared Ely	jared.ely@sakilacustomer.org
2	Mary Smith	mary.smith@sakilacustomer.org
3	Patricia Johnson	patricia.johnson@sakilacustomer.org
4	Linda Williams	linda.williams@sakilacustomer.org
5	Barbara Jones	barbara.jones@sakilacustomer.org
6	Elizabeth Brown	elizabeth.brown@sakilacustomer.org
7	Jennifer Davis	jennifer.davis@sakilacustomer.org
8	Maria Miller	maria.miller@sakilacustomer.org
9	Susan Wilson	susan.wilson@sakilacustomer.org
10	Margaret Moore	margaret.moore@sakilacustomer.org
11	Dorothy Taylor	dorothy.taylor@sakilacustomer.org

En este ejemplo, usamos el operador de concatenación || para concatenar el nombre, el espacio y el apellido de cada cliente.

Aprenderá a usar alias de columna para asignar expresiones con nombres más significativos en el siguiente tutorial.

5) Uso de SELECT la declaración de PostgreSQL con el ejemplo de expresiones

El siguiente ejemplo utiliza la SELECT instrucción con una expresión. Omite la FROM cláusula:

```
SELECT 5 * 3;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Aquí está la salida:

	?column? integer
1	15

Alias de columna de PostgreSQL

Resumen : en este tutorial, aprenderá sobre los alias de columna de PostgreSQL y cómo usar los alias de columna para asignar nombres temporales a las columnas en las consultas.

Introducción a los alias de columna de PostgreSQL

Un alias de columna le permite asignar SELECTun nombre temporal a una columna o una expresión en la lista de selección de una declaración. El alias de columna existe temporalmente durante la ejecución de la consulta.

A continuación, se ilustra la sintaxis del uso de un alias de columna:

```
SELECT column_name AS alias_name
FROM table_name;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

En esta sintaxis, se column_namele asigna un alias alias_name. La ASpalabra clave es opcional, por lo que puede omitirla así:

```
SELECT column_name alias_name
FROM table_name;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

La siguiente sintaxis ilustra cómo establecer un alias para una expresión en la SELECTcláusula:

```
SELECT expression AS alias_name
FROM table_name;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

El propósito principal de los alias de columna es hacer que los encabezados del resultado de una consulta sean más significativos.

Ejemplos de alias de columna de PostgreSQL

Usaremos la customertabla de la base de datos de muestra para mostrarle cómo trabajar con alias de columna.

1) Asignar un alias de columna a un ejemplo de columna

La siguiente consulta devuelve los nombres y apellidos de todos los clientes de la customertabla:

```
SELECT
   first_name,
   last_name
FROM customer;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	first_name character varying (45)	last_name character varying (45)
1	Jared	Ely
2	Mary	Smith
3	Patricia	Johnson
4	Linda	Williams
5	Barbara	Jones
6	Elizabeth	Brown
7	Jennifer	Davis
8	Maria	Miller
9	Susan	Wilson
10	Margaret	Moore

Si desea cambiar el nombre del last_nameencabezado, puede asignarle un nuevo nombre usando un alias de columna como este:

```
SELECT
   first_name,
   last_name AS surname
FROM customer;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Esta consulta asignó surnamecomo el alias de la last_namecolumna:

	first_name character varying (45)	surname character varying (45)
1	Jared	Ely
2	Mary	Smith
3	Patricia	Johnson
4	Linda	Williams
5	Barbara	Jones
6	Elizabeth	Brown
7	Jennifer	Davis
8	Maria	Miller
9	Susan	Wilson
10	Margaret	Moore
11	Dorothy	Taylor

O puede acortarlo eliminando la ASpalabra clave de la siguiente manera:

```
SELECT
   first_name,
   last_name surname
FROM customer;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

2) Asignar un alias de columna a un ejemplo de expresión

La siguiente consulta devuelve los nombres completos de todos los clientes. Construye el nombre completo concatenando el nombre, el espacio y el apellido:

```
SELECT
   first_name || ' ' || last_name
FROM
   customer;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Tenga en cuenta que en PostgreSQL, usa ||como el operador de concatenación que concatena una o más cadenas en una sola cadena.

	?column? text
1	Aaron Selby
2	Adam Gooch
3	Adrian Clary
4	Agnes Bishop
5	Alan Kahn
6	Albert Crouse
7	Alberto Henning
8	Alex Gresham
9	Alexander Fennell
10	Alfred Casillas
11	Alfredo Mcadams
12	Alice Stewart
13	Alicia Mills

Como puede ver claramente en la salida, el encabezado de la columna no es significativo ?column?.

Para solucionar esto, puede asignar a la expresión first_name || ' ' || last_nameun alias de columna, por ejemplo full_name:

```
SELECT
    first_name || ' ' || last_name AS full_name
FROM
    customer;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	full_name text
1	Aaron Selby
2	Adam Gooch
3	Adrian Clary
4	Agnes Bishop
5	Alan Kahn
6	Albert Crouse
7	Alberto Henning
8	Alex Gresham
9	Alexander Fennell
10	Alfred Casillas
11	Alfredo Mcadams
12	Alice Stewart
13	Alicia Mills

3) Alias de columna que contienen espacios

Si un alias de columna contiene uno o más espacios, debe rodearlo con comillas dobles como esta:

```
column_name AS "column_alias"
```
Lenguaje de código: PHP (php)

Por ejemplo:

```
SELECT
    first_name || ' ' || last_name "full name"
FROM
    customer;
```
Lenguaje de código: JavaScript (javascript)

	full name text
1	Jared Ely
2	Mary Smith
3	Patricia Johnson
4	Linda Williams
5	Barbara Jones
6	Elizabeth Brown
7	Jennifer Davis
8	Maria Miller
9	Susan Wilson

Resumen

- Asigne un alias de columna a una columna o expresión utilizando la sintaxis column_name AS alias_nameo expression AS alias_name.
- La ASpalabra clave es opcional.
- Use comillas dobles (") para rodear un alias de columna que contiene espacios.

PostgreSQL ORDER BY

Resumen : en este tutorial, aprenderá cómo ordenar el conjunto de resultados devuelto por la SELECTdeclaración mediante el uso de la cláusula **ORDER BY de PostgreSQL .**

Introducción a ORDER BYla cláusula de PostgreSQL

Cuando consulta datos de una tabla, la SELECTdeclaración devuelve filas en un orden no especificado. Para ordenar las filas del conjunto de resultados, utilice la ORDER BY cláusula en la SELECTinstrucción.

La ORDER BYcláusula le permite ordenar las filas devueltas por una SELECTcláusula en orden ascendente o descendente según una expresión de ordenación.

A continuación se ilustra la sintaxis de la ORDER BY cláusula:

```
SELECT
    select_list
FROM
```

```
      table_name
ORDER BY
      sort_expression1 [ASC | DESC],
        ...
      sort_expressionN [ASC | DESC];
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

En esta sintaxis:

- Primero, especifique una expresión de ordenación, que puede ser una columna o una expresión, que desea ordenar después de las ORDER BYpalabras clave. Si desea ordenar el conjunto de resultados según varias columnas o expresiones, debe colocar una coma (,) entre dos columnas o expresiones para separarlas.
- En segundo lugar, utiliza la ASCopción para ordenar las filas en orden ascendente y la DESCopción para ordenar las filas en orden descendente. Si omite la opción ASCo DESC, los ORDER BY usos ASCpor defecto.

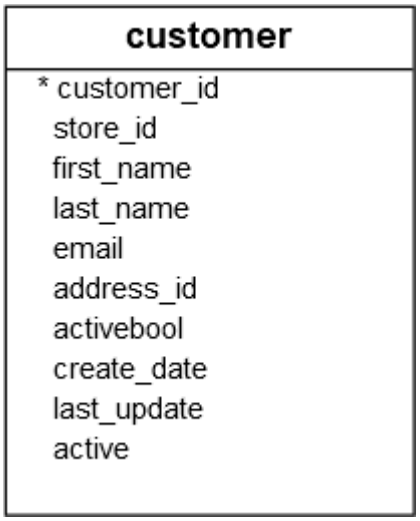
PostgreSQL evalúa las cláusulas de la SELECTdeclaración en el siguiente orden: FROM, SELECT y ORDER BY:

Debido al orden de evaluación, si tiene un alias de columna en la SELECTcláusula, puede usarlo en la ORDER BYcláusula.

Tomemos algunos ejemplos del uso de la ORDER BY cláusula de PostgreSQL.

ORDER BY Ejemplos de PostgreSQL

Usaremos la customertabla en la base de datos de muestra para la demostración.



1) Uso de ORDER BYla cláusula PostgreSQL para ordenar filas por una columna

La siguiente consulta usa la ORDER BYcláusula para ordenar a los clientes por sus nombres en orden ascendente:

```
SELECT
	first_name,
	last_name
FROM
	customer
ORDER BY
	first_name ASC;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	first_name character varying (45)	last_name character varying (45)
1	Aaron	Selby
2	Adam	Gooch
3	Adrian	Clary
4	Agnes	Bishop
5	Alan	Kahn
6	Albert	Crouse
7	Alberto	Henning
8	Alex	Gresham
9	Alexander	Fennell
10	Alfred	Casillas
11	Alfredo	Mcadams
12	Alice	Stewart
13	Alicia	Mills

Dado que la ASCopción es la predeterminada, puede omitirla en la ORDER BYcláusula de esta manera:

```
SELECT
	first_name,
	last_name
FROM
	customer
ORDER BY
	first_name;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

2) Uso de la cláusula PostgreSQL ORDER BYpara ordenar filas por una columna en orden descendente

La siguiente declaración selecciona el nombre y el apellido de la customertabla y ordena las filas por valores en la columna del apellido en orden descendente:

```
SELECT
	first_name,
	last_name
FROM
	customer
```

```
ORDER BY
	last_name DESC;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	first_name character varying (45)	last_name character varying (45)
1	Cynthia	Young
2	Marvin	Yee
3	Luis	Yanez
4	Brian	Wyman
5	Brenda	Wright
6	Tyler	Wren
7	Florence	Woods
8	Lori	Wood
9	Virgil	Wofford
10	Darren	Windham
11	Susan	Wilson
12	Bernice	Willis
13	Gina	Williamson
14	Linda	Williams
15	Jon	Wiles

3) Uso de la cláusula PostgreSQL ORDER BY para ordenar filas por varias columnas

La siguiente declaración selecciona el nombre y el apellido de la tabla de clientes y ordena las filas por el nombre en orden ascendente y el apellido en orden descendente:

```
SELECT
	first_name,
	last_name
FROM
	customer
ORDER BY
	first_name ASC,
	last_name DESC;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	first_name character varying (45)	last_name character varying (45)
321	Kathleen	Adams
322	Kathryn	Coleman
323	Kathy	James
324	Katie	Elliott
325	Kay	Caldwell
326	Keith	Rico
327	Kelly	Torres
328	Kelly	Knott
329	Ken	Prewitt
330	Kenneth	Gooden
331	Kent	Arsenault
332	Kevin	Schuler
333	Kim	Cruz
334	Kimberly	Lee
335	Kirk	Stclair
336	Kristen	Chavez
337	Kristin	Johnston
338	Kristina	Chambers
339	Kurt	Emmons

En este ejemplo, la cláusula ORDER BY ordena primero las filas por valores en la columna de nombre. Y luego ordena las filas ordenadas por valores en la columna del apellido.

Como puede ver claramente en la salida, dos clientes con el mismo nombre Kellytienen el apellido ordenado en orden descendente.

4) Uso de ORDER BYla cláusula PostgreSQL para ordenar filas por expresiones

La función LENGTH() acepta una cadena y devuelve la longitud de esa cadena.

La siguiente declaración selecciona los nombres y sus longitudes. Ordena las filas por la longitud de los nombres:

```
SELECT
	first_name,
	LENGTH(first_name) len
FROM
	customer
ORDER BY
	len DESC;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	first_name character varying (45)	len integer
1	Christopher	11
2	Jacqueline	10
3	Constance	9
4	Katherine	9
5	Nathaniel	9
6	Catherine	9
7	Christian	9
8	Christine	9
9	Charlotte	9
10	Geraldine	9
11	Elizabeth	9
12	Priscilla	9

Debido a que la cláusula ORDER BY se evalúa después de la cláusula SELECT, el alias de la columna len está disponible y se puede usar en la cláusula ORDER BY.

Cláusula ORDER BY de PostgreSQL y NULL

En el mundo de las bases de datos, NULL es un marcador que indica los datos faltantes o los datos son desconocidos en el momento de la grabación.

Cuando ordena filas que contienen NULL, puede especificar el orden de NULL con otros valores no nulos usando la opción NULLS FIRST o NULLS LAST de la cláusula ORDER BY:

```
ORDER BY sort_expresssion [ASC | DESC] [NULLS FIRST | NULLS LAST]
```
Lenguaje de código: CSS (css)

La opción NULLS FIRST se coloca antes NULL de otros valores no nulos y la opción NULL LAST se coloca después NULL de otros valores no nulos.

Vamos a crear una tabla para la demostración.

```
-- create a new table
CREATE TABLE sort_demo(
	num INT
);

-- insert some data
INSERT INTO sort_demo(num)
VALUES(1),(2),(3),(null);
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

> Tenga en cuenta que no necesita comprender las declaraciones CREATE TABLEy INSERT. Solo necesita ejecutarlo desde pgAdmino psqlcrear la sort_demotabla e insertar datos en ella.

La siguiente consulta devuelve datos de la sort_demotabla:

```
SELECT num
FROM sort_demo
ORDER BY num;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	num integer
1	1
2	2
3	3
4	[null]

En este ejemplo, la ORDER BYcláusula ordena los valores de la numcolumna de la sort_demotabla en orden ascendente. Se coloca NULLdespués de otros valores.

Entonces, si usa la ASCopción, la ORDER BYcláusula usa la NULLS LASTopción de manera predeterminada. Por lo tanto, la siguiente consulta devuelve el mismo resultado:

```
SELECT num
FROM sort_demo
ORDER BY num NULLS LAST;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Para colocar NULLantes de otros valores no nulos, utiliza la NULLS FIRSTopción:

```
SELECT num
FROM sort_demo
ORDER BY num NULLS FIRST;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	num integer
1	[null]
2	1
3	2
4	3

La siguiente declaración ordena los valores en la numcolumna de la sort_demotabla en orden descendente:

```
SELECT num
FROM sort_demo
ORDER BY num DESC;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	num integer
1	[null]
2	3
3	2
4	1

Como puede ver claramente en la salida, la ORDER BYcláusula con la DESCopción usa NULLS FIRSTde forma predeterminada.

Para invertir el orden, puede utilizar la NULLS LASTopción:

```
SELECT num
FROM sort_demo
ORDER BY num DESC NULLS LAST;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	num integer
1	3
2	2
3	1
4	[null]

Resumen

- Use la cláusula ORDER BY en la declaración SELECT para ordenar las filas.
- Utilice la opción ASC para ordenar las filas en orden ascendente y la opción DESC para ordenar las filas en orden descendente. La cláusula ORDER BY usa la opción ASC por defecto.
- Utilice las opciones NULLS FIRST y NULLS LAST para especificar explícitamente el orden de NULL con otros valores no nulos.

PostgreSQL SELECT DISTINCT

Resumen : en este tutorial, aprenderá a usar la cláusula SELECT DISTINCT de PostgreSQL para eliminar filas duplicadas de un conjunto de resultados devuelto por una consulta.

Introducción a la cláusula SELECT DISTINCT de PostgreSQL

La DISTINCTcláusula se usa en la SELECTinstrucción para eliminar filas duplicadas de un conjunto de resultados. La DISTINCTcláusula mantiene una fila para cada grupo de duplicados. La DISTINCTcláusula se puede aplicar a una o más columnas en la lista de selección de la SELECTinstrucción.

A continuación se ilustra la sintaxis de la DISTINCT cláusula:

```
SELECT
   DISTINCT column1
FROM
   table_name;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

En esta sentencia, los valores de la column1columna se utilizan para evaluar el duplicado.

Si especifica varias columnas, la DISTINCTcláusula evaluará el duplicado en función de la combinación de valores de estas columnas.

```
SELECT
   DISTINCT column1, column2
FROM
   table_name;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

En este caso, la combinación de valores en ambas columnas column1y column2se utilizará para evaluar el duplicado.

PostgreSQL también proporciona la DISTINCT ON (expression) opción de mantener la "primera" fila de cada grupo de duplicados usando la siguiente sintaxis:

```
SELECT
   DISTINCT ON (column1) column_alias,
   column2
FROM
   table_name
ORDER BY
   column1,
   column2;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

El orden de las filas devueltas por la SELECTdeclaración no se especifica, por lo tanto, la "primera" fila de cada grupo del duplicado tampoco se especifica.

