Data Base Administrator PostgreSQL

Ernesto Quiñones A.
ernesto@eqsoft.net

Sesión 8

Tipos de Datos Avanzados Y Binarios

Encriptación

Tipos de Datos - Avanzados

a)Tsvector

El tipo de datos Vector nos permite almacenar cadenas en lexemas para facilitar las búsquedas de cadenas específicas en textos muy largos.

```
prueba=# create table tbl_vector ( campo1 text, campo2 tsvector);

prueba=# insert into tbl_vector values ( 'esta es una prueba de un vector
repitiendo la palabra vector','esta es una prueba de un vector repitiendo la
palabra vector');

prueba=# insert into tbl_vector values ( 'esta es una prueba de un vector
repitiendo la palabra vector','esta es una prueba de un vector repitiendo la
palabra vector');

prueba=# insert into tbl_vector values ( 'árbol y vector añade acentos y
eñes','árbol y vector añade acentos y eñes');

prueba=# insert into tbl_vector values ( 'las ratas y los ratones estan
ratoneando un rato', 'las ratas y los ratones estan ratoneando un rato');
```

Tipos de Datos - Avanzados

a)Tsvector

El tipo de datos Vector nos permite almacenar cadenas en lexemas para facilitar las búsquedas de cadenas específicas en textos muy largos.

```
prueba=# select * from tbl_vector;
```

```
                              campo1                                |             campo2
------------+-------------------------------------------------------------------------------
 esta es una prueba de un vector                                    | 'de' 'es' 'esta' 'prueba'
                                                                      'un' 'una' 'vector'
 esta es una prueba de un vector repitiendo la palabra vector       | 'de' 'es' 'esta' 'la'
                                                                      'palabra' 'prueba'
                                                                      'repitiendo' 'un' 'una'
                                                                      'vector'
 árbol y vector añade acentos y eñes                                | 'acentos' 'añade' 'eñes'
                                                                      'vector' 'y' 'árbol'
 las ratas y los ratones estan ratoneando un rato                   | 'estan' 'las' 'los' 'ratas'
                                                                      'rato' 'ratoneando'
                                                                      'ratones' 'un' 'y'
```

Tipos de Datos - Avanzados

Tsvector

```
prueba=# select campo1 from tbl_vector where to_tsvector(campo1) @@
to_tsquery('añade');
                campo1
-------------------------------------
 árbol y vector añade acentos y eñes


prueba=# select campo1 from tbl_vector where to_tsvector(campo1) @@
to_tsquery('vector & palabra');
                             campo1
-------------------------------------------------------------
 esta es una pruena de un vector repitiendo la palabra vector
```

Tipos de Datos - Avanzados

Tsvector

Podemos acceder directamente al campo del vector.

```
prueba=# select campo1 from tbl_vector where campo2 @@
to_tsquery('pg_catalog.spanish','vector');
                            campo1
-----------------------------------------------------------
 esta es una pruena de un vector
 esta es una pruena de un vector repitiendo la palabra vector
 árbol y vector añade acentos y eñes
```

Pero esto no funciona, ¿porque?

```
prueba=# select campo1 from tbl_vector where campo2 @@
to_tsquery('pg_catalog.spanish','palabra');
 campo1
--------
(0 filas)
```

Tipos de Datos - Avanzados

Tsvector

Para una mejor administración de los datos debemos especificar el idioma en el cual estamos añadiendo los datos.

```
prueba=# insert into tbl_vector values('esto añade soporte en
español', to_tsvector('pg_catalog.spanish','esto añade soporte en
español'));

prueba=# select campo2 from tbl_vector;
                                  campo2
-----------------------------------------------------------------
 'acentos' 'añade' 'eñes' 'vector' 'y' 'árbol'
 'añad':2 'español':5 'soport':3
```

Tipos de Datos - Avanzados

Tsvector

Para una mejor administración de los datos debemos especificar el idioma en el cual estamos añadiendo los datos.

```
prueba=# select campo1 from tbl_vector where campo2 @@
to_tsquery('pg_catalog.spanish','añad');
            campo1
-------------------------------
 esto añade soporte en español


prueba=# select campo1 from tbl_vector where campo2 @@
to_tsquery('pg_catalog.spanish','añada');
            campo1
-------------------------------
 esto añade soporte en español
```

