



# POSTGRESQL

Por: [enidev911](#)



# TIPOS DE DATOS



# ENTEROS

NOMBRE	TAMAÑO	DESDE	HASTA
SMALLINT o INT2	2 bytes	-32768	32767
INTEGER o INT4	4 bytes	-2147483648	2147483647
BIGINT o INT8	8 bytes	-9223372036854775808	9223372036854775807

# AUTO-INCREMENTABLES

NOMBRE	TAMAÑO	DESDE	HASTA
SMALLSERIAL	2 bytes	1	32767
SERIAL	4 bytes	1	2147483647
BIGSERIAL	8 bytes	1	9223372036854775807



# AUTO-INCREMENTABLES

Los tipos de datos `smallserial`, `serial`, `bigserial` no son tipos verdaderos, sino simplemente una forma a conveniencia para crear columnas con identificador único.

```
CREATE TABLE nombretabla (  
    columna1 SERIAL  
);
```

```
CREATE SEQUENCE tabla1_columna1_seq  
AS INTEGER;  
  
CREATE TABLE tabla1 (  
    columna1 INTEGER DEFAULT  
    nextval('tabla1_columna1_seq')  
);
```



# CARACTERES

NOMBRE	DESCRIPCIÓN
CHARACTER VARYING(n) o VARCHAR(n)	Longitud variable con límite
CHARACTER(n) o CHAR(n)	Longitud fija
TEXT	Longitud ilimitada variable

# CADENAS BINARIAS

NOMBRE	TAMAÑO	DESCRIPCIÓN
BYTEA	1 a 4 bytes más la cadena binaria real	Cadena binaria de longitud variable



# OPERADORES

# COMPARACIÓN

OPERADOR	DESCRIPCIÓN	USO
<	MENOR QUE	$A < B$
>	MAYOR QUE	$A > B$
<=	MENOR O IGUAL A QUE	$A <= B$
>=	MAYOR O IGUAL A QUE	$A >= B$
=	IGUAL A QUE	$A = B$
<> o !=	NO ES IGUAL	$A <> B$

# LÓGICOS

OPERADOR	DESCRIPCIÓN	USO	RESULTADO
AND	Espera a que se cumplan todas las condiciones	$5 = 5 \text{ AND } 3 > 2$	T => TRUE
OR	Espera a que se cumplan al menos una de todas las condiciones	$5 = 2 \text{ OR } 3 > 2$	T => TRUE
NOT	Espera a que no se cumpla ninguna condición	NOT $4=6$	T => TRUE

# MATEMÁTICOS

OPERADOR	DESCRIPCIÓN	USO	RESULTADO
+	ADICIÓN	4 + 4	8
-	SUSTRACCIÓN	2 - 3	-1
*	MULTIPLICACIÓN	2 * 3	6
/	DIVISIÓN	4 / 2	2
%	MÓDULO (RESTO)	5 % 4	1
^	EXPONENCIACIÓN	2 ^ 4	16
	RAÍZ CUADRADA	/ 25	5

# CRITERIOS

OPERADOR	DESCRIPCIÓN	USO	RESULTADO
BETWEEN	Intervalos de valores	BETWEEN 20 AND 30	Todos los registros dentro del intervalo {20,21,...30}
IN	Conjunto de valores en una lista	IN (44, 55, 66)	Todos los registros que estén dentro del conjunto





# FUNCIONES

# FUNCIONES MATEMÁTICAS

Además de los operadores matemáticos tenemos algunas funciones que podemos emplear directamente en el servidor:

FUNCIÓN	DESCRIPCIÓN	USO	RESULTADO
<code>abs(num)</code>	Retorna el valor absoluto	<code>abs(-17)</code>	17
<code>cbrt(pd)</code>	Retorna la raíz cúbica	<code>cubrt(27)</code>	3
<code>ceil(pd   num)</code>	Retorna el valor entero mayor más cercano	<code>ceil(28.2)</code>	29
<code>random()</code>	Retorna un número aleatorio entre 0.0 y 1.0	<code>random()</code>	0.129633..
<code>power(a dp, b dp)</code>	Retorna A elevado a la potencia de B	<code>power(5,4)</code>	625
<code>mod(a dp, b dp)</code>	Retorna el módulo (resto) de la operación	<code>mod(9,4)</code>	1
<code>factorial(num)</code>	Retorna el factorial del argumento	<code>factorial(5)</code>	120
<code>trunc(v num, s int)</code>	Retorna el valor truncado a S decimales	<code>trunc(42.2382, 2)</code>	42.23

num: valor numérico

pd: valor de precisión doble

# FUNCIONES DE CADENAS (string)

Las funciones y operadores de la siguiente lista nos permiten examinar y manipular valores de cadenas:

FUNCIÓN	DESCRIPCIÓN	USO	RESULTADO
<code>concat(str, [...])</code>	Concatenar todos los argumentos. Los argumentos NULL se ignoran	<code>concat('abc', 'de', NULL, 12)</code>	abcde12
<code>substring(str from pattern)</code>	Extraer subcadena.	<code>substring('marco' from 2 for 3)</code>	arc
<code>position(substr in str)</code>	Retorna un número entero de la posición de substr	<code>position('om' in 'thomas')</code>	3
<code>char_length(str)</code> o <code>character_length(str)</code>	Longitud de la cadena (str)	<code>char_length('jose')</code>	4
<code>lower(str)</code>	Convierte str a minúscula	<code>lower('TOM')</code>	tom
<code>upper(str)</code>	Convierte str a mayúscula	<code>upper('tom')</code>	TOM
<code>reverse(str)</code>	Retorna la cadena inversa	<code>reverse('hello')</code>	olleh

# FUNCIONES DE CADENAS (string)

Las funciones y operadores de la siguiente lista nos permiten examinar y manipular valores de cadenas:

FUNCIÓN	DESCRIPCIÓN	USO	RESULTADO
<code>right(str text, n int)</code>	Devuelve los últimos N caracteres de str	<code>right('abcde', 2)</code>	de
<code>left(str text, n int)</code>	Devuelve los primeros N caracteres de la cadena	<code>left('abcde', 2)</code>	ab
<code>ltrim(str text [, characters text])</code>	Recorta 'characters' de izquierda a derecha	<code>ltrim('xzzytimx', 'xyz')</code>	timx
<code>rtrim(str text [, characters text])</code>	Recorta 'characters' de derecha a izquierda	<code>rtrim('timxxzyz', 'xyz')</code>	tim
<code>trim([leading   trailing   both] [characters] from str)</code>	Convierte str a mayúscula	<code>trim(both 'x' from 'xTomxx')</code>	Tom
<code>replace(str, from str, to str)</code>	Reemplaza el str de from al str de to	<code>concat('abc', 'de', NULL, 12)</code>	abcde12



# EJERCICIO

# CONTEXTO

El centro de mascotas pet-health tiene interés de mantener una base de datos en un servidor de Postgres para almacenar toda la información relacionada con las mascotas del establecimiento. Para ello necesitan crear una tabla que cumpla una serie de requerimientos en base a un diseño propuesto por el dueño del establecimiento.



# DISEÑO DE LA TABLA

## Tabla – Mascotas

FIELD	TYPE	NULL	PRIMARY KEY
ID_MASCOTA	INT4	NO	PRI
NOMBRE	VARCHAR(30)	NO	
ESPECIE	CHAR(1)	NO	
SEXO	CHAR(1)	NO	
UBICACION	VARCHAR(6)	NO	
ESTADO	CHAR(1)	NO	

## Descripción de los campos

- **ID\_MASCOTA**: Identificador de la mascota.
- **NOMBRE**: Nombre de la mascota.
- **ESPECIE**: Campo codificado donde se guarda 'P' para perro y 'G' para gato.
- **SEXO**: Campo codificado donde se guarda 'M' para macho y 'H' para hembra.
- **UBICACION**: Jaula o estancia donde está ubicada la mascota.
- **ESTADO**: Campo codificado donde se guarda 'A' para alta en el centro y 'B' para baja en el centro.

```
CREATE TABLE mascotas (  
  id_mascota INT2,  
  nombre VARCHAR(40) NOT NULL,  
  especie CHAR(1) NOT NULL,  
  sexo CHAR(1),  
  ubicacion VARCHAR(3),  
  estado CHAR(1),  
  PRIMARY KEY(id_mascota)  
);
```



Clic para ir al  
script.

```
INSERT INTO mascotas  
(id_mascota, nombre, especie, sexo,  
ubicacion, estado)  
VALUES  
(1001, 'Budy', 'P', 'M', 'E05', 'B'),  
(1002, 'Pipo', 'P', 'M', 'E02', 'B'),  
(1003, 'Nuna', 'P', 'H', 'E02', 'A'),  
(1004, 'Americo', 'G', 'M', 'E04', 'A'),  
(1005, 'Sombra', 'P', 'H', 'E05', 'A'),  
(1007, 'Amaya', 'G', 'H', 'E04', 'A'),  
(1008, 'Talia', 'G', 'H', 'E01', 'B'),  
(1009, 'Trabis', 'P', 'M', 'E02', 'A'),  
(1010, 'Titito', 'G', 'M', 'E04', 'B'),  
(1011, 'Truca', 'P', 'M', 'E02', 'A'),  
(1012, 'Zulay', 'P', 'H', 'E05', 'A'),  
(1013, 'Dandi', 'G', 'M', 'E04', 'A'),  
(1014, 'Ras', 'G', 'M', 'E01', 'A'),  
(1015, 'Canela', 'P', 'H', 'E02', 'A'),  
(1016, 'Batan', 'P', 'M', 'E01', 'B'),  
(1017, 'Coco', 'G', 'M', 'E02', 'A');
```