Es una buena práctica usar siempre la ORDER BYcláusula con para DISTINCT ON(expression) hacer que el conjunto de resultados sea predecible.

Observe que la DISTINCT ONexpresión debe coincidir con la expresión más a la izquierda en la ORDER BYcláusula.

SELECT DISTINCT Ejemplos de PostgreSQL

Vamos a crear una nueva tabla llamada distinct_demoe insertar datos en ella para practicar la DISTINCTcláusula.

> Tenga en cuenta que aprenderá a crear una tabla e insertar datos en una tabla en el siguiente tutorial. En este tutorial, solo ejecuta la declaración en psql o pgAdmin para ejecutar las declaraciones.

Primero, use la siguiente CREATE TABLEdeclaración para crear la distinct_demotabla que consta de tres columnas: id, bcolory fcolor.

```
CREATE TABLE distinct_demo (
      id serial NOT NULL PRIMARY KEY,
      bcolor VARCHAR,
      fcolor VARCHAR
);
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

En segundo lugar, inserte algunas filas en la distinct_demotabla usando la siguiente INSERTdeclaración:

```
INSERT INTO distinct_demo (bcolor, fcolor)
VALUES
	('red', 'red'),
	('red', 'red'),
	('red', NULL),
	(NULL, 'red'),
	('red', 'green'),
	('red', 'blue'),
	('green', 'red'),
	('green', 'blue'),
	('green', 'green'),
	('blue', 'red'),
	('blue', 'green'),
	('blue', 'blue');
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Tercero, consulta los datos de la distinct_demotabla usando la SELECTdeclaración:

```
SELECT
	id,
	bcolor,
	fcolor
FROM
	distinct_demo ;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	id integer	bcolor character varying	fcolor character varying
1	1	red	red
2	2	red	red
3	3	red	[null]
4	4	[null]	red
5	5	red	green
6	6	red	blue
7	7	green	red
8	8	green	blue
9	9	green	green
10	10	blue	red
11	11	blue	green
12	12	blue	blue

DISTINCT Ejemplo de una columna de PostgreSQL

La siguiente declaración selecciona valores únicos en la bcolorcolumna de la t1tabla y ordena el conjunto de resultados en orden alfabético usando la ORDER BYcláusula.

```
SELECT
	DISTINCT bcolor
FROM
```

```
	distinct_demo
ORDER BY
	bcolor;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	bcolor character varying
1	blue
2	green
3	red
4	[null]

DISTINCTColumnas múltiples de PostgreSQL

La siguiente declaración demuestra cómo usar la DISTINCTcláusula en varias columnas:

```
SELECT
	DISTINCT bcolor,
	fcolor
FROM
	distinct_demo
ORDER BY
	bcolor,
	fcolor;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	bcolor character varying	fcolor character varying
1	blue	blue
2	blue	green
3	blue	red
4	green	blue
5	green	green
6	green	red
7	red	blue
8	red	green
9	red	red
10	red	[null]
11	[null]	red

Debido a que especificamos las columnas bcolory fcoloren la SELECT DISTINCT cláusula, PostgreSQL combinó los valores en las columnas bcolory fcolorpara evaluar la singularidad de las filas.

La consulta devuelve la combinación única de bcolory fcolorde la distinct_demo tabla. Observe que la distinct_demotabla tiene dos filas con redvalor en ambas bcolorcolumnas fcolor. Cuando aplicamos a DISTINCT ambas columnas, se eliminó una fila del conjunto de resultados porque es el duplicado.

DISTINCT ON Ejemplo de PostgreSQL

La siguiente declaración ordena el conjunto de resultados por bcolory fcolor, y luego, para cada grupo de duplicados, mantiene la primera fila en el conjunto de resultados devuelto.

```
SELECT
	DISTINCT ON (bcolor) bcolor,
	fcolor
FROM
	distinct_demo
ORDER BY
	bcolor,
	fcolor;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Aquí está la salida:

	bcolor character varying	fcolor character varying
1	blue	blue
2	green	blue
3	red	blue
4	[null]	red

En este tutorial, aprendió a usar SELECT DISTINCTla declaración de PostgreSQL para eliminar filas duplicadas devueltas por una consulta.

Seccion 2. Filtrado de Datos

PostgreSQL WHERE

Resumen : en este tutorial, aprenderá cómo usar WHERE la cláusula de PostgreSQL para filtrar las filas devueltas por una SELECTdeclaración.

La SELECTinstrucción devuelve todas las filas de una o más columnas de una tabla. Para seleccionar filas que cumplan una condición específica, utilice una WHERECláusula.

WHEREDescripción general de la cláusula de PostgreSQL

La sintaxis de la WHERECláusula de PostgreSQL es la siguiente:

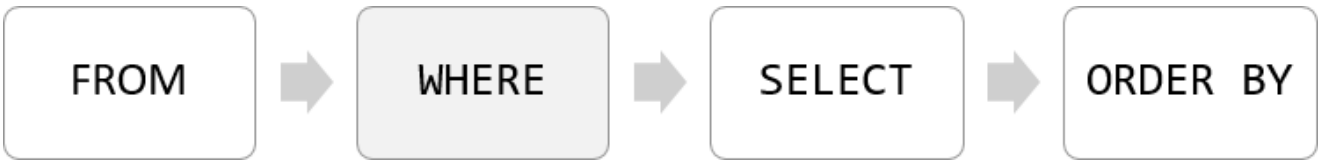
```
SELECT select_list
FROM table_name
WHERE condition
ORDER BY sort_expression
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

La WHERECláusula aparece justo después de la FROMCláusula de la SELECTinstrucción. La WHERECláusula usa conditionpara filtrar las filas devueltas de la SELECTCláusula.

Debe conditionevaluarse como verdadero, falso o desconocido. Puede ser una expresión booleana o una combinación de expresiones booleanas usando los operadores ANDy OR.

La consulta devuelve solo las filas que cumplen los requisitos conditionde la WHERE cláusula. En otras palabras, solo conditionse incluirán en el conjunto de resultados las filas que hagan que las evaluaciones sean verdaderas.

PostgreSQL evalúa la WHERE cláusula después de la FROM cláusula y antes de la cláusula SELECTy :ORDER BY



Si usa alias de columna en la SELECT cláusula, no puede usarlos en la WHERE cláusula.

Además de la SELECT declaración, puede usar la WHERE cláusula en la declaración UPDATEy DELETEpara especificar las filas que se actualizarán o eliminarán.

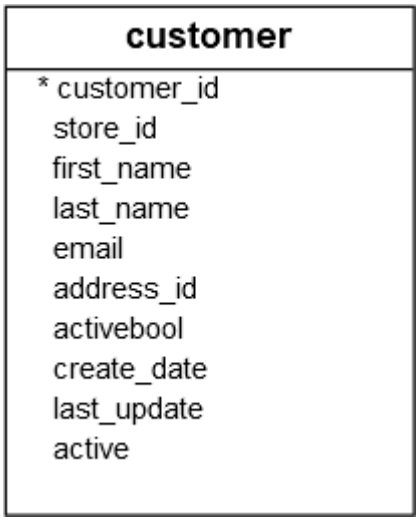
Para formar la condición en la WHERE cláusula, utiliza operadores lógicos y de comparación:

Operador	**Descripción**
=	Igual
>	Mas grande que
<	Menos que
>=	Mayor que o igual
<=	Menor o igual
<> o !=	No es igual
Y	Operador lógico Y
O	Operador lógico O
IN	Devuelve verdadero si un valor coincide con cualquier valor en una lista
BETWEEN	Retorna verdadero si un valor está entre un rango de valores
LIKE	Devuelve verdadero si un valor coincide con un patrón

Operador	Descripción
IS NULL	Retorna verdadero si un valor es NULL
NO	Negar el resultado de otros operadores

Ejemplos de la cláusula WHERE de PostgreSQL

Practiquemos con algunos ejemplos del uso de la WHEREcláusula. Usaremos la customer tabla de la base de datos de ejemplo para la demostración.



1) Ejemplo de uso de la cláusula WHERE con el operador igual = ( )

La siguiente declaración utiliza la WHERE cláusula clientes cuyos nombres son Jamie:

```
SELECT
	last_name,
	first_name
FROM
	customer
WHERE
	first_name = 'Jamie';
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	last_name character varying (45)	first_name character varying (45)
1	Rice	Jamie
2	Waugh	Jamie

2) Uso de la cláusula WHERE con el AND ejemplo del operador

El siguiente ejemplo encuentra clientes cuyo nombre y apellido son Jamiey riceusa el ANDoperador lógico para combinar dos expresiones booleanas:

```
SELECT
	last_name,
```

```
	first_name
FROM
	customer
WHERE
	first_name = 'Jamie' AND
	  last_name = 'Rice';
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	last_name character varying (45)	first_name character varying (45)
1	Rice	Jamie

3) Usar la cláusula WHERE con el OR ejemplo del operador

Este ejemplo busca los clientes cuyo apellido Rodriguezo nombre es Adammediante el ORoperador:

```
SELECT
	first_name,
	last_name
FROM
	customer
WHERE
	last_name = 'Rodriguez' OR
	first_name = 'Adam';
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	first_name character varying (45)	last_name character varying (45)
1	Laura	Rodriguez
2	Adam	Gooch

4) Uso de la cláusula WHERE con el IN ejemplo del operador

Si desea hacer coincidir una cadena con cualquier cadena en una lista, puede usar el IN operador.

Por ejemplo, la siguiente declaración devuelve clientes cuyo nombre es Ann, o Anne, o Annie:

```
SELECT
	first_name,
	last_name
FROM
	customer
WHERE
	first_name IN ('Ann','Anne','Annie');
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	first_name character varying (45)	last_name character varying (45)
1	Ann	Evans
2	Anne	Powell
3	Annie	Russell

5) Usar la cláusula WHERE con el LIKE ejemplo del operador

Para encontrar una cadena que coincida con un patrón específico, utilice el LIKE operador. El siguiente ejemplo devuelve todos los clientes cuyos nombres comienzan con la cadena Ann:

```
SELECT
	first_name,
	last_name
FROM
	customer
WHERE
	first_name LIKE 'Ann%'
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	first_name character varying (45)	last_name character varying (45)
1	Anna	Hill
2	Ann	Evans
3	Anne	Powell
4	Annie	Russell
5	Annette	Olson

se %llama un comodín que coincide con cualquier cadena. El patrón 'Ann%' coincide con cualquier cadena que comience con 'Ann'.

6) Uso de la cláusula WHERE con el ejemplo del operador BETWEEN

El siguiente ejemplo encuentra clientes cuyos nombres comienzan con la letra Ay contienen de 3 a 5 caracteres mediante el uso del BETWEEN operador.

El BETWEENoperador devuelve verdadero si un valor está en un rango de valores.

```
SELECT
	first_name,
	LENGTH(first_name) name_length
FROM
	customer
WHERE
	first_name LIKE 'A%' AND
	LENGTH(first_name) BETWEEN 3 AND 5
ORDER BY
	name_length;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	first_name character varying (45)	name_length integer
1	Amy	3
2	Ann	3
3	Ana	3
4	Andy	4
5	Anna	4
6	Anne	4
7	Alma	4
8	Adam	4
9	Alan	4
10	Alex	4
11	Angel	5
12	Agnes	5
13	Andre	5
14	Aaron	5
15	Allan	5
16	Allen	5
17	Alice	5
18	Alvin	5
19	Anita	5
20	Amber	5
21	April	5
22	Annie	5

En este ejemplo, usamos la función LENGTH() obtiene el número de caracteres de una cadena de entrada.

7) Ejemplo de uso de la cláusula WHERE con el operador distinto (<>)

Este ejemplo encuentra clientes cuyos nombres comienzan con Bra y los apellidos no son Motley:

```
SELECT
	first_name,
	last_name
FROM
	customer
WHERE
	first_name LIKE 'Bra%' AND
	last_name <> 'Motley';
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	first_name character varying (45)	last_name character varying (45)
1	Brandy	Graves
2	Brandon	Huey
3	Brad	Mccurdy

Tenga en cuenta que puede usar el !=operador y <>el operador indistintamente porque son equivalentes.

En este tutorial, aprendió a usar WHEREla cláusula de PostgreSQL en la SELECTinstrucción para filtrar filas en función de una condición específica.

LÍMIT de PostgreSQL

Resumen : en este tutorial, aprenderá a usar la cláusula **LIMIT de PostgreSQL** para obtener un subconjunto de filas generadas por una consulta.

Introducción a la cláusula LIMIT de PostgreSQL

PostgreSQL LIMIT es una cláusula opcional de la SELECT declaración que restringe el número de filas devueltas por la consulta.

A continuación se ilustra la sintaxis de la LIMIT cláusula:

```
SELECT select_list
FROM table_name
ORDER BY sort_expression
LIMIT row_count
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

La instrucción devuelve row_count filas generadas por la consulta. Si row_count es cero, la consulta devuelve un conjunto vacío. En caso de que row_count sea NULL, la consulta devuelve el mismo conjunto de resultados ya que no tiene la LIMIT cláusula.

En caso de que desee omitir una cantidad de filas antes de devolver las row_count filas, use OFFSETla cláusula colocada después de la LIMIT cláusula como la siguiente declaración:

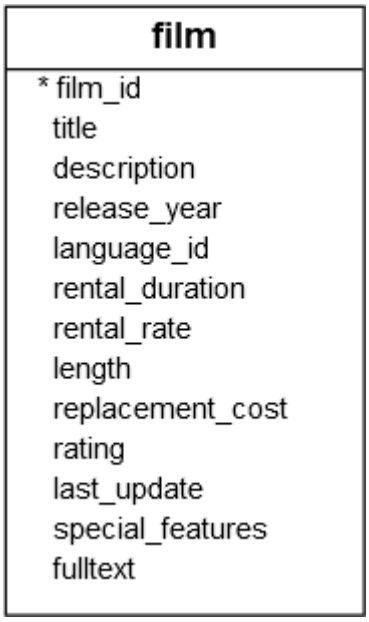
```
SELECT select_list
FROM table_name
LIMIT row_count OFFSET row_to_skip;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

La declaración primero omite row_to_skipfilas antes de devolver row_countfilas generadas por la consulta. Si row_to_skipes cero, la declaración funcionará como si no tuviera la OFFSETcláusula.

Debido a que una tabla puede almacenar filas en un orden no especificado, cuando usa la LIMITcláusula, siempre debe usar la ORDER BYcláusula para controlar el orden de las filas. Si no usa la ORDER BYcláusula, puede obtener un conjunto de resultados con el orden de filas no especificado.

LIMIT Ejemplos de PostgreSQL

Tomemos algunos ejemplos del uso de la LIMITcláusula de PostgreSQL. Usaremos la filmtabla en la base de datos de muestra para la demostración.



1) Uso de PostgreSQL LIMIT para restringir el número de filas devueltas ejemplo

Este ejemplo usa la cláusula LIMIT para ordenar las primeras cinco películas por film_id:

```
SELECT
	film_id,
	title,
	release_year
FROM
	film
ORDER BY
	film_id
LIMIT 5;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	film_id integer	title character varying (255)	release_year integer
1	1	Academy Dinosaur	2006
2	2	Ace Goldfinger	2006
3	3	Adaptation Holes	2006
4	4	Affair Prejudice	2006
5	5	African Egg	2006

2) Uso de PostgreSQL LIMIT con ejemplo de DESPLAZAMIENTO

Para recuperar 4 películas a partir de la cuarta ordenada por film_id, utilice las cláusulas LIMITy OFFSET de la siguiente manera:

```
SELECT
	film_id,
	title,
	release_year
FROM
	film
ORDER BY
	film_id
```

```
LIMIT 4 OFFSET 3;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	film_id integer	title character varying (255)	release_year integer
1	4	Affair Prejudice	2006
2	5	African Egg	2006
3	6	Agent Truman	2006
4	7	Airplane Sierra	2006

3) Usando PostgreSQL LIMIT OFFSSET para obtener N filas superiores / inferiores

Por lo general, a menudo usa la cláusula LIMIT para seleccionar filas con los valores más altos o más bajos de una tabla.

