Data Base Administrator PostgreSQL

Ernesto Quiñones A.
ernesto@eqsoft.net

Sesión 2
Estructuras en PostgreSQL

1: Tipos de Datos Básicos

2: Conversiones

3: Creación de objetos básicos

1. Tipos de datos básicos

PostgreSQL maneja muchos tipos de datos, en la mayoría de los casos el conjunto de tipos de datos básicos cubre el 90% e las necesidades, estos son:

a)Tipos de datos numéricos

Name	Storage Size	Description	Range
smallint	2 bytes	small-range integer	-32768 to +32767
integer	4 bytes	typical choice for integer	-2147483648 to +2147483647
bigint	8 bytes	large-range integer	-9223372036854775808 to +9223372036854775807
decimal	variable	user-specified precision, exact	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
numeric	variable	user-specified precision, exact	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
real	4 bytes	variable-precision, inexact	6 decimal digits precision
double precision	8 bytes	variable-precision, inexact	15 decimal digits precision
smallserial	2 bytes	small autoincrementing integer	1 to 32767
serial	4 bytes	autoincrementing integer	1 to 2147483647
bigserial	8 bytes	large autoincrementing integer	1 to 9223372036854775807

Algunas consideraciones:

- **Los tipos de datos de punto flotante por su naturaleza pueden generar problemas de precisión decimal, por ejemplo: el número 10 puede ser que se represente como 9.9999999 (radical al infinito).**
- **Para cálculos financieros usar preferiblemente NUMERIC antes que REAL o DOUBLE.**
- **Los tipos de datos seriales utilizan funciones adicionales para ser utilizados correctamente.**
- **Existe el tipo de datos MONEY que tiene ciertas ventajas:**
  - **Es tan grande como un bigint pero con decimales**
  - **La precisión decimal se define por un parámetro de configuración llamado "lc_monetary"**
  - **En su representación incluye el símbolo de la moneda definida en tu regionalización del sistema operativo.**
- **El serial se relaciona mucho a los SEQUENCE, lo veremos más adelante.**

1. Tipos de datos básicos

b)Tipos de datos caracter.

Name	Description
character varying(*n*), varchar(*n*)	variable-length with limit
character(*n*), char(*n*)	fixed-length, blank padded
text	variable unlimited length

Algunas consideraciones:

- **N es un entero positivo, N no es cantidad de bytes es cantidad de caracteres.**
- **Si en char no se especifica N es 1 carácter, en varchar si no se especifica N entonces es tan grande como el máximo disponible (1Gb).**
- **Una cadena corta ocupa hasta 126bytes + 1byte, una cadena larga ocupa N + 4bytes.**
- **Las cadenas largas se comprimen por defecto antes de ser almacenada, las Muy Largas se guardan en un repositorio especial para no afectar la performance de la db.**

1. Tipos de datos básicos

c)Tipos de datos fecha.

Nombre	Espacio de almacenamiento	Descripción	Valor más bajo	Valor más Alto	Precisión
timestamp [(p)] [without time zone]	8 bytes	both date and time (no time zone)	4713 BC	294276 AD	1 microsecond / 14 digits
timestamp [(p)] with time zone	8 bytes	both date and time, with time zone	4713 BC	294276 AD	1 microsecond / 14 digits
date	4 bytes	date (no time of day)	4713 BC	5874897 AD	1 day
time [(p)] [without time zone]	8 bytes	time of day (no date)	00:00:00	24:00:00	1 microsecond / 14 digits
time [(p)] with time zone	12 bytes	times of day only, with time zone	00:00:00+1459	24:00:00-1459	1 microsecond / 14 digits
interval [fields] [(p)]	16 bytes	time interval	-178000000 years	178000000 years	1 microsecond / 14 digits

Algunas consideraciones:

- **Si se va a trabajar fecha y horas locales es más simple trabajar con datos “without time zone”.**
- **En algunos casos es mejor trabajar el DATE y el TIME como datos separados que utilizar TIMPESTAMP.**
- **La representación de la fecha se puede configurar en el parámetro DateStyle.**
- **Existen muchas tablas de equivalencias y formatos de representación de fechas aceptados por PostgreSQL, esto se puede consultar aquí: https://www.postgresql.org/docs/10/static/datatype-datetime.html#datatype-datetime-date-table**