# REPORTES

Pasadas unas semanas nos piden hacer auditoria del centro y responder a las siguientes preguntas:

1. ¿Cuántos perros de cada sexo hay actualmente en el centro?
  2. ¿Cuántos ejemplares contiene actualmente cada jaula o ubicación?
-

# 1) ¿Cuántos perros de cada sexo hay total actualmente en el centro?

¿Dónde están los datos?  
En la tabla mascotas.

¿Qué datos nos piden?  
El número de perros.

```
SELECT sexo, COUNT(*) AS "perros vigentes"  
FROM mascotas  
WHERE especie = 'P' AND estado = 'A'  
GROUP BY sexo;
```

¿Qué requisitos deben  
cumplir los registros?  
Deben ser perros y estar  
de alta en el centro.

¿Cómo debemos agrupar los datos?  
Por sexo.

sexo	perros vigente
H	4
M	2

## 2) ¿Cuántos ejemplares contiene actualmente cada jaula o ubicación?

¿Dónde están los datos?  
En la tabla mascotas.

¿Qué datos nos piden?  
El número de ejemplares.

```
SELECT ubicacion, COUNT(*) AS ejemplares
FROM mascotas
WHERE estado = 'A'
GROUP BY ubicacion;
```

¿Qué requisitos deben cumplir los registros?  
Las mascotas deben estar de alta en el centro.

¿Cómo debemos agrupar los datos?  
Por ubicación.

ubicacion	ejemplares
E01	1
E02	5
E04	3
E05	2

# CASE

---


La declaración `CASE` en SQL retorna un valor en una condición especificada. En una declaración `CASE` simple, este va a evaluar todas las condiciones una por una dentro de una expresión `WHEN`. En el momento en el que la condición y la expresión llegan a coincidir, se devuelve la expresión mencionada en la cláusula `THEN`.



```
SELECT CASE campo
WHEN expresion1 THEN resultado1
WHEN expresion2 THEN resultado2
...
ELSE resultado-por-defecto
END
```

Generalmente, vamos almacenando abreviaturas en una tabla en lugar de su forma completa. En un campo abreviado por ejemplo `sexo`, el valor se guarda como una `M` o `H` y con una declaración `CASE` se puede mostrar estos valores como `Masculino` y `Hembra`.

```
SELECT
CASE especie
  WHEN 'P' THEN 'Perro'
  WHEN 'G' THEN 'Gato'
END,
CASE sexo
  WHEN 'M' THEN 'Masculino'
  WHEN 'H' THEN 'Hembra'
  ELSE sexo
END AS "sexo mascota"
FROM mascotas;
```



especie	sexo mascota
Perro	Masculino
Gato	Hembra
Gato	Masculino
Perro	Hembra

Otro tipo de uso que podemos darle a la sentencia CASE es utilizar expresiones que nos permitan trabajar con rangos de valores. El siguiente ejemplo se basa en el rango de salarios de un grupo de empleados utilizando operadores de comparación y el operador de intervalo BETWEEN:

```
SELECT nombre,  
CASE  
  WHEN salario > 0 AND salario <= 300000 THEN 'Bajo'  
  WHEN salario >= 300000 AND salario <= 500000 THEN 'Promedio'  
  WHEN salario >= 500000 THEN 'Alto'  
END AS "nivel de salario"  
FROM empleados;  
/* equivalente */  
SELECT nombre,  
CASE  
  WHEN salario BETWEEN 0 AND 300000 THEN 'Bajo'  
  WHEN salario BETWEEN 300000 AND 500000 THEN 'Promedio'  
  WHEN salario > 500000 THEN 'Alto'  
END AS "nivel de salario"  
FROM empleados;
```

nombre	nivel de salario
armando	Bajo
pedro	Bajo
francisco	Bajo
javier	Promedio
ignacio	Alto