Por ejemplo, para obtener las 10 películas más caras en términos de alquiler, ordena las películas por la tarifa de alquiler en orden descendente y usa la LIMITcláusula para obtener las 10 primeras películas. La siguiente consulta ilustra la idea:

```
SELECT
	film_id,
	title,
	rental_rate
FROM
	film
ORDER BY
	rental_rate DESC
LIMIT 10;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

El resultado de la consulta es el siguiente:

	film_id integer	title character varying (255)	rental_rate numeric (4,2)
1	13	Ali Forever	4.99
2	20	Amelie Hellfighters	4.99
3	7	Airplane Sierra	4.99
4	10	Aladdin Calendar	4.99
5	2	Ace Goldfinger	4.99
6	8	Airport Pollock	4.99
7	98	Bright Encounters	4.99
8	133	Chamber Italian	4.99
9	384	Grosse Wonderful	4.99
10	21	American Circus	4.99

En este tutorial, ha aprendido a usar la LIMIT OFFSET cláusula de PostgreSQL para recuperar un subconjunto de filas devueltas por una consulta.

FETCH de PostgreSQL

Resumen : en este tutorial, aprenderá a usar la cláusula FETCH de PostgreSQL para recuperar una parte de las filas devueltas por una consulta.

Introducción a la cláusula FETCH de PostgreSQL

Para restringir el número de filas devueltas por una consulta, a menudo usa la LIMITcláusula. La LIMITcláusula es ampliamente utilizada por muchos sistemas de administración de bases de datos relacionales como MySQL, H2 y HSQLDB. Sin embargo, la LIMITcláusula no es un estándar SQL.

Para cumplir con el estándar SQL, PostgreSQL admite la FETCHcláusula para recuperar una cantidad de filas devueltas por una consulta. Tenga en cuenta que la FETCHcláusula se introdujo en SQL:2008.

A continuación se ilustra la sintaxis de la FETCHcláusula de PostgreSQL:

```
OFFSET start { ROW | ROWS }
FETCH { FIRST | NEXT } [ row_count ] { ROW | ROWS } ONLY
Lenguaje de código:  SQL (lenguaje de consulta estructurado)  ( sql )
```

En esta sintaxis:

- ROW es el sinónimo de ROWS, FIRST es el sinónimo de NEXT. Entonces puedes usarlos indistintamente
- El start es un número entero que debe ser cero o positivo. De forma predeterminada, es cero si OFFSET no se especifica la cláusula. En caso de que start sea mayor que el número de filas en el conjunto de resultados, no se devuelve ninguna fila;
- El row_countes 1 o mayor. De forma predeterminada, el valor predeterminado de row_countes 1 si no lo especifica explícitamente.

Debido a que no se especifica el orden de las filas almacenadas en la tabla, siempre debe usar la cláusula FETCH con la cláusula ORDER BY para que el orden de las filas en el conjunto de resultados devuelto sea coherente.

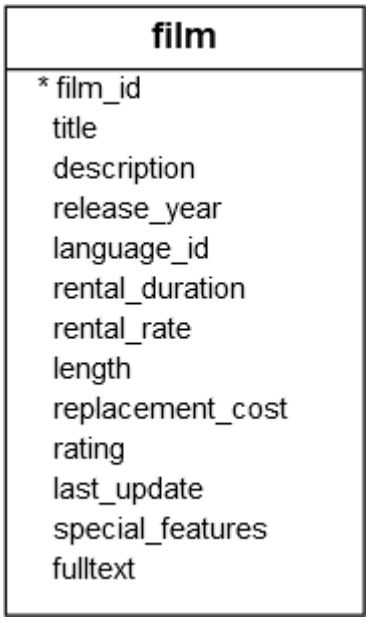
Tenga en cuenta que la cláusula OFFSET debe ir antes de la cláusula FETCH en SQL:2008. Sin embargo, las cláusulas OFFSET y FETCH pueden aparecer en cualquier orden en PostgreSQL.

FETCH frente a LIMIT

La cláusula FETCH es funcionalmente equivalente a la cláusula LIMIT. Si planea hacer que su aplicación sea compatible con otros sistemas de bases de datos, debe usar la cláusula FETCH porque sigue el estándar SQL.

Ejemplos de FETCH de PostgreSQL

Usemos la tabla film en la base de datos de ejemplo para la demostración.



La siguiente consulta usa la cláusula FETCH para seleccionar la primera película ordenada por títulos en orden ascendente:

```
SELECT
    film_id,
    title
FROM
    film
ORDER BY
    title
FETCH FIRST ROW ONLY;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	film_id integer	title character varying (255)
1	1	Academy Dinosaur

Es equivalente a la siguiente consulta:

```
SELECT
    film_id,
    title
FROM
    film
ORDER BY
    title
FETCH FIRST 1 ROW ONLY;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

La siguiente consulta usa la cláusula FETCH para seleccionar las primeras cinco películas ordenadas por títulos:

```
SELECT
    film_id,
    title
FROM
    film
ORDER BY
```

```
    title
FETCH FIRST 5 ROW ONLY;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	film_id integer	title character varying (255)
1	1	Academy Dinosaur
2	2	Ace Goldfinger
3	3	Adaptation Holes
4	4	Affair Prejudice
5	5	African Egg

La siguiente declaración devuelve las siguientes cinco películas después de las primeras cinco películas ordenadas por títulos:

```
SELECT
    film_id,
    title
FROM
    film
ORDER BY
    title
OFFSET 5 ROWS
FETCH FIRST 5 ROW ONLY;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	film_id integer	title character varying (255)
1	6	Agent Truman
2	7	Airplane Sierra
3	8	Airport Pollock
4	9	Alabama Devil
5	10	Aladdin Calendar

En este tutorial, ha aprendido a usar la cláusula FETCH de PostgreSQL para recuperar una parte de las filas devueltas por una consulta.

PostgreSQL IN

Resumen : en este tutorial, aprenderá cómo usar el operador **IN de PostgreSQL** en la cláusula WHERE para verificar si un valor coincide con algún valor en una lista.

Sintaxis del operador PostgreSQL IN

Utiliza el operador IN en la cláusula WHERE para verificar si un valor coincide con cualquier valor en una lista de valores.

La sintaxis del operador IN es la siguiente:

```
value IN (value1,value2,...)
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

El INoperador devuelve verdadero si valuecoincide con cualquier valor de la lista, es decir, value1, value2, ...

La lista de valores puede ser una lista de valores literales como números, cadenas o el resultado de una SELECTdeclaración como esta:

```
value IN (SELECT column_name FROM table_name);
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

La consulta entre paréntesis se denomina subconsulta , que es una consulta anidada dentro de otra consulta. Tenga en cuenta que aprenderá más sobre la subconsulta en el siguiente tutorial

Ejemplos de operadores IN de PostgreSQL

Supongamos que desea conocer la información de alquiler de los ID de cliente 1 y 2, puede usar el IN operador en la WHERE cláusula de la siguiente manera:

```
SELECT customer_id,
       rental_id,
       return_date
FROM
       rental
WHERE
       customer_id IN (1, 2)
ORDER BY
       return_date DESC;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	customer_id smallint	rental_id integer	return_date timestamp without time zone
1	2	15145	2005-08-31 15:51:04
2	1	15315	2005-08-30 01:51:46
3	2	14743	2005-08-29 00:18:56
4	1	15298	2005-08-28 22:49:37
5	2	14475	2005-08-27 08:59:32
6	1	14825	2005-08-27 07:01:57
7	2	15907	2005-08-25 23:23:35
8	2	12963	2005-08-23 11:37:04
9	1	13176	2005-08-23 08:50:54
10	1	14762	2005-08-23 01:30:57
11	1	12250	2005-08-22 23:05:29
12	1	13068	2005-08-20 14:44:16

La siguiente consulta utiliza los operadores igual (=) y OR en lugar del operador IN. Es equivalente a la consulta anterior:

```
SELECT
       rental_id,
       customer_id,
       return_date
```

```
FROM
	rental
WHERE
	customer_id = 1 OR customer_id = 2
ORDER BY
	return_date DESC;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

La consulta que usa el INoperador es más corta y más legible que la consulta que usa los operadores igual (=) y OR. Además, PostgreSQL ejecuta la consulta con el INoperador mucho más rápido que la misma consulta que utiliza una lista de ORoperadores.

Operador PostgreSQL NOT IN

Puede combinar el INoperador con el NOT operador para seleccionar filas cuyos valores no coincidan con los valores de la lista.

Por ejemplo, la siguiente declaración encuentra todos los alquileres con la identificación del cliente que no es 1 o 2.

```
SELECT
	customer_id,
	rental_id,
	return_date
FROM
	rental
WHERE
	customer_id NOT IN (1, 2);
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	customer_id smallint	rental_id integer	return_date timestamp without time zone
1	459	2	2005-05-28 19:40:33
2	408	3	2005-06-01 22:12:39
3	333	4	2005-06-03 01:43:41
4	222	5	2005-06-02 04:33:21
5	549	6	2005-05-27 01:32:07
6	269	7	2005-05-29 20:34:53
7	239	8	2005-05-27 23:33:46
8	126	9	2005-05-28 00:22:40
9	399	10	2005-05-31 22:44:21
10	142	11	2005-06-02 20:56:02
11	261	12	2005-05-30 05:44:27

Similar al IN operador, puede usar los operadores no iguales (<>) y ANDpara escribir el NOT IN operador:

```
SELECT
	customer_id,
	rental_id,
```

```
	return_date
FROM
	rental
WHERE
	customer_id <> 1
AND customer_id <> 2;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Esta consulta devuelve el mismo resultado que la consulta anterior que usa el NOT IN operador.

PostgreSQL IN con una subconsulta

La siguiente consulta devuelve una lista de ID de clientes de la rentaltabla con la fecha de devolución 2005-05-27:

```
SELECT customer_id
FROM rental
WHERE CAST (return_date AS DATE) = '2005-05-27'
ORDER BY customer_id;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	customer_id smallint
1	37
2	47
3	48
4	65
5	73
6	75
7	93
8	114
9	119
10	131
11	158

Debido a que esta consulta devuelve una lista de valores, puede usarla como la entrada del INoperador de esta manera:

```
SELECT
	customer_id,
	first_name,
	last_name
FROM
	customer
WHERE
	customer_id IN (
		SELECT customer_id
		FROM rental
		WHERE CAST (return_date AS DATE) = '2005-05-27'
	)
ORDER BY customer_id;
```

```
Lenguaje de código:  SQL (lenguaje de consulta estructurado)  ( sql )
```

	customer_id integer	first_name character varying (45)	last_name character varying (45)
1	37	Pamela	Baker
2	47	Frances	Parker
3	48	Ann	Evans
4	65	Rose	Howard
5	73	Beverly	Brooks
6	75	Tammy	Sanders
7	93	Phyllis	Foster
8	114	Grace	Ellis
9	119	Sherry	Marshall
10	131	Monica	Hicks
11	158	Veronica	Stone
12	167	Sally	Pierce

Para obtener más información sobre la subconsulta, consulte el tutorial de subconsulta .

En este tutorial, ha aprendido a utilizar el INoperador de PostgreSQL para verificar si un valor coincide con algún valor en una lista de valores.

PostgreSQL BETWEEN

Resumen : en este tutorial, aprenderá a usar el operador **BETWEEN de PostgreSQL** para hacer coincidir un valor con un rango de valores.

Introducción al operador BETWEEN de PostgreSQL

El operador se utiliza BETWEEN para hacer coincidir un valor con un rango de valores. A continuación se ilustra la sintaxis del BETWEEN operador:

```
value BETWEEN low AND high;
Lenguaje de código:  SQL (lenguaje de consulta estructurado)  ( sql )
```

Si valuees mayor o igual que el lowvalor y menor o igual que el highvalor, la expresión devuelve verdadero, de lo contrario, devuelve falso.

Puede reescribir el BETWEENoperador usando los operadores mayor o igual (>=) o menor o igual (<=) de esta manera:

```
value >= low and value <= high
Lenguaje de código:  SQL (lenguaje de consulta estructurado)  ( sql )
```

Si desea verificar si un valor está fuera de un rango, combine el NOT operador con el BETWEEN operador de la siguiente manera:

```
value NOT BETWEEN low AND high;
Lenguaje de código:  SQL (lenguaje de consulta estructurado)  ( sql )
```

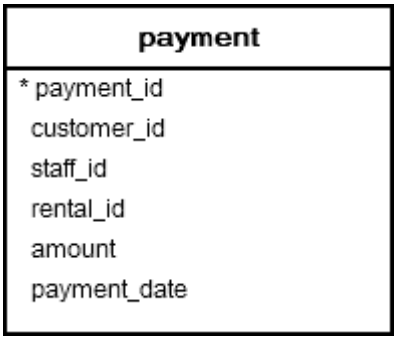
La siguiente expresión es equivalente a la expresión que utiliza los operadores NOT y :BETWEEN

```
value < low OR value > high
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

A menudo usa el BETWEEN operador en la cláusula WHERE de una declaración SELECT , INSERT , UPDATE o DELETE .

PostgreSQL ENTRE ejemplos de operadores

Echemos un vistazo a la payment tabla en la base de datos de ejemplo .



La siguiente consulta utiliza el operador BETWEEN para seleccionar pagos cuyo monto esté entre 8 y 9 (USD):

```
SELECT
	customer_id,
	payment_id,
	amount
FROM
	payment
WHERE
	amount BETWEEN 8 AND 9;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	customer_id smallint	payment_id integer	amount numeric (5,2)
1	343	17517	8.99
2	347	17529	8.99
3	347	17532	8.99
4	348	17535	8.99
5	349	17540	8.99
6	379	17648	8.99
7	403	17747	8.99
8	409	17775	8.99
9	423	17817	8.99

Para obtener pagos cuyo monto no está en el rango de 8 y 9, utiliza la siguiente consulta:

```
SELECT
	customer_id,
```

```
	payment_id,
	amount
FROM
	payment
WHERE
	amount NOT BETWEEN 8 AND 9;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	customer_id smallint	payment_id integer	amount numeric (5,2)
1	341	17503	7.99
2	341	17504	1.99
3	341	17505	7.99
4	341	17506	2.99
5	341	17507	7.99
6	341	17508	5.99
7	342	17509	5.99
8	342	17510	5.99
9	342	17511	2.99
10	343	17512	4.99
11	343	17513	6.99

Si desea comparar un valor con rangos de fechas, debe usar la fecha literal en formato ISO 8601, es decir, AAAA-MM-DD. Por ejemplo, para obtener el pago cuya fecha de pago está entre 2007-02-07 y 2007-02-15, utilice la siguiente consulta:

```
SELECT
	customer_id,
	payment_id,
	amount,
 payment_date
FROM
	payment
WHERE
	payment_date BETWEEN '2007-02-07' AND '2007-02-15';
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	customer_id smallint	payment_id integer	amount numeric (5,2)	payment_date timestamp without time zone
1	368	17610	0.99	2007-02-14 23:25:11.996577
2	370	17617	6.99	2007-02-14 23:33:58.996577
3	402	17743	4.99	2007-02-14 23:53:34.996577
4	416	17793	2.99	2007-02-14 21:21:59.996577
5	432	17854	5.99	2007-02-14 23:07:27.996577
6	481	18051	2.99	2007-02-14 22:03:35.996577
7	512	18155	6.99	2007-02-14 22:57:03.996577
8	516	18173	4.99	2007-02-14 21:23:39.996577
9	546	18276	1.99	2007-02-14 23:10:43.996577
10	561	18322	2.99	2007-02-14 23:52:46.996577
11	592	18441	6.99	2007-02-14 21:41:12.996577

En este tutorial, ha aprendido a usar BETWEEN el operador de PostgreSQL para seleccionar un valor que se encuentra en un rango de valores.

PostgreSQL LIKE

Resumen : en este tutorial, aprenderá a usar LIKE y ILIKE operadores de PostgreSQL para consultar datos usando coincidencias de patrones.

Introducción al operador LIKE de PostgreSQL

Suponga que desea encontrar un cliente pero no recuerda exactamente su nombre. Sin embargo, solo recuerda que su nombre comienza con algo como Jen.

¿Cómo se encuentra el cliente exacto de la base de datos? Puede encontrar al cliente en la customertabla mirando la columna del nombre para ver si hay algún valor que comience con Jen. Llevará mucho tiempo si la tabla de clientes tiene muchas filas.

Afortunadamente, puede usar el LIKE operador de PostgreSQL para hacer coincidir el nombre del cliente con una cadena como esta consulta:

```
SELECT
	first_name,
	  last_name
FROM
	customer
WHERE
	first_name LIKE 'Jen%';
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	first_name character varying (45)	last_name character varying (45)
1	Jennifer	Davis
2	Jennie	Terry
3	Jenny	Castro

Observe que la WHERE cláusula contiene una expresión especial: el first_name, el LIKEoperador y una cadena que contiene un signo de porcentaje (%). La cadena 'Jen%'se llama patrón.

La consulta devuelve filas cuyos valores en la columna first_name comienzan con Jeny pueden ir seguidos de cualquier secuencia de caracteres. Esta técnica se llama coincidencia de patrones.

Construye un patrón combinando valores literales con caracteres comodín y usa el operador LIKEo NOT LIKEpara encontrar las coincidencias. PostgreSQL le proporciona dos comodines:

- El signo de porcentaje (%) coincide con cualquier secuencia de cero o más caracteres.
- El signo de subrayado (_) coincide con cualquier carácter único.

La sintaxis del operador PostgreSQL LIKEes la siguiente:

```
value LIKE pattern
Lenguaje de código:  SQL (lenguaje de consulta estructurado)  ( sql )
```

La expresión devuelve verdadero si valuecoincide con pattern.

Para negar el LIKEoperador, utilice el NOToperador de la siguiente manera:

```
value NOT LIKE pattern
Lenguaje de código:  SQL (lenguaje de consulta estructurado)  ( sql )
```

El NOT LIKEoperador devuelve verdadero cuando valueno coincide con pattern.

Si el patrón no contiene ningún carácter comodín, el LIKEoperador se comporta como el =operador igual ( ).

Operador LIKE de PostgreSQL: ejemplos de coincidencia de patrones

Tomemos algunos ejemplos del uso del operador LIKE

Ejemplos de LIKE de PostgreSQL

Vea el siguiente ejemplo:

```
SELECT
      'foo' LIKE 'foo', -- true
      'foo' LIKE 'f%', -- true
      'foo' LIKE '_o_', -- true
      'bar' LIKE 'b_'; -- false
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Cómo funciona.

