Postgres
Guia de atajos

{dev/talles}

Tipos de datos generales

Tipo	Alias	Descripción
`bigint`	`int8`	Entero de ocho bytes con signo
`bit [ (n) ]`		Cadena de bits de longitud fija
`bit varying [(n)]`	`varbit [ (n) ]`	Cadena de bits de longitud variable
`boolean`	`bool`	Booleano lógico (verdadero/falso)
`character varying [(n)]`	`varchar[(n)]`	Cadena de caracteres de longitud variable
`date`		Fecha del calendario (año, mes, día)
`integer`	`int, int4`	Entero de cuatro bytes con signo
`json`		Datos JSON textuales
`money`		Cantidad de moneda
`numeric [(p, s)]`	`decimal[(p,s)]`	Numérico exacto de precisión seleccionable
`smallint`	`int2`	Entero de dos bytes con signo
`smallserial`	`serial2`	Entero de dos bytes autoincrementable
`serial`	`serial4`	Entero de cuatro bytes autoincrementable
`text`		Cadena de caracteres de longitud variable
`time [ (p) ] [ without time zone ]`		Hora del día (sin zona horaria)
`time [ (p) ] with time zone`	`timetz`	Hora del día, incluida la zona horaria
`timestamp [ (p) ] [ without time zone ]`		Fecha y hora (sin zona horaria)
`timestamp [ (p) ] with time zone`	`timestamptz`	Fecha y hora, incluida la zona horaria
`uuid`		Identificador único universal
`xml`		Datos XML

Operadores de Strings y funciones

Operador Función	Descripción
||	Concatena dos o más strings
CONCAT( )	Une dos o más strings
LOWER( )	Resultado en minúscula
UPPER( )	Resultado en mayúscula
LENGTH( )	Número de caracteres del string
POSITION ('term' in field)	Buscar 'term' en el campo y retorna el indice
TRIM(text)	Remueve los espacios iniciales y finales del string (ltrim, rtrim)

Operadores matemáticos y funciones

Operador Función	Descripción
+	Sumar
-	Restar
*	Multiplicar
/	Dividir (divisiones entre enteros cortan el resultado)
%	Resultado de la división
ROUND(v, p)	Redondea el **v**alor y **p**recisión decimal

Operadores de Comparación

Operador Función	Descripción
=	¿Son los valores iguales?
>	¿Es el valor de la izquierda es más grande que el de la derecha?
<	¿Es el valor de la izquierda más grande que el de la derecha?
>=	¿Es el valor de la derecha mayor o igual al de la derecha?
<=	¿Es el valor de la izquierda mayor o igual al de la derecha?
IN	¿El valor se encuentra en la lista?
NOT IN	El valor no se encuentra en la lista

Postgres
Guia de atajos

{dev/talles}

<>	¿Los valores no son iguales?
!=	¿Los valores no son iguales?
BETWEEN	El valor se encuentra entre estos dos valores.
NOT BETWEEN	El valor no se encuentra entre estos dos valores.
IS NULL	Realiza la verificación si el resultado o campo es nulo
LIKE	El término de búsqueda contiene un patrón especifico
NOT LIKE	El término de búsqueda NO contiene un patrón especifico

Operadores lógicos

Operador	Descripción
AND	Ambas condiciones se deben de cumplir
OR	Una de las condiciones se tiene que cumplir
NOT	Depende de donde se use, pero en general es una negación.

Tabla lógica

a	b	a AND b	a OR b
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	NULL	NULL	TRUE
FALSE	FALSE	FALSE	FALSE
FALSE	NULL	FALSE	NULL
NULL	NULL	NULL	NULL

Tabla lógica de NOT

a	NOT a
TRUE	FALSE
FALSE	TRUE
NULL	NULL

Primary Keys automáticas

Tipo	Descripción
SERIAL	Serie valores numéricos correlativos
SEQUENCE	Secuencia personalizada única
gen_random_uuid()	UUID en formato V4

Ejemplo de SERIAL primary key

```
CREATE TABLE books (
  id              SERIAL PRIMARY KEY,
  title           VARCHAR(100) NOT NULL,
  primary_author  VARCHAR(100) NULL
);
```

Ejemplo de secuencia

```
CREATE SEQUENCE books_sequence
  start 2
  increment 2;

INSERT INTO books
  (id, title)
VALUES
 ( nextval('books_sequence'), 'TheHobbit' );
```

En definición de tabla

```
CREATE TABLE Students (
   id bigint DEFAULT nextval('integer_id_seq')
PRIMARY KEY,
   name VARCHAR(200) not NULL
);

CREATE TABLE Students (
   id uuid DEFAULT gen_random_uuid() PRIMARY KEY,
   name VARCHAR(200) not NULL
);

CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
DROP EXTENSION "uuid-ossp";
```