Tipos de Datos - Avanzados

Tsvector

Tenemos a nuestra disposición diferentes operadores lógicos

```
prueba=# delete from tbl_vector;
prueba=# insert into tbl_vector values('esto añade soporte en
español', to_tsvector('pg_catalog.spanish','esto añade soporte en
español'));
prueba=# insert into tbl_vector values('vamos a repetir vector 2
veces más, vector, vector', to_tsvector('pg_catalog.spanish','vamos a
repetir vector 2 veces más, vector, vector'));

prueba=# select campo1 from tbl_vector where campo2 @@
to_tsquery('añada & vector');
 campo1
--------
(0 filas)
```

Tipos de Datos - Avanzados

Tsvector

Tenemos a nuestra disposición diferentes operadores lógicos

```
rueba=# select campo1 from tbl_vector where campo2 @@ to_tsquery('añada |
vector');
                        campo1
-----------------------------------------------------
 esto añade soporte en español
 vamos a repetir vector 2 veces más, vector, vector
(2 filas)

prueba=# select campo1 from tbl_vector where campo2 @@ to_tsquery('!vec');
            campo1
-------------------------------
 esto añade soporte en español

prueba=# select campo1 from tbl_vector where campo2 @@ to_tsquery('!vec &
añada');
            campo1
-------------------------------
 esto añade soporte en español
```

Tipos de Datos - Avanzados

Tsvector

Para combinar operadores lógicos entre varios campos:
&& <-- and
•|| <-- or
•!! <-- not

Los vectores se pueden indexar, para ellos existe el tipo de índice GIN.

```
prueba=# create index idx_vector3 on tbl_vector using gin(campo2);
CREATE INDEX
```

En caso de que los textos se actualicen demasiado seguido entonces podemos crear un trigger para que actualice la columna donde están guardados los lexemas.

Tipos de Datos - Avanzados

Tsvector

Para listar los lexemas más usados:

```
prueba=# SELECT * FROM ts_stat('SELECT campo2 FROM tbl_vector') ORDER
BY nentry DESC, ndoc DESC, word LIMIT 5;
  word   | ndoc | nentry
---------+------+--------
 cociner |    1 |      5
 adrià   |    1 |      3
 gaston  |    1 |      3
 mund    |    1 |      3
 peruan  |    1 |      3
```

Tipos de Datos - Avanzados

Arreglos

Los arreglos en PostgreSQL son como en cualquier lenguaje de programación, su longitud y dimensión es variable.

```
prueba4=# create table tbl_array (data int[]);
prueba4=# insert into tbl_array values('{1,2,3,4,5}');
prueba4=# select * from tbl_array;
    data
-------------
 {1,2,3,4,5}

prueba4=# alter table tbl_array add column data2 int[][];
prueba4=# insert into tbl_array (data2) values('{{1,2},{2,3},{3,4},
{4,5},{5,6}}');
prueba4=# select * from tbl_array;
    data     |              data2
-------------+---------------------------------
 {1,2,3,4,5} |
             | {{1,2},{2,3},{3,4},{4,5},{5,6}}
```

Tipos de Datos - Avanzados

Arreglos

```
prueba4=# select data from tbl_array where data[2]=2;
    data
-------------
 {1,2,3,4,5}

prueba4=# select data from tbl_array where data[2]=1;
 data
------
(0 rows)

prueba4=# select data2 from tbl_array where data2[1][2]=2;
              data2
---------------------------------
 {{1,2},{2,3},{3,4},{4,5},{5,6}}

prueba4=# select data2 from tbl_array where data2[1][3]=2;
 data2
-------
(0 rows)
```

Tipos de Datos - Avanzados

Arreglos

```
prueba4=# alter table tbl_array add column data3 int[][3];
ALTER TABLE

prueba4=# insert into tbl_array (data3) values('{{1,2,3},{2,3,4}}');
INSERT 0 1
prueba4=# select * from tbl_array;

    data     |               data2                |       data3
-------------+------------------------------------+-------------------
 {1,2,3,4,5} |                                    |
             | {{1,2},{2,3},{3,4},{4,5},{5,6}}    |
             |                                    | {{1,2,3},{2,3,4}}
(3 rows)
```

Tipos de Datos - Avanzados

Arreglos

```
prueba4=# select data3 from tbl_array where data3[2][3]=4;
       data3
--------------------
 {{1,2,3},{2,3,4}}
(1 row

prueba4=# select data3 from tbl_array where data3[2][5]=4;
 data3
-------
(0 rows)
```