- La primera expresión devuelve verdadero porque el foopatrón no contiene ningún carácter comodín, por lo que el LIKEoperador actúa como el =operador igual ( ).
- La segunda expresión devuelve verdadero porque coincide con cualquier cadena que comience con la letra fy sea seguida por cualquier número de caracteres.
- La tercera expresión devuelve verdadero porque el patrón (_o_) coincide con cualquier cadena que comience con cualquier carácter único, seguido de la letra oy finalice con cualquier carácter único.
- La cuarta expresión devuelve falso porque el patrón b_coincide con cualquier cadena que comience con la letra by sea seguida por un solo carácter.

Es posible utilizar comodines al principio y/o al final del patrón.

Por ejemplo, la siguiente consulta devuelve clientes cuyo nombre contiene ercadenas como Jenifer, Kimberly, etc.

```
SELECT
	first_name,
	last_name
FROM
	customer
WHERE
	first_name LIKE '%er%'
ORDER BY
	first_name;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	first_name character varying (45)	last_name character varying (45)
1	Albert	Crouse
2	Alberto	Henning
3	Alexander	Fennell
4	Amber	Dixon
5	Bernard	Colby
6	Bernice	Willis
7	Bertha	Ferguson
8	Beverly	Brooks
9	Catherine	Campbell
10	Cheryl	Murphy
11	Chester	Benner

Puede combinar el porcentaje (%) con el guión bajo (_) para construir un patrón como el siguiente ejemplo:

```
SELECT
	first_name,
	last_name
```

```
FROM
	customer
WHERE
	first_name LIKE '_her%'
ORDER BY
	first_name;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	first_name character varying (45)	last_name character varying (45)
1	Cheryl	Murphy
2	Sherri	Rhodes
3	Sherry	Marshall
4	Theresa	Watson

El patrón _her%coincide con cualquier cadena que:

- Comience con cualquier carácter individual (_)
- Y es seguido por la cadena literal her.
- Y termina con cualquier número de caracteres.

Los primeros nombres devueltos son C **her** yl, S **her** ri, S **her** ry y T **her** asa.

PostgreSQL NO ME GUSTA ejemplos

La siguiente consulta utiliza el NOT LIKEoperador para encontrar clientes cuyos nombres no comiencen con Jen:

```
SELECT
	first_name,
	last_name
FROM
	customer
WHERE
	first_name NOT LIKE 'Jen%'
ORDER BY
	first_name
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	first_name character varying (45)	last_name character varying (45)
1	Aaron	Selby
2	Adam	Gooch
3	Adrian	Clary
4	Agnes	Bishop
5	Alan	Kahn
6	Albert	Crouse
7	Alberto	Henning
8	Alex	Gresham
9	Alexander	Fennell
10	Alfred	Casillas
11	Alfredo	Mcadams

Extensiones PostgreSQL del operador LIKE

PostgreSQL admite el ILIKEoperador que funciona como el LIKEoperador. Además, el ILIKEoperador coincide con el valor sin distinción entre mayúsculas y minúsculas. Por ejemplo:

```
SELECT
	first_name,
	last_name
FROM
	customer
WHERE
	first_name ILIKE 'BAR%';
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	first_name character varying (45)	last_name character varying (45)
1	Barbara	Jones
2	Barry	Lovelace

El BAR%patrón coincide con cualquier cadena que comience con BAR, Bar, BaR, etc. Si usa el LIKEoperador en su lugar, la consulta no devolverá ninguna fila.

PostgreSQL también proporciona algunos operadores que actúan como el operador LIKE, NOT LIKE, ILIKEand NOT ILIKEcomo se muestra a continuación:

Operador	**Equivalente**
~~	COMO
~~*	DOBLAR
!~~	DIFERENTE A

Operador	Equivalente
!~~*	NO ME GUSTA

En este tutorial, ha aprendido a usar PostgreSQL LIKEy ILIKEoperadores para consultar datos mediante la coincidencia de patrones.

PostgreSQL IS NULL

Resumen : en este tutorial, aprenderá cómo usar el IS NULLoperador PostgreSQL para verificar si un valor es NULL o no.

Introducción NULLy IS NULLoperador

En el mundo de las bases de datos, NULL significa información faltante o no aplicable. NULL no es un valor, por lo tanto, no puede compararlo con ningún otro valor como números o cadenas. La comparación de NULL con un valor siempre dará como resultado NULL, lo que significa un resultado desconocido.

Además, NULL no es igual a NULL, por lo que la siguiente expresión devuelve NULL:

```
NULL = NULL
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Suponiendo que tiene una contactstabla que almacena el nombre, el apellido, el correo electrónico y el número de teléfono de los contactos. En el momento de grabar el contacto, es posible que no sepa el número de teléfono del contacto.

Para solucionar esto, defina la phonecolumna como una columna anulable e inserte NULL en la phonecolumna cuando guarde la información de contacto.

```
CREATE TABLE contacts(
    id INT GENERATED BY DEFAULT AS IDENTITY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(255) NOT NULL,
    phone VARCHAR(15),
    PRIMARY KEY (id)
);
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

> Tenga en cuenta que aprenderá a crear una nueva tabla en el siguiente tutorial. Por ahora, solo necesita ejecutar la instrucción anterior para crear la contactstabla.

Si obtiene un error al ejecutar la CREATE TABLEdeclaración, es posible que su versión de PostgreSQL no admita la sintaxis de la columna de identidad . En este caso, puede utilizar la siguiente declaración:

```
CREATE TABLE contacts(
```

```
    id SERIAL,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(255) NOT NULL,
    phone VARCHAR(15),
    PRIMARY KEY (id)
);
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

La siguiente declaración inserta dos contactos, uno tiene un número de teléfono y el otro no:

```
INSERT INTO contacts(first_name, last_name, email, phone)
VALUES ('John','Doe','john.doe@example.com',NULL),
    ('Lily','Bush','lily.bush@example.com','(408-234-2764)');
```
Lenguaje de código: PHP (php)

Para encontrar el contacto que no tiene un número de teléfono, puede encontrar la siguiente declaración:

```
SELECT
    id,
    first_name,
    last_name,
    email,
    phone
FROM
    contacts
WHERE
    phone = NULL;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

La sentencia no devuelve ninguna fila. Esto se debe a que la expresión `phone = NULL`en la `WHERE`cláusula siempre devuelve falso.

Aunque hay un NULL en la columna del teléfono, la expresión `NULL = NULL`devuelve falso. Esto se debe a que NULL no es igual a ningún valor, ni siquiera a sí mismo.

Para verificar si un valor es NULL o no, usa el `IS NULL`operador en su lugar:

```
value IS NULL
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

La expresión devuelve verdadero si el valor es NULL o falso si no lo es.

Entonces, para obtener el contacto que no tiene ningún número de teléfono almacenado en la columna de teléfono, use la siguiente declaración en su lugar:

```
SELECT
    id,
    first_name,
    last_name,
    email,
    phone
FROM
    contacts
```

```
WHERE
    phone IS NULL;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Aquí está la salida:

	id integer	first_name character varying (50)	last_name character varying (50)	email character varying (255)	phone character varying (15)
1	1	John	Doe	john.doe@example.com	[null]

IS NOT NULL Operador PostgreSQL

Para comprobar si un valor no es NULL, utiliza el IS NOT NULLoperador:

```
value IS NOT NULL
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

La expresión devuelve verdadero si el valor no es NULL o falso si el valor es NULL.

Por ejemplo, para encontrar el contacto que tiene un número de teléfono, utilice la siguiente declaración:

```
SELECT
    id,
    first_name,
    last_name,
    email,
    phone
FROM
    contacts
WHERE
    phone IS NOT NULL;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

La salida es:

	id integer	first_name character varying (50)	last_name character varying (50)	email character varying (255)	phone character varying (15)
1	2	Lily	Bush	lily.bush@example.com	(408-234-2764)

En este tutorial, ha aprendido a utilizar el IS NULLoperador de PostgreSQL para comprobar si un valor es NULL o no.

Uniones Join de PostgreSQL

Resumen : en este tutorial, aprenderá sobre varios tipos de combinaciones de PostgreSQL, incluidas la combinación interna, la combinación izquierda, la combinación derecha y la combinación externa completa.

La combinación de PostgreSQL se usa para combinar columnas de una (autocombinación) o más tablas en función de los valores de las columnas comunes entre las tablas relacionadas. Las columnas comunes suelen ser las columnas de clave principal de la primera tabla y las columnas de clave externa de la segunda tabla.

PostgreSQL es compatible con la combinación interna , la combinación izquierda , la combinación derecha , la combinación externa completa , la combinación cruzada , la combinación natural y un tipo especial de combinación llamada autocombinación .

Configuración de tablas de muestra

Supongamos que tiene dos mesas llamadas basket_ay basket_bque almacenan frutas:

```
CREATE TABLE basket_a (
    a INT PRIMARY KEY,
    fruit_a VARCHAR (100) NOT NULL
);

CREATE TABLE basket_b (
    b INT PRIMARY KEY,
    fruit_b VARCHAR (100) NOT NULL
);

INSERT INTO basket_a (a, fruit_a)
VALUES
    (1, 'Apple'),
    (2, 'Orange'),
    (3, 'Banana'),
    (4, 'Cucumber');

INSERT INTO basket_b (b, fruit_b)
VALUES
    (1, 'Orange'),
    (2, 'Apple'),
    (3, 'Watermelon'),
    (4, 'Pear');
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Las tablas tienen algunas frutas comunes como appley orange.

La siguiente instrucción devuelve datos de la basket_atabla:

	a integer	fruit_a character varying (100)
1	1	Apple
2	2	Orange
3	3	Banana
4	4	Cucumber

Y la siguiente declaración devuelve datos de la basket_btabla:

	b integer	fruit_b character varying (100)
1	1	Orange
2	2	Apple
3	3	Watermelon
4	4	Pear

Inner Join de PostgreSQL

La siguiente declaración une la primera tabla (basket_a) con la segunda tabla (basket_b) haciendo coincidir los valores en las columnas fruit_ay :fruit_b

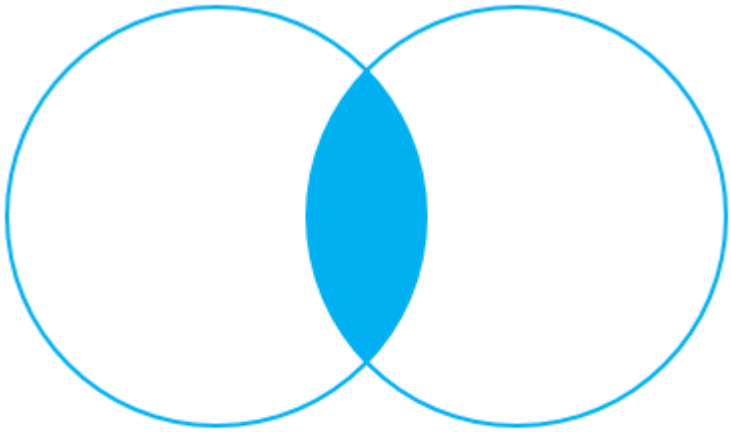
```
SELECT
    a,
    fruit_a,
    b,
    fruit_b
FROM
    basket_a
INNER JOIN basket_b
    ON fruit_a = fruit_b;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	a integer	fruit_a character varying (100)	b integer	fruit_b character varying (100)
1	1	Apple	2	Apple
2	2	Orange	1	Orange

La combinación interna examina cada fila en la primera tabla (basket_a). Compara el valor de la fruit_acolumna con el valor de la fruit_bcolumna de cada fila de la segunda tabla (basket_b). Si estos valores son iguales, la combinación interna crea una nueva fila que contiene columnas de ambas tablas y agrega esta nueva fila al conjunto de resultados.

El siguiente diagrama de Venn ilustra la unión interna:



INNER JOIN

LEFT JOIN de PostgreSQL

La siguiente declaración usa la cláusula de combinación izquierda para unir la basket_atabla con la basket_btabla. En el contexto de combinación izquierda, la primera tabla se denomina tabla izquierda y la segunda tabla se denomina tabla derecha.

```
SELECT
    a,
    fruit_a,
    b,
    fruit_b
FROM
    basket_a
LEFT JOIN basket_b
   ON fruit_a = fruit_b;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

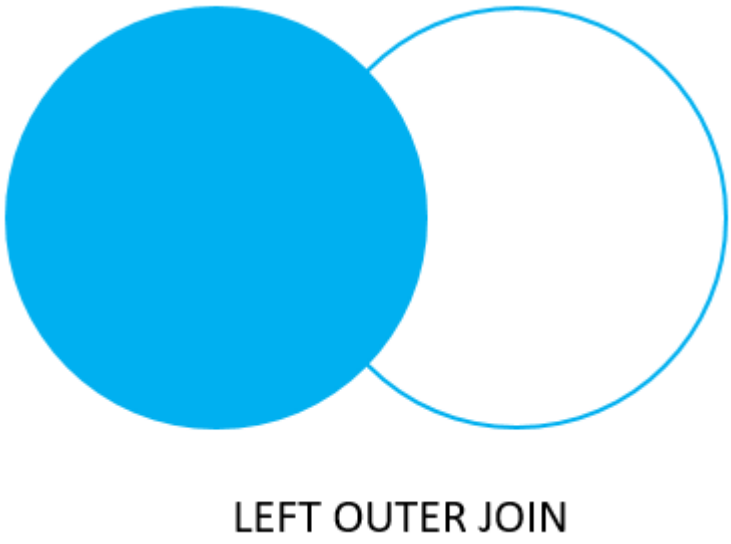
	a integer	fruit_a character varying (100)	b integer	fruit_b character varying (100)
1	1	Apple	2	Apple
2	2	Orange	1	Orange
3	3	Banana	[null]	[null]
4	4	Cucumber	[null]	[null]

La combinación izquierda comienza a seleccionar datos de la tabla izquierda. Compara los valores de la columna fruit_a con los valores de la columna fruit_b de la tabla basket_b.

Si estos valores son iguales, la combinación izquierda crea una nueva fila que contiene columnas de ambas tablas y agrega esta nueva fila al conjunto de resultados. (ver la fila #1 y #2 en el conjunto de resultados).

En caso de que los valores no sean iguales, la combinación izquierda también crea una nueva fila que contiene columnas de ambas tablas y la agrega al conjunto de resultados. Sin embargo, llena las columnas de la tabla derecha (basket_b) con nulo. (ver la fila #3 y #4 en el conjunto de resultados).

El siguiente diagrama de Venn ilustra la combinación izquierda:



Para seleccionar filas de la tabla de la izquierda que no tienen filas coincidentes en la tabla de la derecha, utilice la combinación izquierda con una WHERE cláusula. Por ejemplo:

```
SELECT
    a,
    fruit_a,
    b,
    fruit_b
FROM
    basket_a
LEFT JOIN basket_b
    ON fruit_a = fruit_b
WHERE b IS NULL;
```

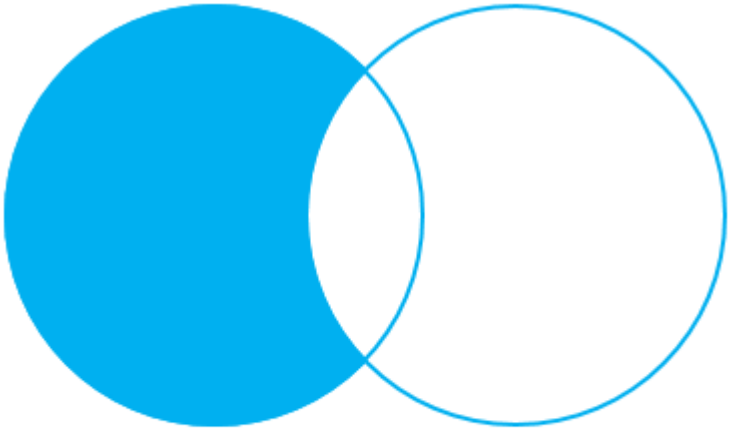
```
Lenguaje de código:  SQL (lenguaje de consulta estructurado)  ( sql )
```

La salida es:

	a integer	fruit_a character varying (100)	b integer	fruit_b character varying (100)
1	3	Banana	[null]	[null]
2	4	Cucumber	[null]	[null]

> Tenga en cuenta que el LEFT JOIN es el mismo que el, LEFT OUTER JOINpor lo que puede usarlos indistintamente.

El siguiente diagrama de Venn ilustra la unión izquierda que devuelve filas de la tabla de la izquierda que no tienen filas coincidentes de la tabla de la derecha:



LEFT OUTER JOIN – only rows from the left table

RIGHT JOIN de PostgreSQL

La combinación derecha es una versión invertida de la combinación izquierda. La combinación derecha comienza a seleccionar datos de la tabla derecha. Compara cada valor de la columna fruit_b de cada fila de la tabla de la derecha con cada valor de la columna fruit_a de cada fila de la tabla fruit_a.

Si estos valores son iguales, la combinación derecha crea una nueva fila que contiene columnas de ambas tablas.

En caso de que estos valores no sean iguales, la combinación derecha también crea una nueva fila que contiene columnas de ambas tablas. Sin embargo, llena las columnas de la tabla de la izquierda con NULL.

La siguiente declaración usa la unión derecha para unir la basket_atabla con la basket_btabla:

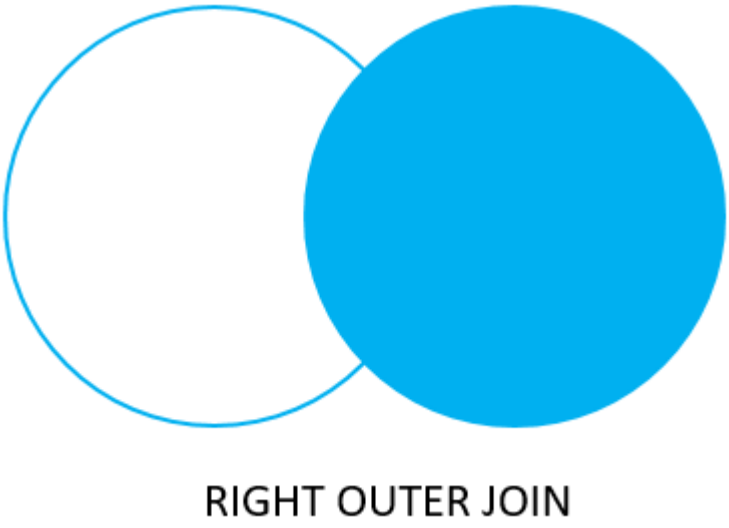
```
SELECT
    a,
    fruit_a,
    b,
    fruit_b
FROM
    basket_a
RIGHT JOIN basket_b ON fruit_a = fruit_b;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Aquí está la salida:

	a integer	fruit_a character varying (100)	b integer	fruit_b character varying (100)
1	2	Orange	1	Orange
2	1	Apple	2	Apple
3	[null]	[null]	3	Watermelon
4	[null]	[null]	4	Pear

El siguiente diagrama de Venn ilustra la unión derecha:



De manera similar, puede obtener filas de la tabla de la derecha que no tengan filas coincidentes de la tabla de la izquierda agregando una WHERE cláusula de la siguiente manera:

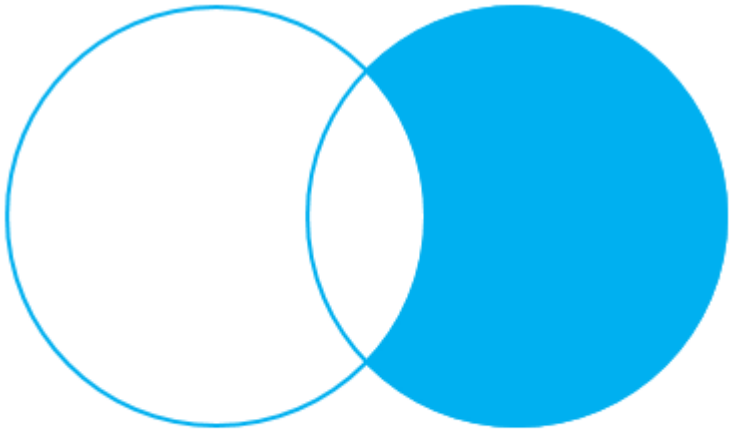
```
SELECT
    a,
    fruit_a,
    b,
    fruit_b
FROM
    basket_a
RIGHT JOIN basket_b
   ON fruit_a = fruit_b
WHERE a IS NULL;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	a integer	fruit_a character varying (100)	b integer	fruit_b character varying (100)
1	[null]	[null]	3	Watermelon
2	[null]	[null]	4	Pear

> Los RIGHT JOIN y RIGHT OUTER JOIN son lo mismo, por lo tanto, puede usarlos indistintamente.

El siguiente diagrama de Venn ilustra la combinación derecha que devuelve filas de la tabla derecha que no tienen filas coincidentes en la tabla izquierda:



RIGHT OUTER JOIN – only rows from the right table

FULL OUTER JOIN de PostgreSQL

La combinación externa completa o la combinación completa devuelve un conjunto de resultados que contiene todas las filas de las tablas izquierda y derecha, con las filas coincidentes de ambos lados, si están disponibles. En caso de que no haya coincidencia, las columnas de la tabla se rellenarán con NULL.

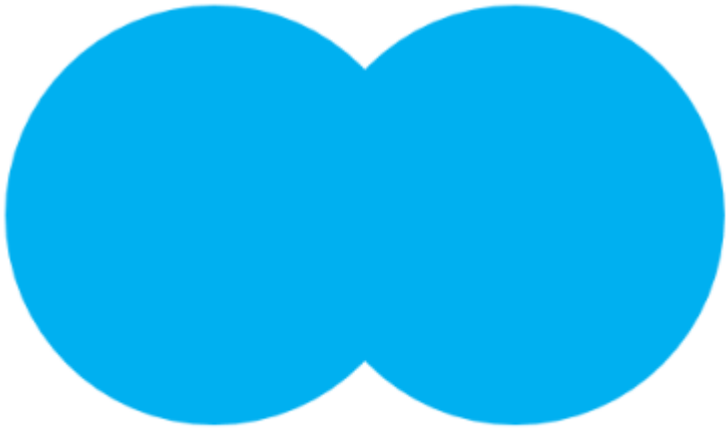
```
SELECT
    a,
    fruit_a,
    b,
    fruit_b
FROM
    basket_a
FULL OUTER JOIN basket_b
    ON fruit_a = fruit_b;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Producción:

	a integer	fruit_a character varying (100)	b integer	fruit_b character varying (100)
1	1	Apple	2	Apple
2	2	Orange	1	Orange
3	3	Banana	[null]	[null]
4	4	Cucumber	[null]	[null]
5	[null]	[null]	3	Watermelon
6	[null]	[null]	4	Pear

El siguiente diagrama de Venn ilustra la unión externa completa:



Para devolver filas en una tabla que no tienen filas coincidentes en la otra, usa la unión completa con una WHERE cláusula como esta:

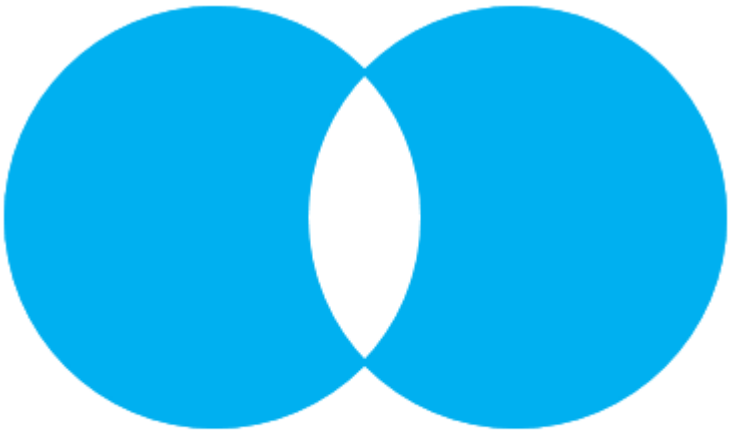
```
SELECT
    a,
    fruit_a,
    b,
    fruit_b
FROM
    basket_a
FULL JOIN basket_b
   ON fruit_a = fruit_b
WHERE a IS NULL OR b IS NULL;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Aquí está el resultado:

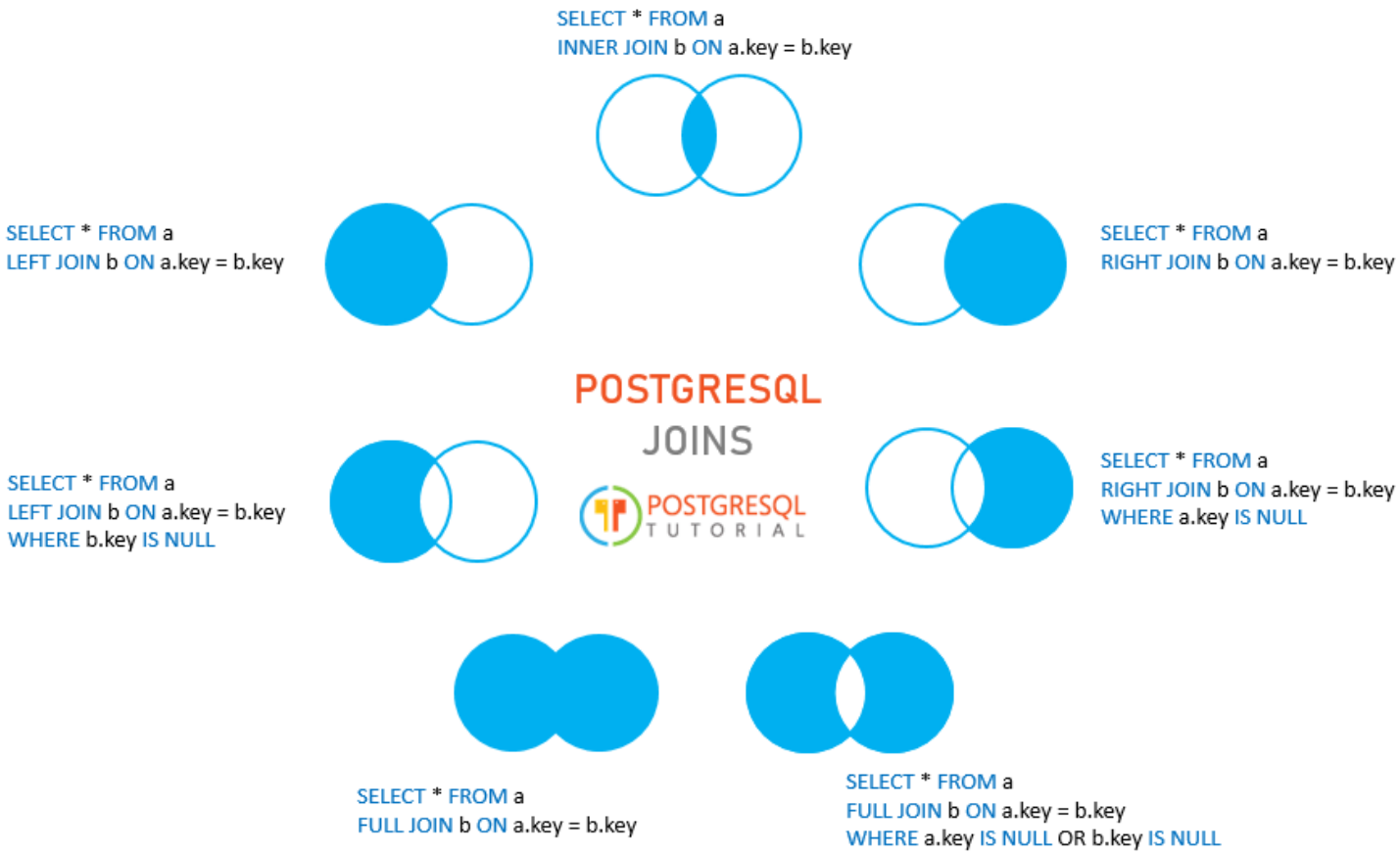
	a integer	fruit_a character varying (100)	b integer	fruit_b character varying (100)
1	3	Banana	[null]	[null]
2	4	Cucumber	[null]	[null]
3	[null]	[null]	3	Watermelon
4	[null]	[null]	4	Pear

El siguiente diagrama de Venn ilustra la unión externa completa que devuelve filas de una tabla que no tienen las filas correspondientes en la otra tabla:



FULL OUTER JOIN – only
rows unique to both tables

La siguiente imagen muestra todas las uniones de PostgreSQL que discutimos hasta ahora con la sintaxis detallada:



En este tutorial, aprendió a usar varios tipos de uniones de PostgreSQL para combinar datos de varias tablas relacionadas.

alias_name de PostgreSQL

Resumen : en este tutorial, aprenderá sobre los alias de tablas de PostgreSQL y sus aplicaciones prácticas.

Introducción a los alias de tablas de PostgreSQL

Los alias de tabla asignan temporalmente nuevos nombres a las tablas durante la ejecución de una consulta.

A continuación se ilustra la sintaxis de un alias de tabla:

```
table_name AS alias_name;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

En esta sintaxis, se `table_name`le asigna un alias como `alias_name`. Similar a los alias de columna , la `AS`palabra clave es opcional. Significa que omites la `AS`palabra clave así:

```
table_name alias_name;
```

Aplicaciones prácticas de los alias de tablas

Los alias de tabla tienen varias aplicaciones prácticas.

1) Usar alias de tabla para el nombre largo de la tabla para que las consultas sean más legibles

Si debe calificar un nombre de columna con un nombre de tabla largo, puede usar un alias de tabla para ahorrar algo de escritura y hacer que su consulta sea más legible.

Por ejemplo, en lugar de usar la siguiente expresión en una consulta:

```
a_very_long_table_name.column_name
```
Lenguaje de código: CSS (css)

puede asignar a la tabla `a_very_long_table_name`un alias como este:

```
a_very_long_table_name AS alias
```
Lenguaje de código: PHP (php)

Y haga referencia `column_name`en la tabla `a_very_long_table_name`usando el alias de la tabla:

```
alias.column_name
```
Lenguaje de código: CSS (css)

2) Uso de alias de tabla en cláusulas de combinación

Por lo general, a menudo usa una cláusula de combinación para consultar datos de varias tablas que tienen el mismo nombre de columna.

Si usa el mismo nombre de columna que proviene de varias tablas sin calificarlas por completo, obtendrá un error.

Para evitar este error, debe calificar estas columnas con la siguiente sintaxis:

```
table_name.column_name
```
Lenguaje de código: CSS (css)

Para acortar la consulta, puede utilizar los alias de tabla para los nombres de tabla enumerados en las cláusulas FROMy INNER JOIN. Por ejemplo:

```
SELECT
	c.customer_id,
	first_name,
	amount,
	payment_date
FROM
	customer c
INNER JOIN payment p
    ON p.customer_id = c.customer_id
ORDER BY
   payment_date DESC;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

3) Uso de alias de tabla en la autounión

Cuando se une a una tabla a sí misma (también conocida como auto-unión), debe usar alias de tabla. Esto se debe a que hacer referencia a la misma tabla varias veces dentro de una consulta genera un error.

El siguiente ejemplo muestra cómo hacer referencia a la employeetabla dos veces en la misma consulta utilizando los alias de la tabla:

```
SELECT
    e.first_name employee,
    m .first_name manager
FROM
    employee e
INNER JOIN employee m
    ON m.employee_id = e.manager_id
ORDER BY manager;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

En este tutorial, ha aprendido a utilizar los alias de tablas de PostgreSQL para asignar temporalmente nuevos nombres a las tablas durante la ejecución de una consulta.

UNIÓN INTERNA de PostgreSQL

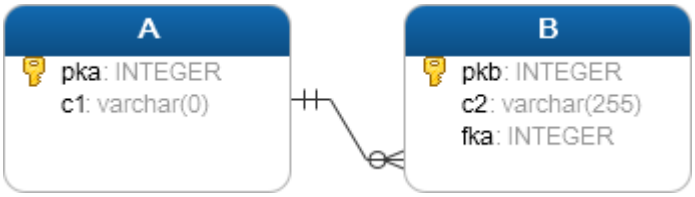
Resumen : en este tutorial, aprenderá cómo seleccionar datos de varias tablas utilizando la cláusula **INNER JOIN de PostgreSQL** .