Otros tipos de datos básicos:

- **Boolean: almacena 1 o 0, TRUE o FALSE.**

- **Tipos de datos de direcciones de red:**
- **Cidr ← solo Ips v.4 ó v.6 especificando netmask**
- **Inet ← Ips v.4 ó v.6 ó hostnames**
- **Macaddr ← dirección de identificador único de una NIC**
- **Macaddr8 ← dirección de identificador único de una NIC**

```
prueba=# create table tbl_inet ( host cidr, hostname varchar(32));

prueba=# insert into tbl_inet values ( '192.168.1.1','192.168.1.1');
INSERT 0 1

prueba=# insert into tbl_inet values ( '292.168.1.1','192.168.1.1');
ERROR:  la sintaxis de entrada no es válida para tipo cidr:
«292.168.1.1»
LÍNEA 1: insert into tbl_inet values ( '292.168.1.1','192.168.1.1');
```

1. Tipos de datos básicos

Otros tipos de datos básicos:

- **Bytea: para almacenar cadenas binarias (principalmente archivos)**
- **OID: númerico del tipo entero que representa un objeto único en la base de datos.**
- **Tipo de dato NULL o NULO**

NO es un tipo de datos en si, es una característica que puede tener un tipo de datos cualquiera y significa que aceptará no tener ningún valor.

OjO:

- **En cadenas una cadena vacía o de longitud 0 es diferente que NULL**
- **En valores numéricos el 0 es diferente que NULL**

Convertir Cadenas a Números

```
template1=# select to_number('12.23','99999');
-----------
      1223

template1=# select to_number('12.23','99D99');
ERROR:  numeric field overflow

template1=# select to_number('12,23','99D99');
-----------
     12.23

template1=# select to_number('12.23','99.99');
-----------
     12.23

template1=# select to_number('1,112.23','999.99');
ERROR:  numeric field overflow

template1=# select to_number('1,112.23','9999.99');
-----------
    1112.2

template1=# select to_number('1,112.23','9,999.99');
-----------
   1112.23
```

```
template1=# select
to_number('A12.23','9,999.99');
-----------
      2.23

template1=# select
to_number('A12.23','999.99');
-----------
     12.23
```

https://www.postgresql.org/docs/10/static/functions-formatting.html#functions-formatting-numeric-table

Convertir Números a Cadenas

```
template1=# select to_char(123,'999');
---------
  123

template1=# select to_char(3123,'999');
---------
  ###

template1=# select to_char(3123,'0999');
---------
  3123

template1=# select to_char(3123,'000999');
---------
  003123

template1=# select to_char(3.23,'9999');
---------
     3
```

```
template1=# select to_char(3.23,'9.99');
---------
  3.23

template1=# select to_char(3.23,'9D99');
---------
  3,23
```

https://www.postgresql.org/docs/10/static/functions-formatting.html#functions-formatting-examples-table

Convertir Fechas a Cadenas

```
template1=# select to_char(now(),'HH12:MI:SS');
----------
 11:12:01

template1=# select to_char(now(),'HH24:MI:SS');
----------
 23:12:08

template1=# select to_char(now(),'dd-mm-yy');
----------
 02-03-17

template1=# select to_char(now(),'dd-MM-yy');
----------
 02-03-17

template1=# select to_char(now(),'dd-MM-yyyy');
------------
 02-03-2017

template1=# select to_char(now(),'dd-Month-yyyy');
--------------------
 02-March    -2017
```

https://www.postgresql.org/docs/10/static/functions-formatting.html#functions-formatting-datetime-table

Convertir Fechas a Cadenas

```
template1=# select to_char(now(),'dd-Mon-yyyy');
-------------
 02-Mar-2017

template1=# select to_char(now(),'day-Mon-yyyy');
--------------------
 thursday -Mar-2017

template1=# select to_char(now(),'DAY-Mon-yyyy');
--------------------
 THURSDAY -Mar-2017

template1=# select to_char(now(),'DAY dd-Mon-yyyy');
-----------------------
 THURSDAY  02-Mar-2017
```