Tipos de Datos - Avanzados

OIDs

Los OIDs son identificadores únicos de los objetos en la base de datos, incluso cada registro tiene un OID propio que jamas se repite en ninguna otra tabla.

```
prueba4=# create table tbl_oid2(id integer) with oids;
prueba4=# insert into tbl_oid2 values(1);
INSERT 24446 1
prueba4=# insert into tbl_oid2 values(2);
INSERT 24447 1
prueba4=# select * from tbl_oid2;
 id
----
  1
  2

prueba4=# select oid,* from tbl_oid2;
  oid  | id
-------+----
 24446 |  1
 24447 |  2
```

Tipos de Datos - Avanzados

OIDs

Para obtener los oids de los objetos en la base de datos

```
prueba4=# select oid,relname from pg_class where substr(relname,1,3) = 'tbl';
  oid  |       relname
-------+---------------------
...
 24440 | tbl_oid
 24443 | tbl_oid2

prueba4=#  SELECT attrelid,attname FROM pg_attribute where attrelid = 'tbl_oid2'::Regclass;
 attrelid | attname
----------+----------
    24443 | tableoid
...
    24443 | oid
...
    24443 | id

prueba4=#  SELECT attrelid,attname FROM pg_attribute where attrelid = 'vw_vista'::Regclass;
 attrelid | attname
----------+---------
    24380 | id
    24380 | nombre
```

Tipos de Datos - Avanzados

OIDs

Podemos borrar data apuntando a su OID

```
prueba4=# delete from tbl_oid2 where oid = 24447;
DELETE 1

prueba4=# select * from tbl_oid2;

 id
----
  1
(1 row)
```

Ahora por defecto ya no se crean OIDs para data, esto debido a que en volúmenes muy grandes de datos los OIDs podrían llegar a ser insuficientes y es por ello que se prescinde de ellos por defecto.

Tipos de Datos - Avanzados

XML

XML es un lenguaje de marcas muy utilizado principalmente en el mundo del intercambio de datos entre sistemas estandarizados, PostgreSQL no es una base de datos orientada al XML, su soporte se da a nivel de tipo de datos y se implementan funciones para “parsear” el contenido bajo formato XML.}

```
DBNAME=# create table tablaXML (id serial, texto xml);
CREATE TABLE
```

Para asegurarnos que el dato almacenado estre bien formado usamos la función XMLPARSE.

```
DBNAME=# insert into tablaXML (texto) values (xmlparse(content
'<novela><nombre>fundacion</nombre><autor>asimov</autor></novela>'));
INSERT 0 1
```

Tipos de Datos - Avanzados

XML

```
DBNAME=# insert into tablaXML (texto) values (
           xmlparse(content
 '<novela><nombre>fundacion</nombre><autor>asimov</autor>
  </novela>
  <novela><nombre>fundacion e imperio</nombre> <autor>asimov</autor>
</novela>'));
INSERT 0 1

DBNAME=# select * from tablaxml;
 id |                                      texto
----+-----------------------------------------------------------------
  3 | <novela><nombre>fundacion</nombre><autor>asimov</autor></novela>
  4 | <novela><nombre>fundacion</nombre><autor>asimov</autor></novela>
<novela><nombre>fundacion e imperio</nombre> <autor>asimov</autor></novela>
(2 rows)
```

El registro con ID 4 puede dar la impresión se haber almacenado correctamente los datos, pero cuando se busque información retornará un mensaje de error.
OJO: entre las marcas (lo que esta entre <>) no puede haber espacios o enters.

Tipos de Datos - Avanzados

XML

XPATH nos permite explorar el contenido de los XML almacenados.

```
DBNAME=# select id, xpath('/novela', texto) from tablaXML;
 id |                  xpath
----+-----------------------------------------
  3 | {"<novela>                              +
    |   <nombre>fundacion</nombre>            +
    |   <autor>asimov</autor>                 +
    | </novela>"}
  4 | {"<novela>                              +
    |   <nombre>fundacion e imperio</nombre>+
    |   <autor>asimov</autor>                 +
    | </novela>"}

DBNAME=# select id, xpath('/novela/nombre', texto) from tablaXML;
 id |                   xpath
----+-------------------------------------------
  3 | {<nombre>fundacion</nombre>}
  4 | {"<nombre>fundacion e imperio</nombre>"}
```