CHECK

---



```
CREATE TABLE IF NOT EXISTS usuarios (  
  id SERIAL PRIMARY KEY,  
  email VARCHAR(50),  
  nombre VARCHAR(50),  
  apellido VARCHAR(50),  
  rol VARCHAR(30),  
  CONSTRAINT CHK_ROL CHECK (rol = 'administrador' OR rol='usuario')  
);
```

```
INSERT INTO usuarios (email, nombre, apellido, rol)  
VALUES  
(  
'marco@gmail.com', 'marco', 'contreras', 'administrador'),  
(  
'felipe@gmail', 'felipe', 'cruz', 'usuario'),  
(  
'jorge@gmail', 'jorge', 'molina', 'usuario'),  
(  
'mario@gmail', 'mario', 'gutierrez', 'usuario'),  
(  
'jonathan@gmail', 'jonathan', 'villarroel', 'usuario');  
SELECT * FROM usuarios;
```

Clic para ir al script.



EL problema se presenta cuando intentamos insertar un registro especificando un valor en el campo `rol` que no es `usuario` o `administrador`. Lo cual tiene sentido si lo que queremos es que se ingresen ciertos valores en concreto.

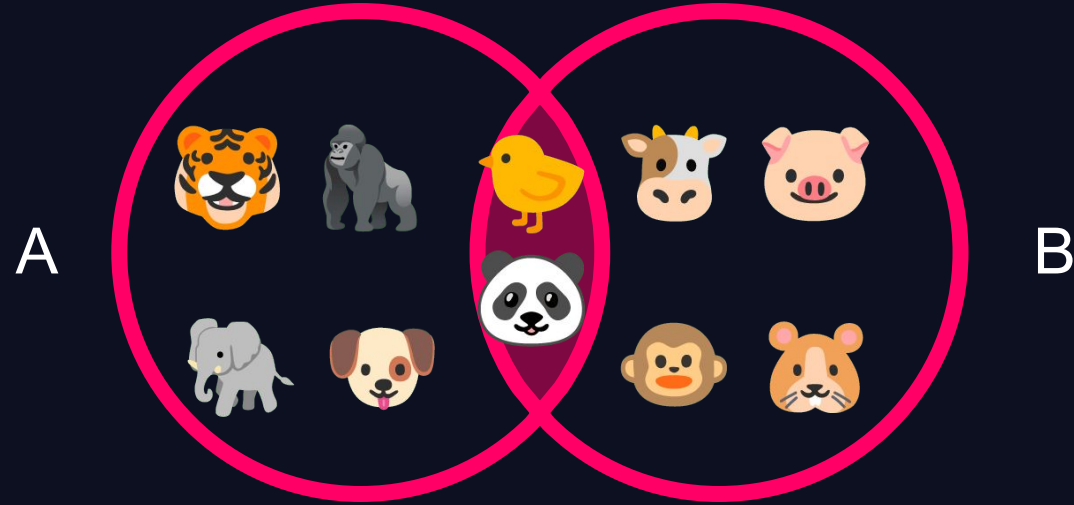


```
INSERT INTO usuarios (nombre, apellido, rol)
VALUES ('marco', 'contreras', 'usuarios');
```

el nuevo registro para la relación «usuarios» viola la restricción «check» «chk\_rol»

# CONSULTAS MULTI-TABLAS

# INNER JOIN

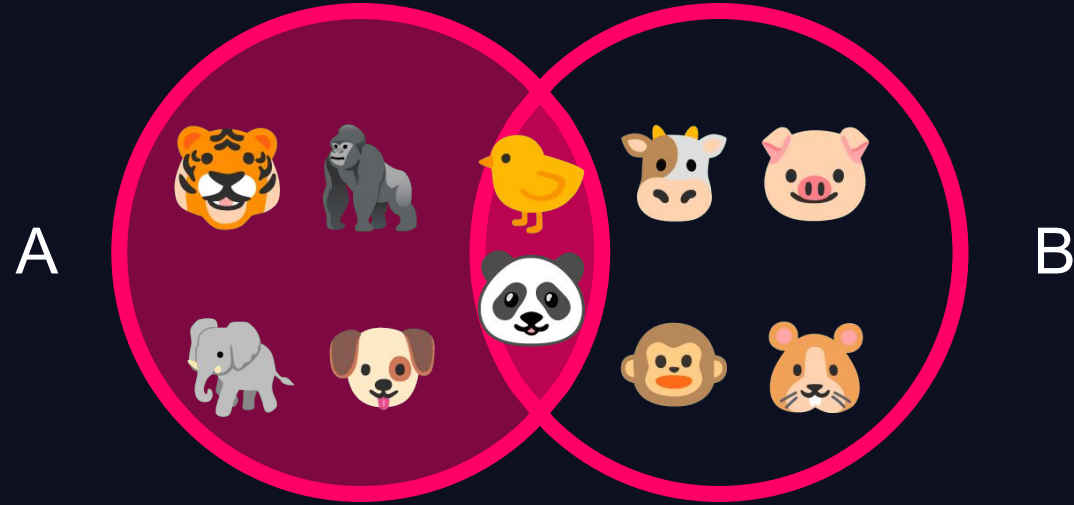


```
SELECT animal  
FROM A  
INNER JOIN B  
ON A.campo = B.campo
```