Introducción a la cláusula INNER JOIN de PostgreSQL

En una base de datos de relaciones, los datos normalmente se distribuyen en más de una tabla. Para seleccionar datos completos, a menudo necesita consultar datos de varias tablas.

En este tutorial, nos estamos enfocando en cómo combinar datos de varias tablas usando la INNER JOINcláusula.

Suponga que tiene dos tablas A y B. La tabla A tiene una columna pka cuyo valor coincide con los valores de la columna fka de la tabla B.



Para seleccionar datos de ambas tablas, utilice la INNER JOINcláusula en la SELECTdeclaración de la siguiente manera:

```
SELECT
	pka,
	c1,
	pkb,
	c2
FROM
	A
INNER JOIN B ON pka = fka;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Para unir tabla Acon tabla B, sigue estos pasos:

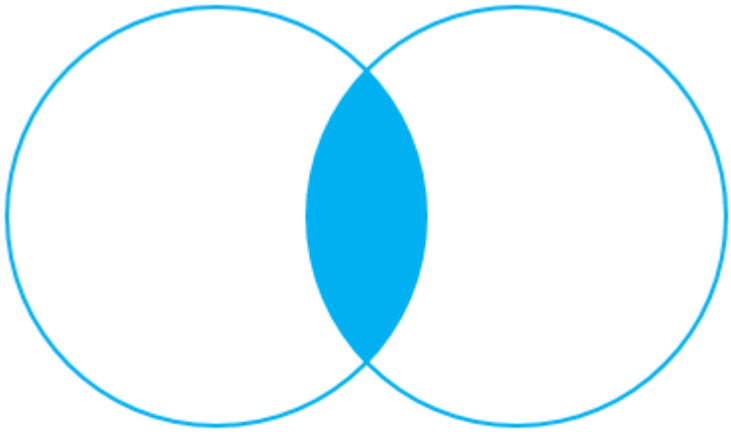
- Primero, especifique las columnas de ambas tablas en las que desea seleccionar datos en la SELECTcláusula.
- En segundo lugar, especifique la tabla principal, es decir, la tabla A en la FROMcláusula.
- En tercer lugar, especifique la segunda tabla (tabla B) en la INNER JOINcláusula y proporcione una condición de combinación después de la ONpalabra clave.

Cómo INNER JOINfunciona.

Para cada fila de la tabla A, la combinación interna compara el valor de la columna pka con el valor de la columna fka de cada fila de la tabla B:

- Si estos valores son iguales, la combinación interna crea una nueva fila que contiene todas las columnas de ambas tablas y la agrega al conjunto de resultados.
- En caso de que estos valores no sean iguales, la unión interna simplemente los ignora y pasa a la siguiente fila.

El siguiente diagrama de Venn ilustra cómo INNER JOIN funciona la cláusula.



INNER JOIN

La mayoría de las veces, las tablas que desea unir tendrán columnas con el mismo nombre, por ejemplo, idcolumna como customer_id.

Si hace referencia a columnas con el mismo nombre de diferentes tablas en una consulta, obtendrá un error. Para evitar el error, debe calificar estas columnas por completo con la siguiente sintaxis:

```
table_name.column_name
```
Lenguaje de código: CSS (css)

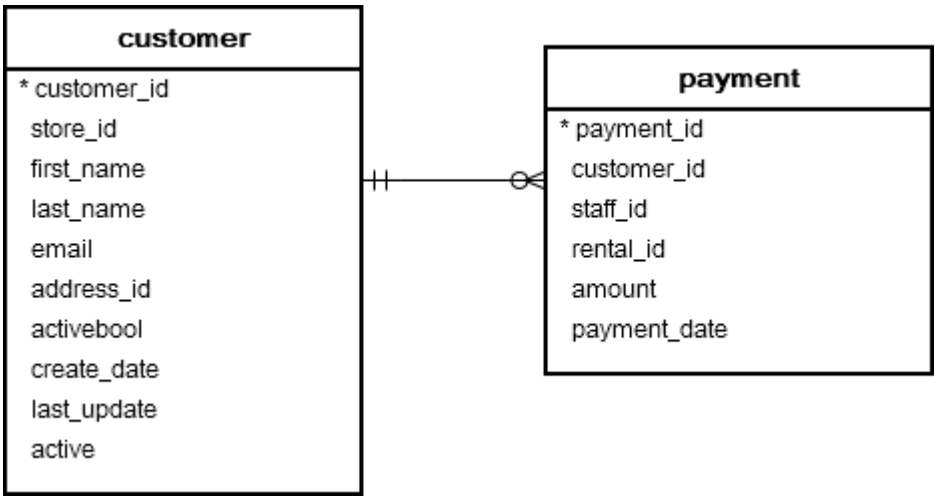
En la práctica, utilizará alias de tabla para asignar nombres cortos a las tablas unidas para que la consulta sea más legible.

Ejemplos de UNIÓN INTERNA de PostgreSQL

Tomemos algunos ejemplos del uso de la INNER JOINcláusula.

1) Usando PostgreSQL INNER JOIN para unir dos tablas

Echemos un vistazo a las tablas customery paymenten la base de datos de ejemplo .



En estas tablas, cada vez que un cliente realiza un pago, se inserta una nueva fila en la paymenttabla.

Cada cliente puede tener cero o muchos pagos. Sin embargo, cada pago pertenece a uno y solo un cliente. La customer_idcolumna establece la relación entre las dos tablas.

La siguiente declaración usa la INNER JOINcláusula para seleccionar datos de ambas tablas:

```
SELECT
	customer.customer_id,
	first_name,
	last_name,
	amount,
	payment_date
FROM
	customer
INNER JOIN payment
    ON payment.customer_id = customer.customer_id
ORDER BY payment_date;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	customer_id integer	first_name character varying (45)	last_name character varying (45)	amount numeric (5,2)	payment_date timestamp without time zone
1	416	Jeffery	Pinson	2.99	2007-02-14 21:21:59.996577
2	516	Elmer	Noe	4.99	2007-02-14 21:23:39.996577
3	239	Minnie	Romero	4.99	2007-02-14 21:29:00.996577
4	592	Terrance	Roush	6.99	2007-02-14 21:41:12.996577
5	49	Joyce	Edwards	0.99	2007-02-14 21:44:52.996577
6	264	Gwendolyn	May	3.99	2007-02-14 21:44:53.996577
7	46	Catherine	Campbell	4.99	2007-02-14 21:45:29.996577
8	481	Herman	Devore	2.99	2007-02-14 22:03:35.996577
9	139	Amber	Dixon	2.99	2007-02-14 22:11:22.996577

La siguiente consulta devuelve el mismo resultado. Sin embargo, utiliza alias de tabla:

```
SELECT
	c.customer_id,
	first_name,
	last_name,
	email,
	amount,
	payment_date
FROM
	customer c
INNER JOIN payment p
    ON p.customer_id = c.customer_id
WHERE
    c.customer_id = 2;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Dado que ambas tablas tienen la misma customer_idcolumna, puede usar la USINGsintaxis:

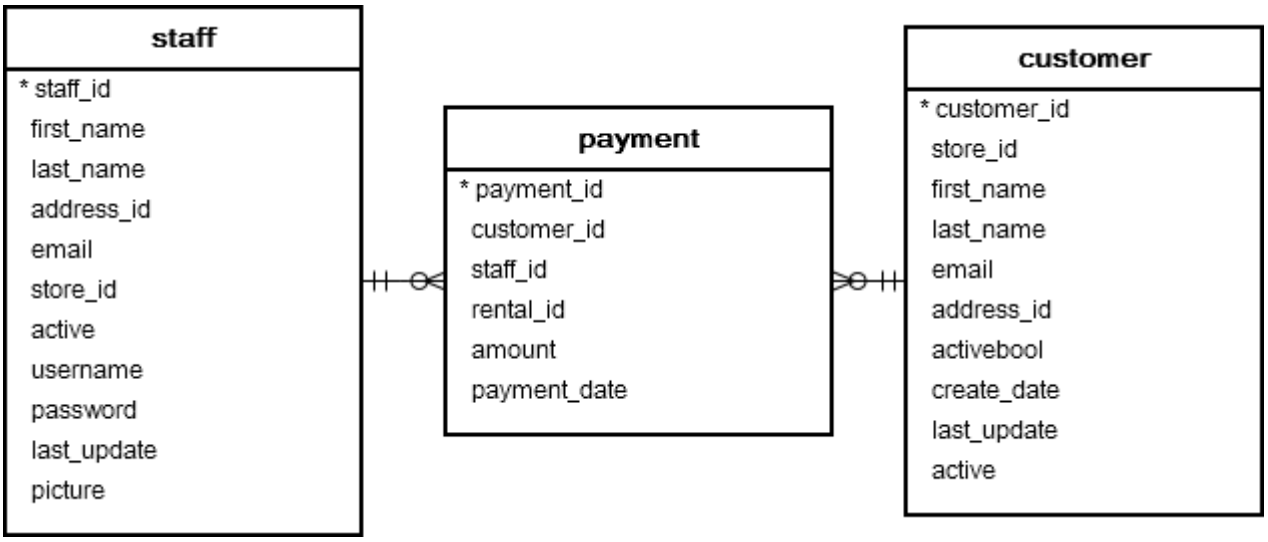
```
SELECT
	customer_id,
```

```
	first_name,
	last_name,
	amount,
	payment_date
FROM
	customer
INNER JOIN payment USING(customer_id)
ORDER BY payment_date;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

2) Usando PostgreSQL INNER JOIN para unir tres tablas

El siguiente diagrama ilustra la relación entre tres tablas: staff, paymenty customer.

- Cada personal maneja cero o muchos pagos. Y cada pago es procesado por un solo personal.
- Cada cliente hizo cero o muchos pagos. Cada pago lo realiza un cliente.



Para unir las tres tablas, coloca la segunda INNER JOINcláusula después de la primera INNER JOINcomo la siguiente consulta:

```
SELECT
	c.customer_id,
	c.first_name customer_first_name,
	c.last_name customer_last_name,
	s.first_name staff_first_name,
	s.last_name staff_last_name,
	amount,
	payment_date
FROM
	customer c
INNER JOIN payment p
    ON p.customer_id = c.customer_id
INNER JOIN staff s
    ON p.staff_id = s.staff_id
ORDER BY payment_date;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	customer_id integer	customer_first_name character varying (45)	customer_last_name character varying (45)	staff_first_name character varying (45)	staff_last_name character varying (45)	amount numeric (5,2)	payment_date timestamp without time zone
1	416	Jeffery	Pinson	Jon	Stephens	2.99	2007-02-14 21:21:59.996577
2	516	Elmer	Noe	Jon	Stephens	4.99	2007-02-14 21:23:39.996577
3	239	Minnie	Romero	Mike	Hillyer	4.99	2007-02-14 21:29:00.996577
4	592	Terrance	Roush	Jon	Stephens	6.99	2007-02-14 21:41:12.996577
5	49	Joyce	Edwards	Mike	Hillyer	0.99	2007-02-14 21:44:52.996577
6	264	Gwendolyn	May	Jon	Stephens	3.99	2007-02-14 21:44:53.996577
7	46	Catherine	Campbell	Mike	Hillyer	4.99	2007-02-14 21:45:29.996577
8	481	Herman	Devore	Jon	Stephens	2.99	2007-02-14 22:03:35.996577
9	139	Amber	Dixon	Jon	Stephens	2.99	2007-02-14 22:11:22.996577

Para unir más de tres tablas, se aplica la misma técnica.

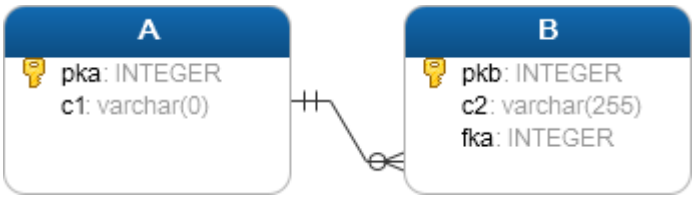
En este tutorial, ha aprendido a seleccionar datos de varias tablas mediante la INNER JOINcláusula de PostgreSQL.

UNIÓN IZQUIERDA de PostgreSQL

Resumen : en este tutorial, aprenderá a usar la LEFT JOINcláusula de PostgreSQL para seleccionar datos de varias tablas.

Introducción a la cláusula LEFT JOIN de PostgreSQL

Suponga que tiene dos tablas: A y B.



Cada fila de la tabla Apuede tener cero o varias filas correspondientes en la tabla B, mientras que cada fila de la tabla B tiene una y sólo una fila correspondiente en la tabla A .

Para seleccionar datos de la tabla A que pueden o no tener filas correspondientes en la tabla B , usa la LEFT JOINcláusula.

La siguiente declaración ilustra la LEFT JOINsintaxis que une la tabla Acon la tabla B:

```
SELECT
	pka,
	c1,
	pkb,
	c2
FROM
	A
LEFT JOIN B ON pka = fka;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Para unir la tabla A con la B tabla de la tabla usando una combinación izquierda, siga estos pasos:

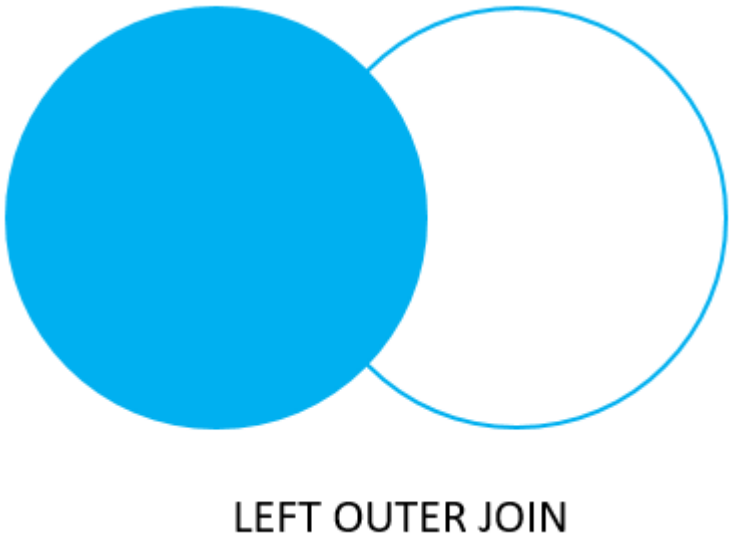
- Primero, especifique las columnas en ambas tablas desde las que desea seleccionar datos en la `SELECT`cláusula.
- En segundo lugar, especifique la tabla de la izquierda (tabla `A`) en la `FROM`cláusula.
- En tercer lugar, especifique la tabla correcta (tabla `B`) en la `LEFT JOIN`cláusula y la condición de combinación después de la `ON`palabra clave.

La `LEFT JOIN`cláusula comienza seleccionando datos de la tabla de la izquierda. Para cada fila de la tabla de la izquierda, compara el valor de la `pka`columna con el valor de cada fila de la `fka`columna de la tabla de la derecha.

Si estos valores son iguales, la cláusula de combinación izquierda crea una nueva fila que contiene las columnas que aparecen en la `SELECT`cláusula y agrega esta fila al conjunto de resultados.

En caso de que estos valores no sean iguales, la cláusula de combinación izquierda también crea una nueva fila que contiene las columnas que aparecen en la `SELECT`cláusula. Además, llena las columnas que provienen de la tabla de la derecha con NULL.

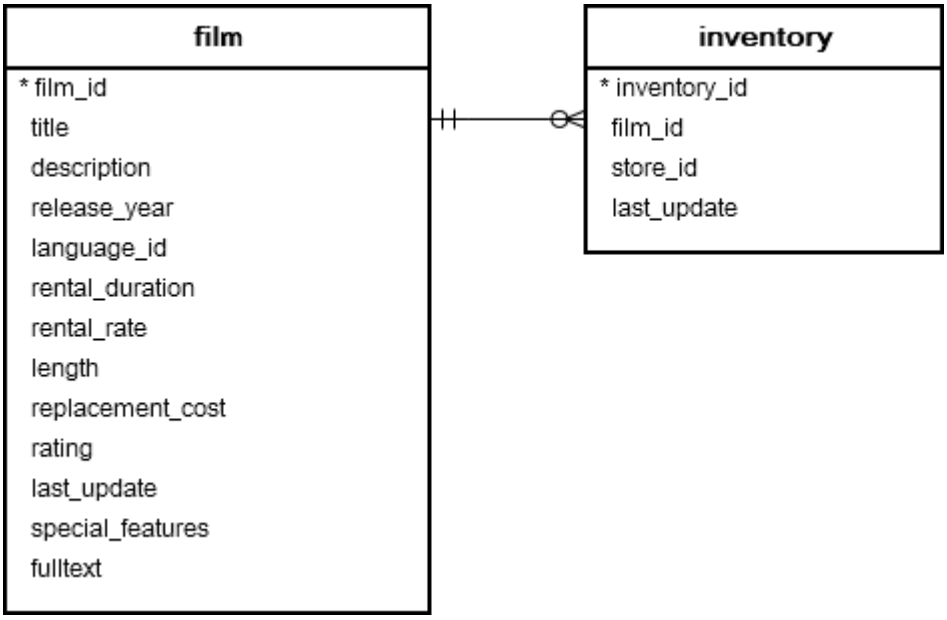
El siguiente diagrama de Venn ilustra cómo `LEFT JOIN`funciona la cláusula:



Tenga en cuenta que `LEFT JOIN` también se conoce como `LEFT OUTER JOIN`.