2. Conversiones de Tipos de Datos

Convertir Cadenas a Fechas

```
template1=# select now() - '15-01-2000';
---------------------------
 6256 days 23:22:39.215958

template1=# select now() - '15-Jan-2000';
---------------------------
 6256 days 23:22:42.011579

template1=# select now() - 'Jan-15-2000';
---------------------------
 6256 days 23:22:53.691821

template1=# select now() - '2000-01-15';
---------------------------
 6256 days 23:23:02.535525

template1=# select  '2000-01-15' || '-->ernesto';
----------------------
 2000-01-15-->ernesto
```

Convertir Cadenas a Fechas

```
template1=# select to_date('01-15-2010', 'MM-dd-YY');
------------
 2010-01-15

template1=# select to_date('01-35-2010', 'MM-dd-YY');
------------
 2010-02-04

template1=# select to_date('Jan-15-2010', 'Mon-dd-YY');
------------
 2010-01-15

template1=# select to_date('Ene-15-2010', 'Mon-dd-YY');
ERROR:  invalid value "Ene" for "Mon"

template1=# select to_date('01-15-69', 'mm-dd-yy');
------------
 2069-01-15

template1=# select to_date('01-15-70', 'mm-dd-yy');
------------
 1970-01-15
```

Los valores de formato son los mismos que los que figuran en el enlace de la diapositiva 13.

Convertir Cadenas a Horas

```
template1=# select to_timestamp('11:12:45', 'hh:mi:ss');
---------------------------------
 0001-01-01 11:12:45-05:08:12 BC

template1=# select to_timestamp('23:12:45', 'hh:mi:ss');
ERROR:  hour "23" is invalid for the 12-hour clock
HINT:  Use the 24-hour clock, or give an hour between 1 and 12.

template1=# select to_timestamp('23:12:45', 'hh24:mi:ss');
---------------------------------
 0001-01-01 23:12:45-05:08:12 BC

template1=# select to_timestamp('23:72:45', 'hh24:mi:ss');
---------------------------------
 0001-01-02 00:12:45-05:08:12 BC

template1=# select to_timestamp('23-12-45', 'hh24:mi:ss');
---------------------------------
 0001-01-01 23:12:45-05:08:12 BC
```

2. Conversiones de Tipos de Datos

CAST

```
template1=# select cast('123' as integer);
 int4
------
  123

template1=# select cast('123.1' as integer);
ERROR:  invalid input syntax for integer: "123.1"
                    ^
template1=# select cast('123.1' as numeric);
 numeric
---------
   123.1
                                  ^
template1=# select cast('2011-01-15' as date);
    date
------------
 2011-01-15

template1=# select cast('2011-Jan-15' as date);
    date
------------
 2011-01-15

template1=# select cast('2011-Ene-15' as date);
ERROR:  invalid input syntax for type date: "2011-Ene-15"
```

2. Conversiones de Tipos de Datos

Operador ::

```
template1=# select (12.3 as integer);
ERROR:  syntax error at or near "as"
LINE 1: select (12.3 as integer);

                     ^
template1=# select '12'::integer;
 int4
------
   12


template1=# select '12.1'::float;
 float8
--------
   12.1


template1=# select '12 jan 2010'::date;
    date
------------
 2010-01-12
```

3. Creación de Objetos Básicos

El comando para la creación de objetos en una base de datos es el comando CREATE, todo lo que se pueda crear en una base de datos es creado usando este comando.

```
CREATE DATABASE -- create a new database
CREATE DOMAIN -- define a new domain
CREATE FUNCTION -- define a new function
CREATE GROUP -- define a new database role
CREATE INDEX -- define a new index
CREATE LANGUAGE -- define a new procedural language
CREATE OPERATOR -- define a new operator
CREATE ROLE -- define a new database role
CREATE RULE -- define a new rewrite rule
CREATE SCHEMA -- define a new schema
CREATE SEQUENCE -- define a new sequence generator
CREATE TABLE -- define a new table
CREATE TABLE AS -- define a new table from the results of a query
CREATE TABLESPACE -- define a new tablespace
CREATE TRIGGER -- define a new trigger
CREATE TYPE -- define a new data type
CREATE USER -- define a new database role
CREATE VIEW -- define a new view
```