Ejemplos de cláusulas

Between

```
SELECT * FROM "users"
WHERE "id" BETWEEN 2 AND 4;
```

Case

```
SELECT "name", "salary",
CASE
  WHEN "salary" > 6000 THEN 'Salario mayor a 6000$'
  WHEN "salary" > 4000 THEN 'Salario mayor a 4000$'
  WHEN "salary" >= 2000 THEN 'Salario mayor a 2000$'
  ELSE 'Salario menor a 2000$'
END AS "Salary information"
FROM "users";
```

Comentarios

```
-- Single-line comment

/*
  Multi-line
  comment
*/
```

Postgres
Guia de atajos

{dev/talles}

Comandos comunes

Crear base de datos

```
CREATE DATABASE “database_name";

CREATE DATABASE IF NOT EXISTS "productsDB";
```

Crear tabla

```
CREATE TABLE "users" (
    id SERIAL,
    name VARCHAR(100) NOT NULL,
    role VARCHAR(15) NOT NULL,
    PRIMARY KEY (id)
);
```

Crear vista y destruir vista

```
CREATE OR REPLACE [MATERIALIZED] VIEW
"v_spain_users" AS
  SELECT "name", "email" FROM "users"
  WHERE "country" = 'Spain';

DROP [MATERIALIZED]  VIEW "v_spawn_users";
```

Eliminar registros

```
DELETE FROM "table_name"
WHERE "column_name" = some_value;
```

Inserciones

```
INSERT INTO "table_name"
    ("column1", "column2", "column3", ...)
VALUES
    ("value1", "value2", "value3", ...);

-- Múltiples

INSERT INTO "table_name"
    ("column1", "column2", "column3", …)
VALUES
    ("value1", "value2", "value3", ...),
    ("value1", "value2", "value3", ...),
    ("value1", "value2", "value3", ...),
    ...;
```

Actualizar registros

```
UPDATE "users"
SET
  "name" = 'Christopher' ,
  "role" = 'admin'
WHERE "id" = 2;
```

Group by y Count

```
SELECT COUNT(name) AS user_count, "country"
FROM "users2"
GROUP BY "country"
ORDER BY user_count DESC;
```

Like Statements

```
SELECT * FROM “users"

-- Nombre inicie con J mayúscula
WHERE  "name"  LIKE 'J%';

-- Nombre inicie con Jo
WHERE  "name"  LIKE 'Jo%';

-- Nombre termine con hn
WHERE  "name"  LIKE '%hn';

-- Nombre tenga 3 letras y las últimas
2
-- tienen que ser "om"
WHERE  "name"  LIKE '_om'; // Tom

-- Puede iniciar con cualquier letra
-- seguido de "om" y cualquier cosa
después
WHERE  "name"  LIKE '_om%'; // Tomas
```

Limit y Offset

```
SELECT * FROM "users"
  LIMIT 5
  OFFSET 10;
```

Order by

```
SELECT "column1", "column2", ...
FROM "table_name"
ORDER BY "column1", ... ASC|DESC;
```

Select Distinct

```
SELECT DISTINCT "country"
FROM "users";
```

Contar registros en tabla y Having

```
select count(*) from photos

select count(*), “column"
from "table"
GROUP by "column"
HAVING count(*) > 3
```

Postgres
Guia de atajos

{dev/talles}

Funciones de agregación

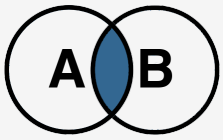
Function	Description
`array_agg(`***expression***`)`	Valores de entrada, incluidos los nulos, concatenados en una matriz
`avg(`***expression***`)`	El promedio (media aritmética) de todos los valores de entrada no nulos
`count(*)`	Número de filas de entrada
`count(`***expression***`)`	Número de filas de entrada para las que el valor de expresión no es nulo
`json_agg(`***expression***`)`	Agrega valores, incluidos valores nulos, como una matriz JSON
`max(`***expression***`)`	Valor máximo de expresión en todos los valores de entrada no nulos
`min(`***expression***`)`	Valor mínimo de expresión en todos los valores de entrada no nulos
`string_agg(`***expression, delimiter***`)`	Valores de entrada no nulos concatenados en una cadena, separados por delimitador
`sum(`***expression***`)`	Suma de expresión en todos los valores de entrada no nulos

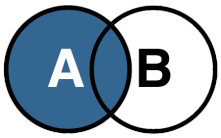
Funciones condicionales

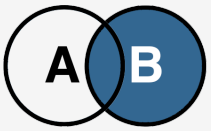
Function	Description
coalesce(ANY REPEATED)	Devuelve el primero de sus argumentos que no es nulo
greatest(ANY REPEATED)	Devuelve el valor más grande de una lista de cualquier número de expresiones
least(ANY REPEATED)	Devuelve el valor más pequeño de una lista de cualquier número de expresiones.
nullif(*value1* ANY, *value2* ANY)	Devuelve un valor nulo si valor1 es igual a valor2; de lo contrario, devuelve valor1.