Ejemplos de UNIÓN IZQUIERDA de PostgreSQL

Veamos lo siguiente `film`y `inventory`las tablas de la base de datos de ejemplo .



Cada fila de la filmtabla puede tener cero o varias filas en la inventorytabla. Cada fila en la inventorytabla tiene una y solo una fila en la filmtabla.

La film_idcolumna establece el vínculo entre las tablas filmy inventory.

La siguiente declaración usa la LEFT JOINcláusula para unir filmtabla con inventorytabla:

```
SELECT
	film.film_id,
	title,
	inventory_id
FROM
	film
LEFT JOIN inventory
    ON inventory.film_id = film.film_id
ORDER BY title;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	film_id integer	title character varying (255)	inventory_id integer
65	12	Alaska Phantom	65
66	12	Alaska Phantom	66
67	13	Ali Forever	67
68	13	Ali Forever	68
69	13	Ali Forever	69
70	13	Ali Forever	70
71	14	Alice Fantasia	[null]
72	15	Alien Center	74
73	15	Alien Center	71

Cuando una fila de la filmtabla no tiene una fila coincidente en la inventorytabla, el valor de la inventory_idcolumna de esta fila es NULL.

La siguiente declaración agrega una WHERE cláusula para encontrar las películas que no están en el inventario:

```
SELECT
	film.film_id,
	film.title,
	inventory_id
FROM
	film
LEFT JOIN inventory
   ON inventory.film_id = film.film_id
WHERE inventory.film_id IS NULL
ORDER BY title;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

La siguiente declaración devuelve el mismo resultado. La diferencia es que utiliza los alias de la tabla para que la consulta sea más concisa:

```
SELECT
	f.film_id,
	title,
	inventory_id
FROM
	film f
LEFT JOIN inventory i
   ON i.film_id = f.film_id
WHERE i.film_id IS NULL
ORDER BY title;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	film_id integer	title character varying (255)	inventory_id integer
1	14	Alice Fantasia	[null]
2	33	Apollo Teen	[null]
3	36	Argonauts Town	[null]
4	38	Ark Ridgemont	[null]
5	41	Arsenic Independence	[null]
6	87	Boondock Ballroom	[null]
7	108	Butch Panther	[null]
8	128	Catch Amistad	[null]
9	144	Chinatown Gladiator	[null]
10	148	Chocolate Duck	[null]

Si ambas tablas tienen el mismo nombre de columna que se usa en la ON cláusula, puede usar la USING sintaxis de esta manera:

```
SELECT
	f.film_id,
	title,
	inventory_id
FROM
	film f
```

```
LEFT JOIN inventory i USING (film_id)
WHERE i.film_id IS NULL
ORDER BY title;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Esta técnica es útil cuando desea seleccionar filas de una tabla que no tienen filas coincidentes en otra tabla.

En este tutorial, aprendió a usar la LEFT JOINcláusula de PostgreSQL para seleccionar filas de una tabla que pueden o no tener filas correspondientes en otras tablas.

UNIÓN DERECHA de PostgreSQL

Resumen : en este tutorial, aprenderá a usar PostgreSQL RIGHT JOINpara seleccionar datos de dos tablas.

Configuración de tablas de muestra

Supongamos que tiene dos tablas filmsy film_reviewsasí:

```
DROP TABLE IF EXISTS films;
DROP TABLE IF EXISTS film_reviews;

CREATE TABLE films(
   film_id SERIAL PRIMARY KEY,
   title varchar(255) NOT NULL
);

INSERT INTO films(title)
VALUES('Joker'),
      ('Avengers: Endgame'),
      ('Parasite');

CREATE TABLE film_reviews(
   review_id SERIAL PRIMARY KEY,
   film_id INT,
   review VARCHAR(255) NOT NULL
);

INSERT INTO film_reviews(film_id, review)
VALUES(1, 'Excellent'),
      (1, 'Awesome'),
      (2, 'Cool'),
      (NULL, 'Beautiful');
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Una película puede tener cero o muchas reseñas y una reseña pertenece a cero o a una película. La film_idcolumna de filmshace referencia a la film_idcolumna de la film_reviewstabla.

A continuación se muestra el contenido de las tablas filmsy film_reviews:

```
SELECT * FROM films;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	film_id integer	title character varying (255)
1	1	Joker
2	2	Avengers: Endgame
3	3	Parasite

```
SELECT * FROM film_reviews;
```

```
Lenguaje de código:  SQL (lenguaje de consulta estructurado)  ( sql )
```

	review_id integer	film_id integer	review character varying (255)
1	1	1	Excellent
2	2	1	Awesome
3	3	2	Cool
4	4	[null]	Beautiful

La película id 1 tiene dos reseñas. La película id 2 tiene 1 reseña. La película id 3 no tiene reseñas. La reseña id 4 no se asocia con ninguna película.

Introducción a la cláusula RIGHT JOIN de PostgreSQL

La siguiente declaración utiliza RIGHT JOINpara seleccionar datos de las tablas filmsy film_reviews:

```
SELECT
    review,
    title
FROM
    films
RIGHT JOIN film_reviews
    ON film_reviews.film_id = films.film_id;
```

```
Lenguaje de código:  SQL (lenguaje de consulta estructurado)  ( sql )
```

	review character varying (255)	title character varying (255)
1	Excellent	Joker
2	Awesome	Joker
3	Cool	Avengers: Endgame
4	Beautiful	[null]

En esta declaración, filmses la tabla de la izquierda y film_reviewses la tabla de la derecha.

La RIGHT JOINcláusula comienza seleccionando datos de la tabla derecha (film_reviews).

Para cada fila de la tabla de la derecha (film_reviews), comprueba si el valor de la film_idcolumna de la film_reviewstabla es igual al valor de la film_idcolumna de cada fila de la tabla de la izquierda (films).

Si son iguales, RIGHT JOINcrea una nueva fila que contiene columnas de ambas tablas especificadas en la SELECTcláusula e incluye esta nueva fila en el conjunto de resultados.

De lo contrario, RIGHT JOINaún crea una nueva fila que contiene columnas de ambas tablas e incluye esta nueva fila en el conjunto de resultados. Sin embargo, llena las columnas de la tabla de la izquierda (films) con NULL.

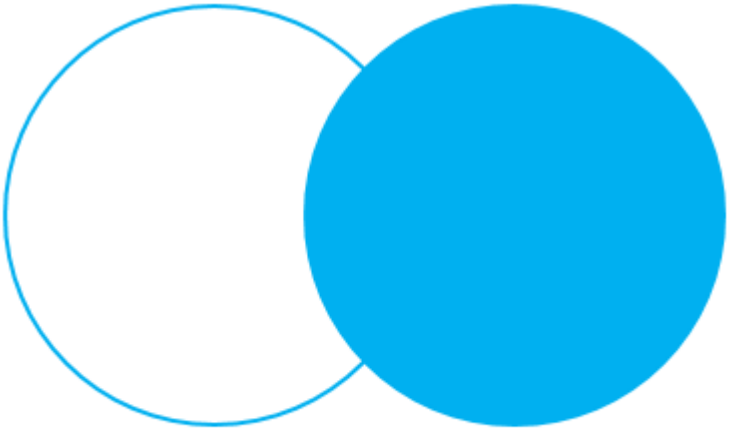
En otras palabras, RIGHT JOINselecciona todas las filas de la tabla de la derecha, tengan o no filas coincidentes de la tabla de la izquierda.

Según los datos de las tablas filmsy film_reviews:

- La reseña con id 1 coincide con la película id 1.
- La revisión con id 2 coincide con la película id 2.
- La reseña con id 3 coincide con la película id 2.
- La reseña con id 4 no coincide con ninguna película, por lo que la titlecolumna se llena con NULL.

Tenga en cuenta que RIGHT OUTER JOINes lo mismo que RIGHT JOIN. La OUTERpalabra clave es opcional

El siguiente diagrama de Venn ilustra cómo RIGHT JOINfunciona:



PostgreSQL RIGHT JOIN con sintaxis USING

Debido a que la columna unida tiene el mismo nombre (film_id), puede usar la USINGsintaxis en el predicado de unión de esta manera:

```
SELECT review, title
FROM films
RIGHT JOIN film_reviews USING (film_id);
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Esta consulta devuelve el mismo resultado que si usara la ONcláusula.

PostgreSQL RIGHT JOIN con cláusula WHERE

Para encontrar las filas de la tabla de la derecha que no tienen filas correspondientes en la tabla de la izquierda, agregue una WHERE cláusula como esta:

```
SELECT review, title
FROM films
RIGHT JOIN film_reviews using (film_id)
WHERE title IS NULL;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	review character varying (255)	title character varying (255)
1	Beautiful	[null]

En este tutorial, ha aprendido a usar la RIGHT JOIN cláusula de PostgreSQL para unir datos de dos tablas.

Self join de PostgreSQL

Resumen : en este tutorial, aprenderá a usar la técnica de autounión de PostgreSQL para comparar filas dentro de la misma tabla.

Introducción a la autounión de PostgreSQL

Una autounión es una unión normal que une una tabla consigo misma. En la práctica, normalmente usa una autocombinación para consultar datos jerárquicos o para comparar filas dentro de la misma tabla.

Para formar una autocombinación, especifica la misma tabla dos veces con diferentes alias de tabla y proporciona el predicado de combinación después de la ON palabra clave.

La siguiente consulta usa an INNER JOIN que une la tabla consigo misma:

```
SELECT select_list
FROM table_name t1
INNER JOIN table_name t2 ON join_predicate;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

En esta sintaxis, el table_name se une a sí mismo usando la INNER JOIN cláusula.

Además, puede usar la cláusula LEFT JOIN o RIGHT JOIN para unir la tabla a sí misma de esta manera:

```
SELECT select_list
FROM table_name t1
LEFT JOIN table_name t2 ON join_predicate;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

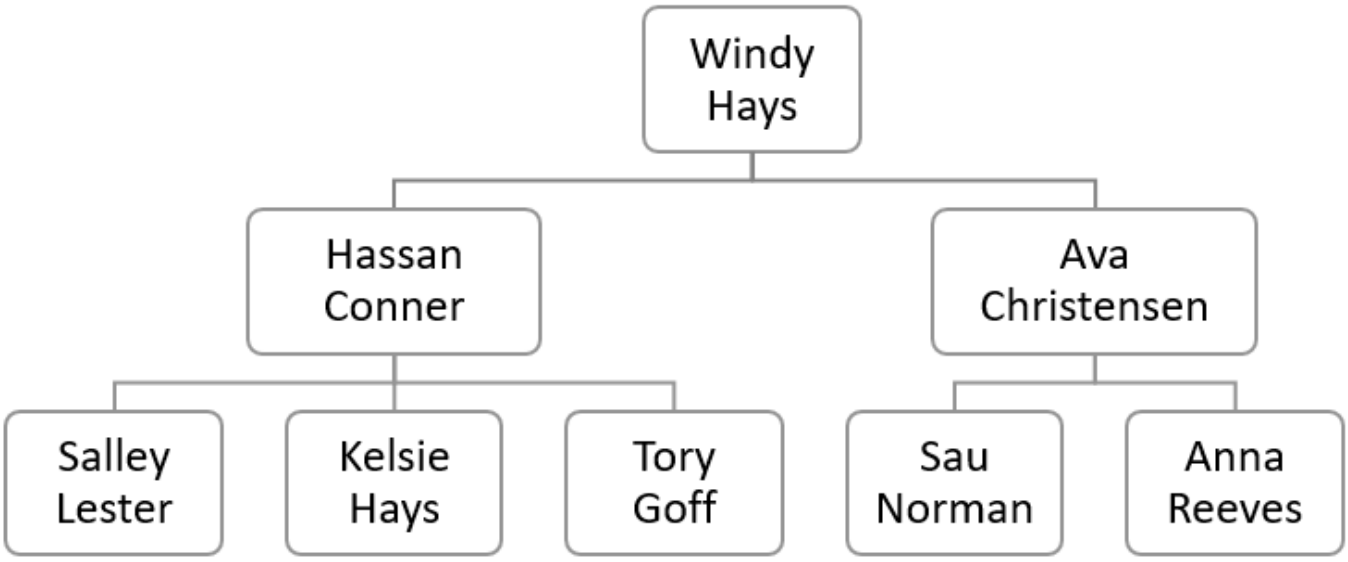
Ejemplos de autounión de PostgreSQL

Tomemos algunos ejemplos del uso de auto-uniones.

1) Ejemplo de consulta de datos jerárquicos

Preparemos una mesa de muestra para la demostración.

Supongamos que tiene la siguiente estructura organizativa:



Las siguientes declaraciones crean la employeetabla e insertan algunos datos de muestra en la tabla.

```
CREATE TABLE employee (
	employee_id INT PRIMARY KEY,
	first_name VARCHAR (255) NOT NULL,
	last_name VARCHAR (255) NOT NULL,
	manager_id INT,
	FOREIGN KEY (manager_id)
	REFERENCES employee (employee_id)
	ON DELETE CASCADE
);
INSERT INTO employee (
	employee_id,
	first_name,
	last_name,
	manager_id
)
VALUES
	(1, 'Windy', 'Hays', NULL),
	(2, 'Ava', 'Christensen', 1),
	(3, 'Hassan', 'Conner', 1),
	(4, 'Anna', 'Reeves', 2),
	(5, 'Sau', 'Norman', 2),
	(6, 'Kelsie', 'Hays', 3),
	(7, 'Tory', 'Goff', 3),
	(8, 'Salley', 'Lester', 3);
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

En esta employeetabla, la manager_idcolumna hace referencia a la employee_idcolumna. El valor en la columna manager_id muestra el gerente a quien el empleado reporta directamente. Cuando el valor de la manager_idcolumna es nulo, ese empleado no informa a nadie. En otras palabras, él o ella es el máximo responsable.

La siguiente consulta utiliza la autocombinación para encontrar quién informa a quién:

```
SELECT
    e.first_name || ' ' || e.last_name employee,
    m .first_name || ' ' || m .last_name manager
FROM
    employee e
INNER JOIN employee m ON m .employee_id = e.manager_id
ORDER BY manager;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	employee text	manager text
1	Sau Norman	Ava Christensen
2	Anna Reeves	Ava Christensen
3	Salley Lester	Hassan Conner
4	Kelsie Hays	Hassan Conner
5	Tory Goff	Hassan Conner
6	Ava Christensen	Windy Hays
7	Hassan Conner	Windy Hays

Esta consulta hace referencia a la employeestabla dos veces, una como empleado y la otra como gerente. Utiliza alias de tabla epara el empleado y mpara el gerente.

El predicado de combinación encuentra el par empleado/gerente haciendo coincidir los valores en las columnas employee_idy manager_id.

Observe que el administrador superior no aparece en la salida.