animal



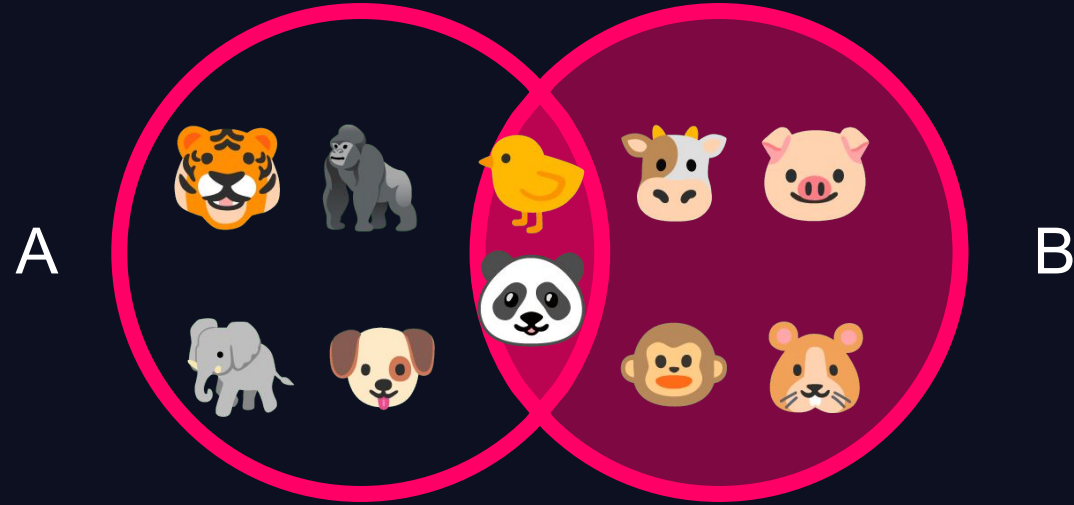
# LEFT JOIN









```
SELECT animal  
FROM A  
LEFT JOIN B  
ON A.campo = B.campo
```

animal







# RIGHT JOIN



```
SELECT animal  
FROM A  
RIGHT JOIN B  
ON A.campo = B.campo
```

animal







INTEGRIDAD DE DATOS

## Tabla - Propietario

FIELD	TYPE	NULL	CONSTRAINT
ID	INT4	NO	PRI
NOMBRE	VARCHAR(50)	NO	
DIRECCION	VARCHAR(80)	NO	
TELEFONO	VARCHAR(12)	NO	
EMAIL	VARCHAR(50)	SI	
CIUDAD	VARCHAR(30)	NO	

## Tabla - Mascotas

FIELD	TYPE	NULL	CONSTRAINT
ID_MASCOTA	INT4	NO	PK
NOMBRE	VARCHAR(30)	NO	
ESPECIE	CHAR(1)	NO	
SEXO	CHAR(1)	NO	
UBICACION	VARCHAR(6)	NO	
ESTADO	CHAR(1)	NO	
ID_PROPIETARIO	INT4	NO	FK



**ON DELETE CASCADE** elimina automáticamente todas las filas de referencias en la tabla secundaria cuando se eliminan las filas a las que hace referencia en la tabla principal. En la práctica, **ON DELETE CASCADE** es la opción más utilizada.





```
DROP TABLE IF EXISTS mascotas;  
DROP TABLE IF EXISTS propietario;
```

```
CREATE TABLE propietario (  
  id INTEGER PRIMARY KEY,  
  nombre VARCHAR(50) NOT NULL,  
  direccion VARCHAR(80) NOT NULL,  
  telefono VARCHAR(12) NOT NULL,  
  email VARCHAR(50),  
  ciudad VARCHAR(30)  
);
```



```
CREATE TABLE mascotas (  
  id BIGINT PRIMARY KEY,  
  nombre VARCHAR(30) NOT NULL,  
  especie CHAR(1),  
  sexo CHAR(1),  
  ubicacion VARCHAR(6),  
  estado CHAR(1),  
  id_propietario INTEGER,  
  CONSTRAINT fk_propietario  
    FOREIGN KEY(id_propietario)  
    REFERENCES propietario(id)  
    ON DELETE CASCADE ←  
);
```