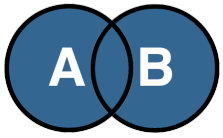
JOINS - Uniones de tablas

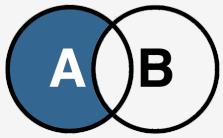
Visualisation	SQL Query

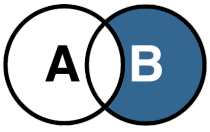
INNER JOIN 	`SELECT * FROM table_a A`
	`  JOIN table_b B`
	`  ON A.key = B.key`

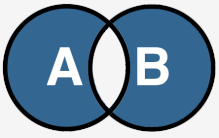
LEFT OUTER JOIN 	`SELECT * FROM table_a A`
	`LEFT JOIN table_b B`
	`ON A.key = B.key`

RIGHT OUTER JOIN 	`SELECT * FROM table_a A`
	`RIGHT JOIN table_b B`
	`ON A.key = B.key`

FULL OUTER JOIN 	`SELECT * FROM table_a A`
	`FULL OUTER JOIN table_b B`
	`ON A.key = B.key`

LEFT OUTER JOIN con exclusión 	`SELECT *`
	`FROM table_a A`
	`LEFT JOIN table_b B ON`
	`A.key = B.key`
	`WHERE B.key IS NULL`

RIGHT OUTER JOIN con exclusión 	`SELECT *`
	`FROM table_a A`
	`RIGHT JOIN table_b B`
	`ON A.key = B.key`
	`WHERE A.key IS NULL`

FULL OUTER JOIN con exclusión 	`SELECT * FROM table_a A`
	`FULL OUTER JOIN table_b B`
	`ON A.key = B.key`
	`WHERE B.key IS NULL OR`
	`A.key IS NULL`

Postgres
Guia de atajos

{dev/talles}

PLPGSQL + CTE

Funciones personalizadas

Sintaxis general:

```
CREATE [OR REPLACE] FUNCTION
function_name([arguments type])
RETURNS return_datatype as $$
DECLARE
  < Declaración de variables >
BEGIN
  < Cuerpo de la función >
  RETURN < Valor >
END; LANGUAGE plpgsql;
```

Ejemplo:

```
CREATE OR REPLACE FUNCTION
greet_employee( emp_name varchar )

RETURNS varchar
    AS $$

BEGIN
  RETURN 'Hola ' || emp_name;
END;

$$
LANGUAGE plpgsql;


select greet_employee('Fernando');
```

Common Table Expression (CTE)

```
with cte_name as (
  select <campos> from <tabla>....
), [cte_name_2] as ()...
select * from cte_name;
```

Recursivo

```
-- Nombre de la tabla en memoria
-- campos que vamos a tener
WITH RECURSIVE countdown( val ) as (
  -- initialización => el primer nivel,
o valores iniciales
  -- values(5)
  select 10 as val
  UNION
  -- Query recursivo
  select val - 1 from countdown where
val > 1
)
-- Select de los campos
select * from countdown;
```

Procedimientos almacenados

```
CREATE OR REPLACE PROCEDURE
proc_name ( [args type] ) AS
$$
DECLARE
  -- variables
BEGIN
  -- cuerpo
END;
$$ LANGUAGE plpgsql;


call proc_name( 'valores' );
```

Triggers

```
create or REPLACE TRIGGER <name>
AFTER UPDATE on "user"
FOR EACH ROW

-Opcional when
WHEN (OLD.field IS DISTINCT FROM NEW.field)

- Procedimiento/Función a ejecutar
EXECUTE FUNCTION create_session_log();


create or REPLACE FUNCTION <name>()
RETURNS TRIGGER as $$

BEGIN
- Cuerpo de la función

  return NEW;
END;
$$ LANGUAGE plpgsql;
```

PgCrypto

```
CREATE EXTENSION pgcrypto;

insert into "user" (username, password)
values(
  'melissa',
  crypt( '123456', gen_salt('bf') )
);

select count(*) from "user"
  where username='fernando' and
  password = crypt('123456', password);
```