Tipos de Datos - Avanzados

XML

```
DBNAME=# insert into tablaXML (texto) values (xmlparse(content
'<libro><nombre>matematica
1</nombre><autor>cbeñas</autor></libro>'));
INSERT 0 1

DBNAME=# select id, xpath('/novela/nombre', texto) from tablaXML;
 id |                  xpath
----+-----------------------------------------
  3 | {<nombre>fundacion</nombre>}
  4 | {"<nombre>fundacion e imperio</nombre>"}
  6 | {}

DBNAME=# select id, xpath('/libro/nombre', texto) from tablaXML;
 id |               xpath
----+-----------------------------------
  3 | {}
  4 | {}
  6 | {"<nombre>matematica 1</nombre>"}
```

Tipos de Datos - Avanzados

XML

XPATH_EXITS nos permitirá determinar si el contenido consultado existe.

```
DBNAME=# select id, xpath('/libro/nombre', texto) as textocadena from
tablaXML where xpath_exists('/libro/nombre',texto) ;
 id |            textocadena
----+-----------------------------------
  6 | {"<nombre>matematica 1</nombre>"}
```

Podemos hacer una versión para aplicar otros critérios de búsqueda, pero no es eficiente.

```
DBNAME=# select id, xpath('/novela/nombre', texto) as textocadena from
tablaXML where xpath('/novela/nombre',texto)::text like '%imperio%' ;
 id |              textocadena
----+-----------------------------------------
  4 | {"<nombre>fundacion e imperio</nombre>"}

DBNAME=# select id, xpath('/novela/nombre', texto) as textocadena from
tablaXML where texto::text like '%imperio%' ;
 id |              textocadena
----+-----------------------------------------
  4 | {"<nombre>fundacion e imperio</nombre>"}
```

Tipos de Datos - Avanzados

JSON

JSON es otro formato muy popular de intercambio de datos, principalmente entre WEB SERVICES (desplazando al XML), básicamente un dato JSON es una arreglo de datos.

```
DBNAME=# create table tablaJSON ( id serial, dato json);
CREATE TABLE

DBNAME=# insert into tablaJSON (dato)  values ( '{"titulo":
"fundacion", "autor": {"nombre": "isaac", "apellido":"asimov"}}');

DBNAME=# insert into tablaJSON (dato)  values ( '{"titulo": "duna",
"autor": {"nombre": "frank", "apellido":"herbert"}}');

DBNAME=# insert into tablaJSON (dato)  values ( '{"titulo":
"farenheit
411", "autor": {"nombre": "arthur", "apellido":"clarke"}}');
```

Tipos de Datos - Avanzados

JSON

JSON es otro formato muy popular de intercambio de datos, principalmente entre WEB SERVICES (desplazando al XML), básicamente un dato JSON es una arreglo de datos.

```
DBNAME=# select * from tablaJSON;
 id |                                          dato
----+-------------------------------------------------------------
  1 | {"titulo": "fundacion", "autor": {"nombre": "isaac",
      "apellido":"asimov"}}
  2 | {"titulo": "duna", "autor": {"nombre": "frank",
      "apellido":"herbert"}}
  3 | {"titulo": "farenheit 411", "autor": {"nombre": "arthur",
      "apellido":"clarke"}}
(3 rows)
```

Tipos de Datos - Avanzados

JSON

```
DBNAME=# select id, dato->'titulo' from tablaJSON;
 id |    ?column?
----+-----------------
  1 | "fundacion"
  2 | "duna"
  3 | "farenheit 411"

DBNAME=# select id, dato->'autor' from tablaJSON;
 id |                 ?column?
----+-----------------------------------------
  1 | {"nombre": "isaac", "apellido":"asimov"}
  2 | {"nombre": "frank", "apellido":"herbert"}
  3 | {"nombre": "arthur", "apellido":"clarke"}

DBNAME=# select id, dato->'autor'->'nombre' from tablaJSON;
 id | ?column?
----+----------
  1 | "isaac"
  2 | "frank"
  3 | "arthur"
```