3. Creación de Objetos Básicos

Los primeros que vamos a ver son:

```
CREATE DATABASE -- create a new database
CREATE DOMAIN -- define a new domain
CREATE FUNCTION -- define a new function
CREATE GROUP -- define a new database role
CREATE INDEX -- define a new index
CREATE LANGUAGE -- define a new procedural language
CREATE OPERATOR -- define a new operator
CREATE ROLE -- define a new database role
CREATE RULE -- define a new rewrite rule
CREATE SCHEMA -- define a new schema
CREATE SEQUENCE -- define a new sequence generator
CREATE TABLE -- define a new table
CREATE TABLE AS -- define a new table from the results of a query
CREATE TABLESPACE -- define a new tablespace
CREATE TRIGGER -- define a new trigger
CREATE TYPE -- define a new data type
CREATE USER -- define a new database role
CREATE VIEW -- define a new view
```

Pero existen varios otros más:

```
CREATE AGGREGATE -- define a new aggregate function
CREATE CAST -- define a new cast
CREATE COLLATION -- define a new collation
CREATE CONVERSION -- define a new encoding conversion
CREATE EVENT TRIGGER -- define a new event trigger
CREATE EXTENSION -- install an extension
CREATE FOREIGN DATA WRAPPER -- define a new foreign-data wrapper
CREATE FOREIGN TABLE -- define a new foreign table
CREATE MATERIALIZED VIEW -- define a new materialized view
CREATE OPERATOR CLASS -- define a new operator class
CREATE OPERATOR FAMILY -- define a new operator family
CREATE SERVER -- define a new foreign server
CREATE TEXT SEARCH CONFIGURATION -- define a new text search
configuration
CREATE TEXT SEARCH DICTIONARY -- define a new text search dictionary
CREATE TEXT SEARCH PARSER -- define a new text search parser
CREATE TEXT SEARCH TEMPLATE -- define a new text search template
CREATE USER MAPPING -- define a new mapping of a user to a foreign
server
```

SQL – Create y Alter

Cada Create tiene su sintaxis específica, esto depende del tipo de objeto que estamos creado, en algunos casos como en "create table" la sintaxis puede ser bastante amplia y compleja y en algunos casos como en "create database" es más sencilla, depende de la utilidad del objeto.

Alter permite modificar la estructura de un objeto, en general todo objeto creado puede ser modificado pero no todos sus atributos pueden ser modificados, así algunas características podrían estar restringidas y será necesario consultarlo en la documentación, algunas podrían incluso ser restricciones de integridad de la base de datos, por ejemplo será imposible borrar una llave primaria en una tabla si es que las referencias que tiene en otras tablas no son eliminadas previamente.

SQL – Create y Drop

Create permite crear objetos en la base de datos, básicamente cualquier objeto puede ser creado con la sintaxis:

Create tipo_de_objeto nombre_de_objeto caracteristicas

```
create database nombre_objeto
create table nombre_objeto (estructura)
create index nombre_objeto on table nombre_tabla
create view nombre_objeto as sentencia_sql
.... etc.
```

Drop permite borrar los objetos de una base de datos.

```
drop database nombre_objeto
drop table nombre_objeto
drop index nombre_objeto
drop view nombre_objeto
... etc.
```

Drop permitirá eliminar un objeto que no tenga restricciones de integridad por estar referencia en otra sección de la base de datos.