Para incluir al gerente superior en el conjunto de resultados, use la cláusula LEFT JOINen lugar de INNER JOINcomo se muestra en la siguiente consulta:

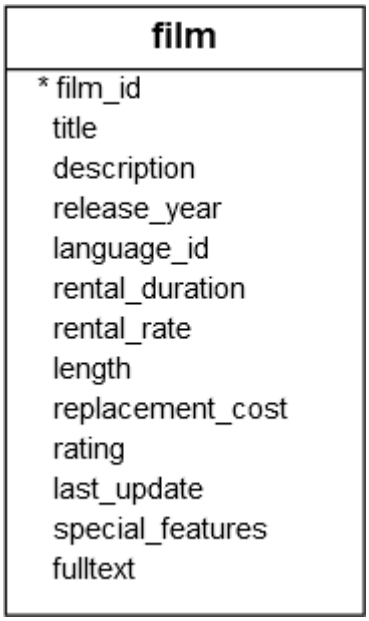
```
SELECT
    e.first_name || ' ' || e.last_name employee,
    m .first_name || ' ' || m .last_name manager
FROM
    employee e
LEFT JOIN employee m ON m .employee_id = e.manager_id
ORDER BY manager;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	employee text	manager text
1	Anna Reeves	Ava Christensen
2	Sau Norman	Ava Christensen
3	Salley Lester	Hassan Conner
4	Kelsie Hays	Hassan Conner
5	Tory Goff	Hassan Conner
6	Hassan Conner	Windy Hays
7	Ava Christensen	Windy Hays
8	Windy Hays	[null]

2) Comparando las filas con la misma tabla

Consulte la siguiente `film`tabla de la base de datos de alquiler de DVD:



La siguiente consulta encuentra todos los pares de películas que tienen la misma duración,

```
SELECT
    f1.title,
    f2.title,
    f1.length
FROM
    film f1
INNER JOIN film f2
    ON f1.film_id <> f2.film_id AND
       f1.length = f2.length;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	title character varying (255)	title character varying (255)	length smallint
1	Chamber Italian	Resurrection Silverado	117
2	Chamber Italian	Magic Mallrats	117
3	Chamber Italian	Graffiti Love	117
4	Chamber Italian	Affair Prejudice	117
5	Grosse Wonderful	Hurricane Affair	49
6	Grosse Wonderful	Hook Chariots	49
7	Grosse Wonderful	Heavenly Gun	49
8	Grosse Wonderful	Doors President	49
9	Airport Pollock	Sense Greek	54
10	Airport Pollock	October Submarine	54
11	Airport Pollock	Kill Brotherhood	54

El predicado de unión coincide con dos películas diferentes (`f1.film_id <> f2.film_id`) que tienen la misma duración (`f1.length = f2.length`)

Resumen

- Una autounión de PostgreSQL es una unión normal que une una tabla a sí misma mediante el comando INNER JOINo LEFT JOIN.
- Las autocombinaciones son muy útiles para consultar datos jerárquicos o para comparar filas dentro de la misma tabla.

Full outer Join de PostgreSQL

Resumen : en este tutorial, aprenderá a usar PostgreSQL FULL OUTER JOINpara consultar datos de dos o más tablas.

Introducción a la UNIÓN EXTERNA COMPLETA de PostgreSQL

Suponga que desea realizar una combinación externa completa de dos tablas: A y B. A continuación, se ilustra la sintaxis de FULL OUTER JOIN:

```
SELECT * FROM A
FULL [OUTER] JOIN B on A.id = B.id;
```

```
Lenguaje de código:  SQL (lenguaje de consulta estructurado)  ( sql )
```

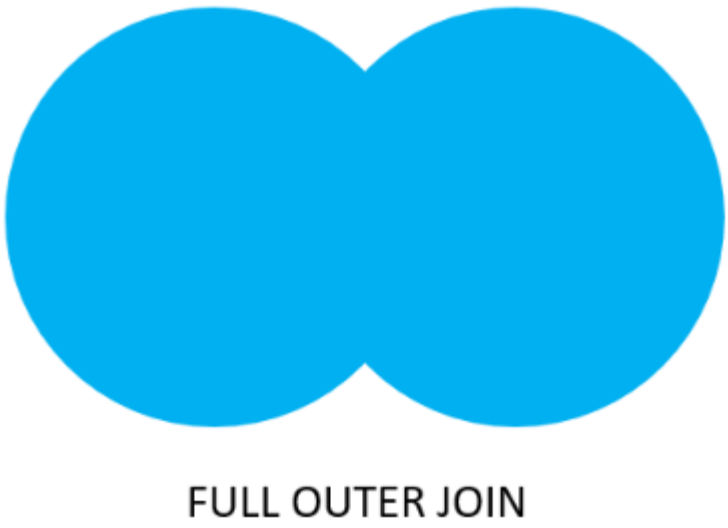
En esta sintaxis, la OUTERpalabra clave es opcional.

La combinación externa completa combina los resultados de la combinación izquierda y la combinación derecha.

Si las filas de la tabla combinada no coinciden, la combinación externa completa establece valores NULL para cada columna de la tabla que no tiene la fila coincidente.

Si una fila de una tabla coincide con una fila de otra tabla, la fila de resultados contendrá columnas completadas con columnas de filas de ambas tablas.

El siguiente diagrama de Venn ilustra la FULL OUTER JOINoperación:



El resultado incluye las filas coincidentes de ambas tablas y también las filas que no coinciden.

FULL OUTER JOINEjemplo de PostgreSQL

Primero, cree dos nuevas tablas para la demostración: employeesy departments:

```
DROP TABLE IF EXISTS departments;
DROP TABLE IF EXISTS employees;

CREATE TABLE departments (
	department_id serial PRIMARY KEY,
	department_name VARCHAR (255) NOT NULL
);

CREATE TABLE employees (
	employee_id serial PRIMARY KEY,
	employee_name VARCHAR (255),
	department_id INTEGER
);
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Cada departamento tiene cero o muchos empleados y cada empleado pertenece a cero o un departamento.

En segundo lugar, inserte algunos datos de muestra en las tablas departmentsy employees.

```
INSERT INTO departments (department_name)
VALUES
	('Sales'),
	('Marketing'),
	('HR'),
	('IT'),
	('Production');

INSERT INTO employees (
	employee_name,
	department_id
)
VALUES
	('Bette Nicholson', 1),
	('Christian Gable', 1),
	('Joe Swank', 2),
	('Fred Costner', 3),
	('Sandra Kilmer', 4),
	('Julia Mcqueen', NULL);
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

En tercer lugar, consulte los datos de las tablas departmentsy employees:

```
SELECT * FROM departments;
```

	department_id integer	department_name character varying (255)
1	1	Sales
2	2	Marketing
3	3	HR
4	4	IT
5	5	Production

```
SELECT * FROM employees;
```

	employee_id integer	employee_name character varying (255)	department_id integer
1	1	Bette Nicholson	1
2	2	Christian Gable	1
3	3	Joe Swank	2
4	4	Fred Costner	3
5	5	Sandra Kilmer	4
6	6	Julia Mcqueen	[null]

En cuarto lugar, use FULL OUTER JOINpara consultar datos de tablas employeesy departments.

```
SELECT
	employee_name,
	department_name
FROM
	employees e
FULL OUTER JOIN departments d
	ON d.department_id = e.department_id;
```

```
Lenguaje de código:  SQL (lenguaje de consulta estructurado)  ( sql )
```

	employee_name character varying (255)	department_name character varying (255)
1	Bette Nicholson	Sales
2	Christian Gable	Sales
3	Joe Swank	Marketing
4	Fred Costner	HR
5	Sandra Kilmer	IT
6	Julia Mcqueen	[null]
7	[null]	Production

El conjunto de resultados incluye todos los empleados que pertenecen a un departamento y todos los departamentos que tienen un empleado. Además, incluye todo empleado que no pertenezca a un departamento y todo departamento que no tenga empleado.

Para encontrar el departamento que no tiene empleados, utilice una cláusula WHERE de la siguiente manera:

```
SELECT
	employee_name,
	department_name
FROM
```

```
        employees e
FULL OUTER JOIN departments d
          ON d.department_id = e.department_id
WHERE
        employee_name IS NULL;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	employee_name character varying (255)	department_name character varying (255)
1	[null]	Production

El resultado muestra que el Productiondepartamento no tiene empleados.

Para encontrar un empleado que no pertenezca a ningún departamento, verifique la NULLde department_nameen la WHEREcláusula como la siguiente declaración:

```
SELECT
        employee_name,
        department_name
FROM
        employees e
FULL OUTER JOIN departments d ON d.department_id = e.department_id
WHERE
        department_name IS NULL;
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	employee_name character varying (255)	department_name character varying (255)
1	Julia Mcqueen	[null]

Como se ve claramente en la salida, Juila Mcqueenno pertenece a ningún departamento.

En este tutorial, ha aprendido a usar la FULL OUTER JOINcláusula de PostgreSQL para unir dos o más tablas.

Unión cruzada de PostgreSQL por ejemplo

Resumen : en este tutorial, aprenderá cómo usar PostgreSQL CROSS JOIN para producir un producto cartesiano de filas de las tablas unidas.

Introducción a la cláusula CROSS JOIN de PostgreSQL

Una CROSS JOINcláusula le permite producir un producto cartesiano de filas en dos o más tablas.

A diferencia de otras cláusulas de combinación , como LEFT JOIN o INNER JOIN , la CROSS JOIN cláusula no tiene un predicado de combinación.

Supongamos que tiene que realizar una CROSS JOINde dos tablas T1 y T2.

Si T1 tiene nfilas y T2 tiene mfilas, el conjunto de resultados tendrá nxmfilas. Por ejemplo, T1 tiene 1,000filas y T2 tiene 1,000filas, el conjunto de resultados tendrá 1,000 x 1,000= 1,000,000filas.

A continuación se ilustra la sintaxis de la CROSS JOINsintaxis:

```
SELECT select_list
FROM T1
CROSS JOIN T2;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

La siguiente declaración es equivalente a la declaración anterior:

```
SELECT select_list
FROM T1, T2;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Además, puede usar una INNER JOINcláusula con una condición que siempre se evalúe como verdadera para simular la unión cruzada:

```
SELECT *
FROM T1
INNER JOIN T2 ON true;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Ejemplo de UNIÓN CRUZADA de PostgreSQL

Las siguientes declaraciones CREATE TABLE crean tablas T1 y T2 e insertan algunos datos de muestra para la demostración cruzada.

```
DROP TABLE IF EXISTS T1;
CREATE TABLE T1 (label CHAR(1) PRIMARY KEY);

DROP TABLE IF EXISTS T2;
CREATE TABLE T2 (score INT PRIMARY KEY);

INSERT INTO T1 (label)
VALUES
	('A'),
	('B');

INSERT INTO T2 (score)
VALUES
	(1),
	(2),
	(3);
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

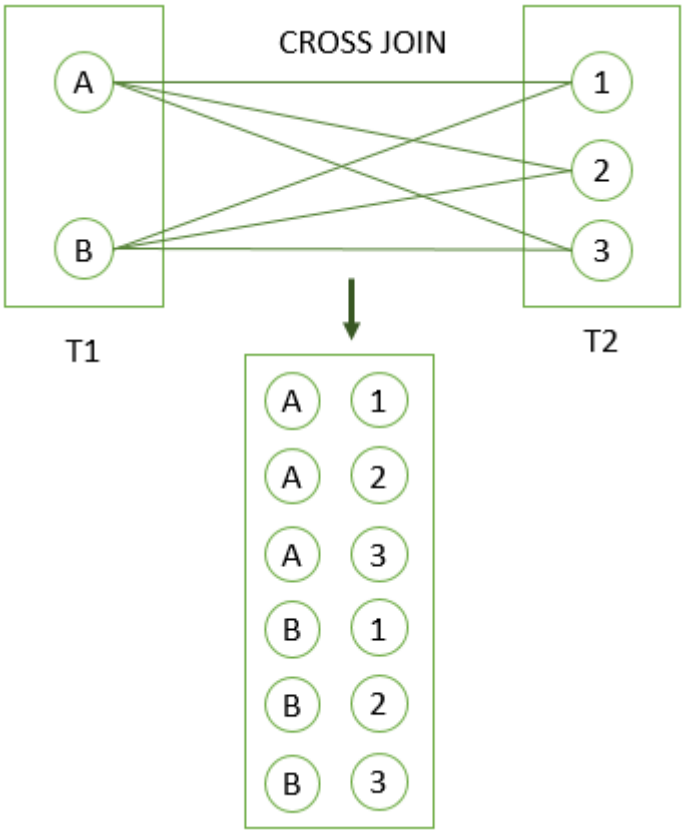
La siguiente declaración usa el CROSS JOINoperador para unir la tabla T1 con la tabla T2.

```
SELECT *
FROM T1
CROSS JOIN T2;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

```
 label | score
-------+-------
 A     |     1
 B     |     1
 A     |     2
 B     |     2
 A     |     3
```

```
 B      |      3
(6 rows)
```

La siguiente imagen ilustra el resultado CROSS JOINal unir la mesa T1 a la mesa T2:



En este tutorial, ha aprendido a utilizar la cláusula CROSS JOIN de PostgreSQL para crear un producto cartesiano de filas en dos o más tablas.

PostgreSQL NATURAL JOIN explicado con ejemplos

Resumen : en este tutorial, aprenderá a usar PostgreSQL NATURAL JOINpara consultar datos de dos o más tablas.

Una combinación natural es una combinación que crea una combinación implícita basada en los mismos nombres de columna en las tablas combinadas.

A continuación se muestra la sintaxis de la unión natural de PostgreSQL:

```
SELECT select_list
FROM T1
NATURAL [INNER, LEFT, RIGHT] JOIN T2;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Una combinación natural puede ser una combinación interna , una combinación izquierda o una combinación derecha. Si no especifica una unión explícitamente, por ejemplo, INNER JOIN, LEFT JOIN, RIGHT JOIN, PostgreSQL usará INNER JOINde forma predeterminada.

Si usa el asterisco (*) en la lista de selección, el resultado contendrá las siguientes columnas:

- Todas las columnas comunes, que son las columnas de ambas tablas que tienen el mismo nombre.
- Cada columna de ambas tablas, que no es una columna común.

Ejemplos de UNIÓN NATURAL de PostgreSQL

Para demostrar la combinación natural de PostgreSQL, crearemos dos tablas: categoriesy products.

Las siguientes CREATE TABLEsentencias crean las tablas categoriesy products.

```
DROP TABLE IF EXISTS categories;
CREATE TABLE categories (
	category_id serial PRIMARY KEY,
	category_name VARCHAR (255) NOT NULL
);

DROP TABLE IF EXISTS products;
CREATE TABLE products (
	product_id serial PRIMARY KEY,
	product_name VARCHAR (255) NOT NULL,
	category_id INT NOT NULL,
	FOREIGN KEY (category_id) REFERENCES categories (category_id)
);
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

Cada categoría tiene cero o muchos productos y cada producto pertenece a una y solo una categoría.

La category_idcolumna de la productstabla es la clave externa que hace referencia a la clave principal de la categoriestabla. es category_idla columna común que usaremos para realizar la unión natural.

Las siguientes instrucciones INSERT insertan algunos datos en las tablas categoriesy products.

```
INSERT INTO categories (category_name)
VALUES
	('Smart Phone'),
	('Laptop'),
	('Tablet');

INSERT INTO products (product_name, category_id)
VALUES
	('iPhone', 1),
	('Samsung Galaxy', 1),
	('HP Elite', 2),
	('Lenovo Thinkpad', 2),
	('iPad', 3),
	('Kindle Fire', 3);
```

Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

La siguiente declaración usa la NATURAL JOINcláusula para unir la productstabla con la categoriestabla:

```
SELECT * FROM products
NATURAL JOIN categories;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

	category_id integer	product_id integer	product_name character varying (255)	category_name character varying (255)
1	1	1	iPhone	Smart Phone
2	1	2	Samsung Galaxy	Smart Phone
3	2	3	HP Elite	Laptop
4	2	4	Lenovo Thinkpad	Laptop
5	3	5	iPad	Tablet
6	3	6	Kindle Fire	Tablet

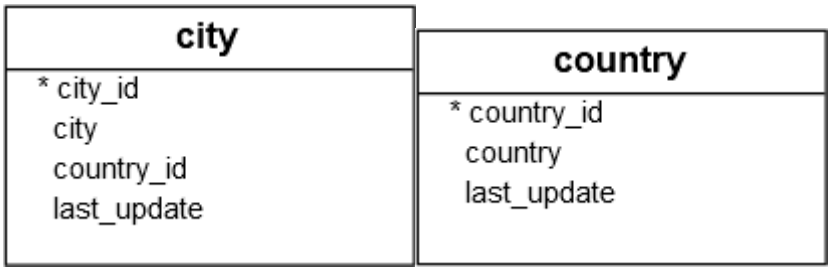
La declaración anterior es equivalente a la siguiente declaración que usa la INNER JOINcláusula.

```
SELECT * FROM products
INNER JOIN categories USING (category id);
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

La conveniencia de es NATURAL JOINque no requiere que especifique la cláusula de unión porque usa una cláusula de unión implícita basada en la columna común.

Sin embargo, debe evitar usar el NATURAL JOIN siempre que sea posible porque a veces puede causar un resultado inesperado.

Por ejemplo, consulte lo siguiente cityy countrylas tablas de la base de datos de ejemplo :



Ambas tablas tienen la misma country_idcolumna, por lo que puede usar NATURAL JOINpara unir estas tablas de la siguiente manera:

```
SELECT *
FROM city
NATURAL JOIN country;
```
Lenguaje de código: SQL (lenguaje de consulta estructurado) (sql)

La consulta devuelve un conjunto de resultados vacío.

La razón es que...

Ambas tablas también tienen otra columna común llamada last_update, que no se puede usar para la combinación. Sin embargo, la NATURAL JOINcláusula solo usa la last_updatecolumna.

En este tutorial, aprendió sobre el NATURAL JOINfuncionamiento de PostgreSQL y cómo usarlo para consultar datos de dos o más tablas que tienen las columnas comunes.