Tipos de Datos - Avanzados

JSON

```
Los operadores de conversión de datos son:
->  int    Get JSON array element (indexed from zero)
->  text   Get JSON object field by key
->> int    Get JSON array element as text
->> text   Get JSON object field as text
#>  text[] Get JSON object at specified path
#>> text[]  Get JSON object at specified path as text

DBNAME=# select id, dato->>'titulo' from tablajson;
 id |   ?column?
----+--------------
  1 | fundacion

DBNANME=# select id, dato->'titulo' from tablajson;
 id |    ?column?
----+-----------------
  1 | "fundacion"
```

Tipos de Datos - Avanzados

JSON

```
Los operadores de conversión de datos son:
->  int    Get JSON array element (indexed from zero)
->  text   Get JSON object field by key
->> int    Get JSON array element as text
->> text   Get JSON object field as text
#>  text[] Get JSON object at specified path
#>> text[]  Get JSON object at specified path as text
```

```
DBNAME=# select id, dato#>'{autor,nombre}' from tablajson;
 id | ?column?
----+----------
  1 | "isaac"

DBNAME=# select id, dato#>>'{autor,nombre}' from tablajson;
 id | ?column?
----+----------
  1 | isaac
```

Binarios

a)Very Large Objects

PostgreSQL permite guardar binarios dentro de una tabla de muchos gigas de extensión, estos files no se guardan directamente dentro de la tabla creada sino en un ambiente especial y se apunta el oid del file para acceder a este.

```
prueba4=# create table tbl_files( name varchar(100), file oid);

prueba4=# insert into tbl_files values('primer archivo',
lo_import('/home/ernesto/Descargas/Programa.pdf'));
INSERT 0 1
prueba4=# select * from tbl_files;
      name      | file
----------------+-------
 primer archivo | 24455

prueba4=# select lo_export(tbl_files.file,'/tmp/Programa.pdf') from
tbl_files where file = '24455';
 lo_export
-----------
         1
```

Binarios

a)Very Large Objects

Para retirar el binario añadido a la base de datos.

```
prueba4=# update tbl_files set file = null;  <-- borramos el link pero no la
                                                 data
prueba4=# insert into tbl_files values('segundo archivo',
lo_import('/home/ernesto/Descargas/LAN.png'));

prueba4=# select * from tbl_files;
      name       | file
-----------------+-------
 primer archivo  |
 segundo archivo | 24456

prueba4=# select lo_unlink(tbl_files.file) from tbl_files where file = 24456;
 lo_unlink
-----------
         1

prueba4=# update tbl_files set file=
lo_import('/home/ernesto/Descargas/LAN.png') where name = 'segundo archivo';
prueba4=# select * from tbl_files;
      name       | file
-----------------+-------
 primer archivo  |
 segundo archivo | 24457
```

Binarios

b)Bytea

Permite almacenar archivos en base64, estos van no como enlace sino como contenido de la tabla.

1) Creamos un archivo en texto plano con este contenido:

```
bla
ble
bli
blo
blu
```

2) Lo convertimos en base64

```
ernesto@ubuntu:/tmp$ base64 ernesto.txt
YmxhCmJsZQpibGkKYmxvCmJsdQo=
```

3) Lo convertimos pero ahora lo grabamos a un file

```
ernesto@ubuntu:/tmp$ base64 ernesto.txt > ernesto.64
```

Binarios

b)Bytea

Permite almacenar archivos en base64, estos van no como enlace sino como contenido de la tabla.

4) El contenido del file

```
ernesto@ubuntu:/tmp$ cat ernesto.64
YmxhCmJsZQpibGkKYmxvCmJsdQo=
```

5) Lo decodificamos de base64

```
ernesto@ubuntu:/tmp$ base64 -d ernesto.64
        bla
        ble
        bli
        blo
        blu
```

Binarios

b)Bytea

```
DBNAME=# create table archivos (id serial, nombre varchar, contenido
bytea);
CREATE TABLE

DBNAME=# insert into archivos(nombre,contenido) values
('ernesto.txt','YmxhCmJsZQpibGkKYmxvCmJsdQo=');
INSERT 0 1
```

La trama en base64 se ha transformado a arreglo de bytes:

```
DBNAME=# select * from archivos;
 id |   nombre    |                          contenido
----+-------------
+-----------------------------------------------------------
  1 | ernesto.txt |
\x596d7868436d4a735a51706962476b4b596d7876436d4a7364516f3d
(1 row)
```

Binarios

b)Bytea

Convertimos el arreglo de bytes a formato base64:

```
DBNASE=# select encode(contenido,'escape') from archivos;
            encode
------------------------------
 YmxhCmJsZQpibGkKYmxvCmJsdQo=
```

Esta trama puede ser descargada a un archivo convencional y de ahí decodificada a su formato original.