SQL – Create Database

```
CREATE DATABASE name
    [ [ WITH ] [ OWNER [=] user_name ]
           [ TEMPLATE [=] template ]
           [ ENCODING [=] encoding ]
           [ LC_COLLATE [=] lc_collate ]
           [ LC_CTYPE [=] lc_ctype ]
           [ TABLESPACE [=] tablespace_name ]
           [ CONNECTION LIMIT [=] connlimit ] ]
```

```
OWNER      :propietario de la DB
TEMPLATE   :crear una DB en base a otra DB previamente creada
ENCODING   :mapa de caracteres a usar (por ejemplo SQL_ASCII)
LC_COLLATE :define orden de los textos (regionalización)
LC_CTYPE   :tipo de caracteres para conversiones (regionalización)
TABLESPACE :lugar físico donde podemos colocar la DB
CONNECTION LIMIT :número máximo de conexiones autorizadas
```

SQL – Create Database

```
create database nombre_base_datos
```

```
usuario@host:# psql template1 -U USUARIO
Contraseña para usuario USUARIO:
Psql (9.4.5)
Digite «help» para obtener ayuda.

template1=# create database prueba2;
CREATE DATABASE
template1=# \q

usuario@host:#  psql prueba2 -U USUARIO
Contraseña para usuario USUARIO:
Psql (9.4.5)
Digite «help» para obtener ayuda.

prueba2=#
```

SQL – Create Table

En su forma más básica la sintaxis es:

```
create table nombre_tabla ( nombre_de_campo tipo_de_datos);
```

```
   prueba2=# create table tablita (campo1 char(10), campo2 int);
   CREATE TABLE

   prueba2=# \dt tablita
        Listado de relaciones
    Esquema | Nombre  | Tipo  |  Dueño
   ---------+---------+-------+---------
    public  | tablita | tabla | dbadmin
   (1 fila)

   prueba2=# \d tablita
         Tabla «public.tablita»
    Columna |     Tipo      | Modificadores
   ---------+---------------+---------------
    campo1  | character(10) |
    campo2  | integer       |
```

*** El owner de la tabla por defecto es el usuario quien la crea.**

SQL – Create Table

Podemos crear tablas temporales, estas se mantienen vivas solo mientras nuestra conexión este activa, cuando salgamos de la DB la tabla se eliminará, pero mientras tanto podrá usarse como cualquier tabla normal, pero solo por el usuario que la creo..

```
prueba2=# create temp table temporal (id int, nombre char(100));
CREATE TABLE
prueba2=# \d
           List of relations
 Schema   |   Name   | Type  |  Owner
-----------+----------+-------+----------
pg_temp_2 | temporal | table | postgres
(1 row)

prueba2=# \q
postgres@ernesto-VirtualBox:~$ psql prueba2
psql (9.4.5)
Type "help" for help.

prueba2=# \d
No relations found.
prueba2=#
```

Schemas

Los esquemas son contenedores de objetos dentro de una db, por default siempre trabajamos en el schema PUBLIC.

La principal utilidad puede estar en la administración de permisos a los objetos de la db ó simplemente la organización.

```
prueba4=# create schema public2;
prueba4=# create table public2.tbl_prueba(id integer);
prueba4=# create table public.tbl_prueba(id integer);
prueba4=# insert into tbl_prueba values(1);
prueba4=# insert into public2.tbl_prueba(2);

prueba4=# select * from public.tbl_prueba;
 id
----
  1
prueba4=# select * from public2.tbl_prueba;
 id
----
  2
```

Schemas

```
CREATE SCHEMA schema_name [ AUTHORIZATION user_name ]
[ schema_element [ ... ] ]

user_name      : owner del schema
schema_element : podemos crear esquemas que solo permitan tablas,
                 vistas, etc.

CREATE SCHEMA IF NOT EXISTS schema_name [ AUTHORIZATION user_name ]
```

Si añadimos IF NOT EXISTS y no existe el schema entonces lo creará, pero si existe no lo hará pero no nos dará un mensaje de error:

```
prueba2=# create schema esquemita;
CREATE SCHEMA

prueba2=# create schema esquemita;
ERROR:  schema "esquemita" already exists

prueba2=# create schema if not exists esquemita;~
NOTICE:  schema "esquemita" already exists, skipping
CREATE SCHEMA
```