ENCRIPTACION

Las funciones de encriptación son funcionalidad extendida de PostgreSQL, se encuentra presente en la base de datos pero debe ser activada.

En el caso de CentOS es necesario instalar el paquete CONTRIB

```
root@host:yum install postgresql10-contrib
```

a)Verificar si existe la extensión:

```
prueba=# select * from pg_available_extensions where name like '%crypto%';
   name   | default_version | installed_version |         comment
----------+-----------------+-------------------+-------------------------
 pgcrypto | 1.0             |                   | cryptographic functions
```

b)Instalamos la extensión:

```
prueba=# create extension pgcrypto;
CREATE EXTENSION
```

c)Volvemos a verificar el estado:

```
prueba=# select * from pg_available_extensions where name like '%crypto%';
   name   | default_version | installed_version |         comment
----------+-----------------+-------------------+-------------------------
 pgcrypto | 1.0             | 1.0               | cryptographic functions
```

ENCRIPTACION

1. Hashes

Este método de encriptación es el más básico, consiste en una generalización de cadenas de texto en otra cadena que suele ser de longitud fija, existen diferentes algoritmos como el md5 o sha256 para esto.

Estos tienen la limitación de ser one-way, osea generan el valor computado pero no se puede dar vuelta atrás para conocer el texto original ingresado.

```
prueba=# select digest('texto a encriptar','md5');
------------------------------------
 \x25c63682db829b79f5cd5348dd3418c8

prueba=# select digest('texto a encriptar','sha1');
-----------------------------------------
 \xa611a30b7a2e9ddef1fbbb20b3c45160689546e5

prueba=# select digest('texto a encriptar','sha256');
-------------------------------------------------------------------
 \x4339fd8970e6fadaeaecf7f26e1a1f09b51d474b84f2a528e4cf39bffd79f53b
```

ENCRIPTACION

1. Hashes

Este método de encriptación es el más básico, consiste en una generalización de cadenas de texto en otra cadena que suele ser de longitud fija, existen diferentes algoritmos como el md5 o sha256 para esto.

Estos tienen la limitación de ser one-way, osea generan el valor computado pero no se puede dar vuelta atrás para conocer el texto original ingresado.

```
prueba=# select digest('texto a encriptar','sha512');
--------------------------------------------------------------------------------
\x19e5c9f859bffdc3068a8d2d39ce3a0e880dd3bdd70cfa74a552c4ecee9e3acf23794c03894dab655cae311a5d267c5ec6c91b494feabe53209d5a9e562eef55
```

El tipo de dato resultante es del tipo BYTEA.

ENCRIPTACION

2. HMAC

Este método de encriptación es parecido a DIGEST, pero permite ingresar una dato “semilla” para generar un MD5 que solo puede ser reconstruido si se conoce la semilla.

```
prueba=# select digest('texto a encriptar','md5');
                digest
-------------------------------------
 \x25c63682db829b79f5cd5348dd3418c8

prueba=# select hmac('texto a encriptar','semilla1', 'md5');
                hmac
-------------------------------------
 \xf59fe00135d193b7640eb9cf874edb2e

prueba=# select hmac('texto a encriptar','semilla2', 'md5');
                hmac
-------------------------------------
 \xd1814d2cfd0dd93645e17896082ae373
```

ENCRIPTACION

3. CRYPT

Diseñado para encriptar password, pero al igual que los hash generados con DIGEST o HMAC no tiene vuelta atrás.

En el primer paso ingresamos nuestro texto a encriptar (el password) y usamos gen_salt() para generar una semilla de encriptación, los algoritmos posibles son MD5, DES, XDES y BF.

```
prueba=# select crypt('texto a encriptar', gen_salt('md5'));
                crypt
-------------------------------------
 $1$JqmCRqOp$XcwbKUfzZnXNGyMLJY4V.0
```

Cuando queremos validar que el texto ingresado corresponde al almacenado volver a ejecutar CRYPTO pero pasamos el texto (password) y el resultado de la primera encriptación, si el valor devuelto es igual al almacenado entonces el texto es correcto.

ENCRIPTACION

3. CRYPT

```
prueba=# select crypt('texto a
encriptar','$1$JqmCRqOp$XcwbKUfzZnXNGyMLJY4V.0');
               crypt
-------------------------------------
 $1$JqmCRqOp$XcwbKUfzZnXNGyMLJY4V.0


prueba=# select crypt('texto a encriptar
xx','$1$JqmCRqOp$XcwbKUfzZnXNGyMLJY4V.0 ');
               crypt
-------------------------------------
 $1$JqmCRqOp$ZCqd5/M74rQPgVjLJqpI01
```

ENCRIPTACION

4. ENCRYPT y DECRYPT

Estas funciones nos permite encriptar y desencriptar un texto, en base a una “semilla” determinada.

a) Encriptamos el texto deseado y definimos una semilla cualquiera, puede ser una palabra o un texto, mientras más larga la semilla más fuerte la encryptación, el resultado final será un BYTEA.

```
prueba=# select encrypt('texto a encryptar','semilla1','bf');
--------------------------------------------------
 \x3e0d090039fbbd7d96177c73eecf112cb6a3d1339eea002a
```

ENCRIPTACION

4. ENCRYPT y DECRYPT

Estas funciones nos permite encriptar y desencriptar un texto, en base a una “semilla” determinada.

b) Desencriptamos el texto, la salida de la función DECRYPT es un BYTEA por lo tanto tenemos que decodificarlo a un tipo de texto que pueda ser entendible.

```
prueba=# select decrypt( encrypt('texto a
encryptar','semilla1','bf'),'semilla1','bf');
----------------------------------------
 \x746578746f20120656e63727970746172

prueba=# select encode(decrypt( encrypt('texto a
encryptar','semilla1','bf'),'semilla1','bf'),'escape');
------------------
 texto a encryptar
```

'BF' es la abreviación del algoritmo Blowfish, también puede usarse AES, este devuelve una cadena encriptada mucho más grande y más difícil de romper.

ENCRIPTACION

5.PGP_SYM_ ENCRYPT y PGP_SYM_DECRYPT

Son funciones muy parecidas a las de su predecesoras, siguen el mismo concepto.

```
prueba=# select pgp_sym_encrypt('texto a encriptar','semilla');
--------------------------------------------------------------------
--------------------------------------------------------------------
-----------------------------

\xc30d0407030202de3edff3fb74446bd242010c9a467629db5aedc546d5069ba83e31
ea708b2041a4149abe67845d62fb3609c711a5dcb15a6496e47ea328f31387634fbfe1
892e1db1564092efad4e0c67b384
```

Como se puede ver PGP_SYM_ENCRYPT devuelve una trama BYTEA, ahora desencriptamos:

ENCRIPTACION

5.PGP_SYM_ ENCRYPT y PGP_SYM_DECRYPT

```
prueba=# select pgp_sym_decrypt(pgp_sym_encrypt('texto a
encriptar','semilla'),'semilla');
  pgp_sym_decrypt
--------------------
 texto a encriptar
(1 row)
```

Las funciones llevan el texto “SYM” porque es una encriptación SIMÉTRICA, es decir la misma “semilla” que encripta es la que desencripta.

ENCRIPTACION

6.Encriptación Asimétrica

Es el más seguro de los métodos, consiste en tener una “semilla” en 2 partes, una nos permite encriptar y otra desencriptar, más una frase secreta para realizar esta operación.
Antes que nada necesitamos generar un par de juegos de claves.

```
ernesto@host:~$ gpg --gen-key
gpg (GnuPG) 1.4.16; Copyright (C) 2013 Free Software Foundation, Inc.

Por favor seleccione tipo de clave deseado:
   (1) RSA y RSA (predeterminado)
    ...
¿Su selección?: 1
¿De qué tamaño quiere la clave? (2048) 1024
```