Schemas

```
CREATE SCHEMA AUTHORIZATION user_name [ schema_element [ ... ] ]
CREATE SCHEMA IF NOT EXISTS AUTHORIZATION user_name
```

Usando el modificador AUTHORIZATION se crea un schema con el mismo nombre del usuario especificado.

```
prueba2-# \dn
  List of schemas
  Name    |  Owner
----------+----------
esquemita | postgres
public    | postgres

prueba2=# create schema authorization postgres;
CREATE SCHEMA
prueba2=# \dn
  List of schemas
  Name    |  Owner
----------+----------
esquemita | postgres
postgres  | postgres
public    | postgres

prueba2=# create schema authorization ernesto; <--Usuario Ernesto no existe
ERROR:  role "ernesto" does not exist
```

Types

Los Types son estructuras de datos definidas por el usuario.

Types de datos enumerados

```
template1=# create type mes as enum ('enero', 'febrero', 'marzo');
CREATE TYPE

template1=# create table tbl_mes ( fecha mes);
CREATE TABLE

template1=# insert into tbl_mes values ('enero');
INSERT 0 1

template1=# insert into tbl_mes values ('abril');
ERROR:  la sintaxis de entrada no es válida para el enum mes:
«abril»
LÍNEA 1: insert into tbl_mes values ('abril');
```

El orden de los enumerados en caso de aplicar al campo un "order by" es el orden en el que fueron declarados los valores.

Types

Crear una tabla de un Type nos ayuda a tener estructuras pre-construidas para los datos.

```
prueba4=# create type tpy_usuario as (id integer,
                    password char(25), nombre varchar(100));

prueba4=# create table tbl_usuario of tpy_usuario;

prueba4=# select * from tbl_usuario;

 id | password | nombre
----+----------+--------
(0 rows)

prueba4=# drop type tpy_usuario cascade;
NOTICE:  drop cascades to table tbl_usuario
DROP TYPE
```

Types

Un campo de la tabla puede tener la estructura de un Type.

```
prueba4=# create table tbl_usuario2 ( id serial, datos_unicos
tpy_usuario);

prueba4=# insert into tbl_usuario2 values (1,
(1,'password','nombre'));
prueba4=# insert into tbl_usuario2 (datos_unicos.id,
datos_unicos.password, datos_unicos.nombre)  values (2, 'password2',
'nombre2');

prueba4=# select * from tbl_usuario2;
 id |              datos_unicos
----+----------------------------------------
  1 | (1,"password                ",nombre)
  1 | (2,"password2               ",nombre2)

prueba4=# select (datos_unicos).password from tbl_usuario2;
         password
--------------------------
 password
 password2
```

Types

Borrar el type hará que se pierdan todas los campos creados a partir de este en todas las tablas donde se haya usado.

```
prueba4=# drop type tpy_usuario cascade;
NOTICE:  drop cascades to 2 other objects
DETAIL:  drop cascades to table tbl_usuario
drop cascades to table tbl_usuario2 column datos_unicos
DROP TYPE


prueba4=# select * from tbl_usuario2;
 id
----
  1
  1
```

Types

RANGE es un type especial que permite gestionar rangos de valores.

```
alterno1=# create type rango1 as range (subtype=integer);
alterno1=# create table t1 (c1 rango1);
alterno1=# insert into t1 values ('[1,100]');
alterno1=# select * from t1;
   c1
---------
 [1,100]
                                ^
alterno1=# select lower(c1) from t1;
 lower
-------
     1

alterno1=# select upper(c1) from t1;
 upper
-------
   100
                        ^
alterno1=# select 1 from t1 where 3 <@ c1;
 ?column?
----------
        1
```

```
alterno1=# select 1 from t1 where 300 <@ c1;
 ?column?
----------
(0 filas)
```

https://www.postgresql.org/docs/10/static/functions-range.html#range-functions-table

Secuencias (Sequences)

Una secuencia es un tipo de datos auto-incremental, podría o no estar ligado a una tabla como valor por defecto para un campo específico (generalmente para la llave primaria), en esencia es un tipo de datos entero.

```
pruebas=# create table tbl_simple (id serial, nombre varchar(10));
NOTICE:  CREATE TABLE creará una secuencia implícita «tbl_simple_id_seq» para
la columna serial «tbl_simple.id»

pruebas=# insert into tbl_simple (nombre) values ('ernesto') returning id;
 id
----
  1
INSERT 0 1

pruebas=# insert into tbl_simple (nombre) values ('juan') returning id;
 id
----
  2

INSERT 0 1
pruebas=# insert into tbl_simple (nombre) values ('10chars') returning id;
 id
----
  3
```

Secuencias (Sequences)

Algunas funciones pre-definidas de PostgreSQL nos permiten administrar las secuencias:

```
pruebas=# \d tbl_simple;
Tabla «public.tbl_simple»
 Columna |         Tipo          |              Modificadores

+-------------------------------------------------------------------
 id      | integer               | not null valor por omisión
                                   nextval('tbl_simple_id_seq'::regclass)
 nombre  | character varying(10) |

pruebas=# select currval('tbl_simple_id_seq'); ← consulta el valor actual
 currval
---------
       3

pruebas=# select nextval('tbl_simple_id_seq'); ← aumenta el valor el 1 de
 nextval                                         la secuencia
---------
       4

pruebas=# select currval('tbl_simple_id_seq');
 currval
---------
       4
```

Secuencias (Sequences)

```
pruebas=# insert into tbl_simple (nombre) values ('10char') returning id;
 id
----
  5

pruebas=# select setval('tbl_simple_id_seq',1);  <-- modifica el valor de la
 setval                                              secuencia
--------
      1

pruebas=# insert into tbl_simple (nombre) values ('10cha') returning id;
 id
----
  2

pruebas=# select currval('tbl_simple_id_seq');
 currval
---------
       2

pruebas=# select * from tbl_simple;
 id | nombre
----+---------
  1 | ernesto
  2 | juan
  3 | 10chars
  5 | 10char
  2 | 10cha
```

Secuencias (Sequences)

```
pruebas=# \d tbl_simple_id_seq  ← consultar la estructura
                                  de la secuencia

     Secuencia «public.tbl_simple_id_seq»
    Columna     |  Tipo   |        Valor
----------------+---------+---------------------
 sequence_name | name    | tbl_simple_id_seq
 last_value    | bigint  | 2
 start_value   | bigint  | 1
 increment_by  | bigint  | 1
 max_value     | bigint  | 9223372036854775807
 min_value     | bigint  | 1
 cache_value   | bigint  | 1
 log_cnt       | bigint  | 32
 is_cycled     | boolean | f
 is_called     | boolean | t
```

Secuencias (Sequences)

Una funcionalidad interesante puede ser aplicable en entornos donde se producen muchas “inserciones” de data, y es mantener una cantidad de números en memoria para un acceso más rápido a los contadores.

```
prueba4=# create sequence contador cache 100;
```
← reserva 100 números en Cache

Cycle permite reiniciar el contador con el mínimo valor especificado cuando este llega a su máximo valor declarado.

```
prueba4=# create sequence cmin_max cycle minvalue 1 maxvalue 3;
```

Increment permite definir cada cuantas unidades aumentará el contador, positivo ó negativo.

```
prueba4=# create sequence salto increment -10;
```

Secuencias (Sequences)

```
CREATE [ TEMPORARY | TEMP ] SEQUENCE [ IF NOT EXISTS ] name
    [ AS data_type ]
    [ INCREMENT [ BY ] increment ]
    [ MINVALUE minvalue | NO MINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE ]
    [ START [ WITH ] start ] [ CACHE cache ] [ [ NO ] CYCLE ]
    [ OWNED BY { table_name.column_name | NONE } ]
```

- Las secuencias pueden ser temporales y persisten en la sesión del usuario que la creo.
- El data_type puede ser smallint, integer y bigint, siendo este último el valor por defecto.
- NO MINVALUE y NO MAXVALUE fuerza a utilizar los límites del tip ode datos usado.
- NO CYCLE genera un error cuando se llega al número máximo mayor posible.