ENCRIPTACION

6.Encriptación Asimétrica

```
Por favor, especifique el período de validez de la clave.
         0 = la clave nunca caduca
¿Validez de la clave (0)? 0
La clave nunca caduca ¿Es correcto? (s/n) s

Necesita un identificador de usuario para identificar su clave. El programa ...
Nombre y apellidos: Clave Hospital
Dirección de correo electrónico: clinica@santaisabel.com
Comentario: clave general de la clinica
Ha seleccionado este ID de usuario:
    «Clave Hospital (clave general de la clinica) <clinica@santaisabel.com>»

¿Cambia (N)ombre, (C)omentario, (D)irección o (V)ale/(S)alir? v
Necesita una frase contraseña para proteger su clave secreta.

Es necesario generar muchos bytes aleatorios. Es una buena idea realizar
...

gpg: clave A27ED249 marcada como de confianza absoluta
claves pública y secreta creadas y firmadas.
```

ENCRIPTACION

6.Encriptación Asimétrica

```
gpg: clave A27ED249 marcada como de confianza absoluta
claves pública y secreta creadas y firmadas.

gpg: comprobando base de datos de confianza
gpg: 3 dudosa(s) necesarias, 1 completa(s) necesarias,
modelo de confianza PGP
gpg: nivel: 0  validez:   2  firmada:   0  confianza: 0-, 0q, 0n, 0m, 0f, 2u
pub   1024R/A27ED249 2015-03-19
      Huella de clave = 7C46 9438 46A0 40C1 53A0  216C 6ABA 0BBF A27E D249
uid                  Clave Hospital (clave general de la clinica)
<clinica@santaisabel.com>
sub   1024R/861CC1CD 2015-03-19
```

ENCRIPTACION

6.Encriptación Asimétrica

Verificamos si el proceso se ejecutó bien.

```
ernesto@ubuntu:~$ gpg --list-secret-keys
/home/ernesto/.gnupg/secring.gpg
--------------------------------
sec   1024R/9352E801 2015-03-19
uid                  ernesto quiñones (el jefe) <ernestoq@gmail.com>
ssb   1024R/E8FE3F69 2015-03-19

sec   1024R/A27ED249 2015-03-19
uid                  Clave Hospital (clave general de la clinica) <clinica@santaisabel.com>
ssb   1024R/861CC1CD 2015-03-19
```

Exportamos las claves generadas.

```
ernesto@ubuntu:~$ gpg -a --export Clave Hospital > public.key
ernesto@ubuntu:~$ gpg -a --export-secret-keys Clave Hospital > secret.key
ernesto@ubuntu:~$ ls -la
drwxr-xr-x 20 ernesto ernesto 4096 mar 19 16:17 .
drwxr-xr-x  3 root    root    4096 mar  4 15:06 ..
-rw-rw-r--  1 ernesto ernesto 1052 mar 19 16:17 public.key
-rw-rw-r--  1 ernesto ernesto 2058 mar 19 16:17 secret.key
```

ENCRIPTACION

6.Encriptación Asimétrica

Ahora podemos utilizar las claves

```
create or replace function encripta() returns integer
as $$
declare
    public text;              <-- para almacenas la llave publica
    secret text;              <-- para almacenar la llave secreta
    texto text := 'texto a encriptar';  <-- texto a encriptar
    dtexto text;              <-- para almacenar el texto desencriptado
    etexto bytea;             <-- para almacenar el texto encriptado
begin

    public:='-----BEGIN PGP PUBLIC KEY BLOCK-----
      Version: GnuPG v1
      mI0EVQs7wgEEALBJCJK739wstPrbC03h0v9U2YnjLmK/uQJrw44WXNFS2FMUPNbb
      -----TODA LA LLAVE PUBLICA AQUI
      =Z3kj
      -----END PGP PUBLIC KEY BLOCK-----';
```

ENCRIPTACION

6.Encriptación Asimétrica

Ahora podemos utilizar las claves

```
    secret:='-----BEGIN PGP PRIVATE KEY BLOCK-----
      Version: GnuPG v1
      lQH9BFULO8IBBACwSQiSu9/cLLT62wtN4dL/VNmJ4y5iv7kCa8OOFlzRUthTFDzW
      -----TODA LA LLAVE SECRETA AQUI
      =rIe0
      -----END PGP PRIVATE KEY BLOCK-----';

    etexto := pgp_pub_encrypt(texto,dearmor(public));  <-- encriptamos
    raise notice 'texto encriptado: %' ,etexto;
    raise notice '';  raise notice '';
    dtexto := pgp_pub_decrypt(etexto,dearmor(secret),'apesol');  <--
                                                          desencriptamos
    raise notice 'texto desencriptado: %' ,dtexto;
    return 1;
end
$$ language plpgsql;
```