FUNCIONES DEFINIDAS POR EL USUARIO (FDU)

1.- Definición: Es un bloque de instrucciones que se guardan en el servidor con un nombre asignado por el usuario. Las funciones pueden recibir N datos (parámetros) y retorna un valor al punto de invocación.

2.- Ventajas:

- La función se almacena dentro de la BD; es decir, se convierte en otro elemento de la BD como lo son: las tablas, las relaciones, los registros (Tuplas), las vistas, entre otros.
- La ejecución de una función ocurre del lado del servidor de la BD. Esto aumenta el rendimiento de nuestra aplicación; pues al ejecutar los procesos del lado del servidor, se le quita al FrontEnd la responsabilidad de realizar dichos cálculos.
- Ahorra líneas de código del lado del aplicativo (FrontEnd - Cliente); pues al tener la lógica de la aplicación implementada dentro de la base de datos se evita re-escribir constantemente dicho código en los aplicativos del cliente. Por consiguiente, se ahorran líneas de código, se evita la redundancia y se disminuye la complejidad del código fuente del FrontEnd.
- Permite la comunicación más fácil de la BD con los diversos tipos de clientes, sistemas y/o lenguajes de programación que use o implemente la BD, pues al tener la lógica estructural embebida dentro de la BD, no tendremos que programar la misma lógica para todos los usuarios y aplicativos; es más; se pueden programar APIs o micro-servicios listos para ser invocados desde diferentes clientes.
- Cualquier cosa corre más rápido en el servidor (BackEnd) que en el entorno del cliente (FrontEnd).

Nota: Lo primero que tenemos que hacer es verificar e instalar el lenguaje plpgsql en el servidor de la BD.

=#**SELECT** version() **;** // ---> Muestra la versión instalada del plpgsql

=#**CREATE PROCEDURAL LANGUAGE** plpgsql **;** //---> Instala el lenguaje plpgsql

3.- Comandos Básicos

A.- Creación:

CREATE OR REPLACE FUNCTION FNombre(Par1 **Tipo ,** … **,** ParN **Tipo**) **RETURNS TIPO AS** $$
[**DECLARE** - declaración de variables Locales **OJO OJO SI SON NECESARIAS**
Variable_Local **Tipo_Variable** := Valor_Inicial **;**]
BEGIN

Código **;**
*
*

RETURN Variable_Local **;**

END ;
$$ **LANGUAGE** plpgsql

B.- Ejecución: =# **SELECT** FNombre(Par1 **,** .... **,** ParN) **;**

C.- Listado:

=#\x [on / off] ----------> Activa o desactiva el despliegue de los parámetros
En formato fila / Columna de las funciones almacenadas.
=#\df ----------> Muestra el listado de todas las funciones almacenadas.
=#\df+ ----------> Muestra el listado de las funciones **con los parámetros extendidos**.
=#\df nombre_función -------> Muestra el listado **de los parámetros** de la función.
=#\df+ nombre_función ------> Muestra el listado **de los parámetros extendidos** de la función.
=#\sf+ nombre_función ------> Muestra el código fuente de la función.

4.- Ejemplos de FDU (FUNCIONES DEFINIDAS POR EL USUARIO)

-- Comentario de una línea /* Comentario de varias líneas */

Ejemplo 01: Parámetros

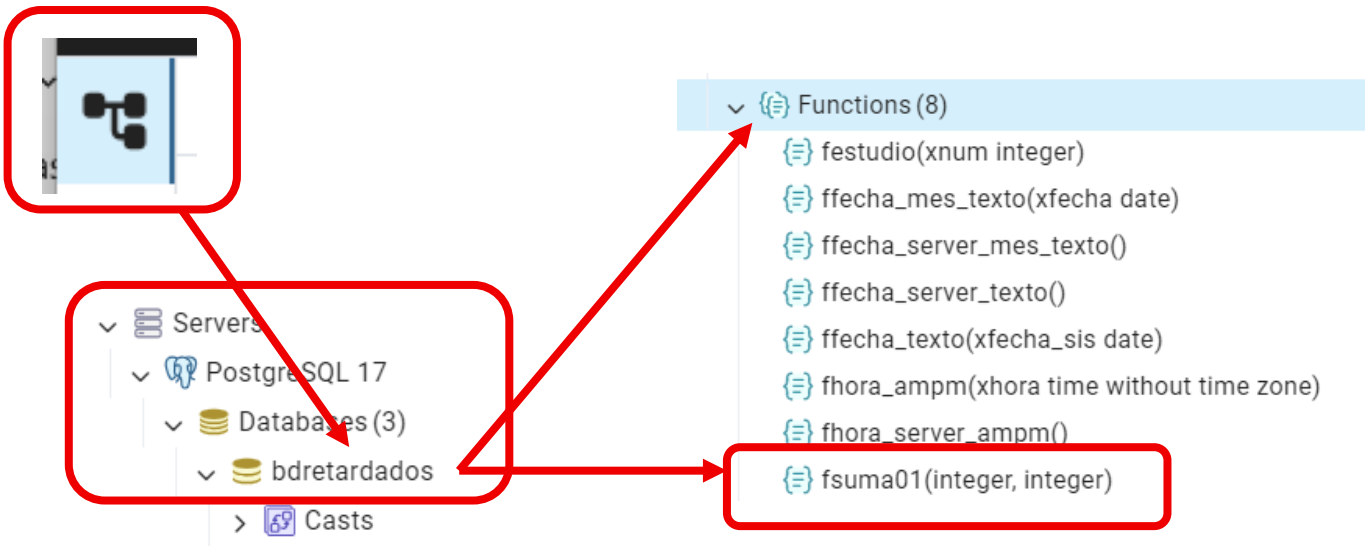
```
CREATE OR REPLACE FUNCTION fsuma01( integer, integer ) RETURNS integer AS $$
-- fsuma01() -----> Función para sumar
-- Autor: Sergio Capacho
-- Fecha: Lunes 30/Marzo/2026 – 03:40 Pm
BEGIN
    RETURN $1 + $2 ;
END ;
$$ LANGUAGE plpgsql ;
```

.

=#\df ----------> Muestra el listado de todas las funciones almacenadas.
=#\df **fsuma** -------> Muestra el listado **de los parámetros** de la función.
=#\sf+ **fsuma** ------> Muestra el código fuente de la función.

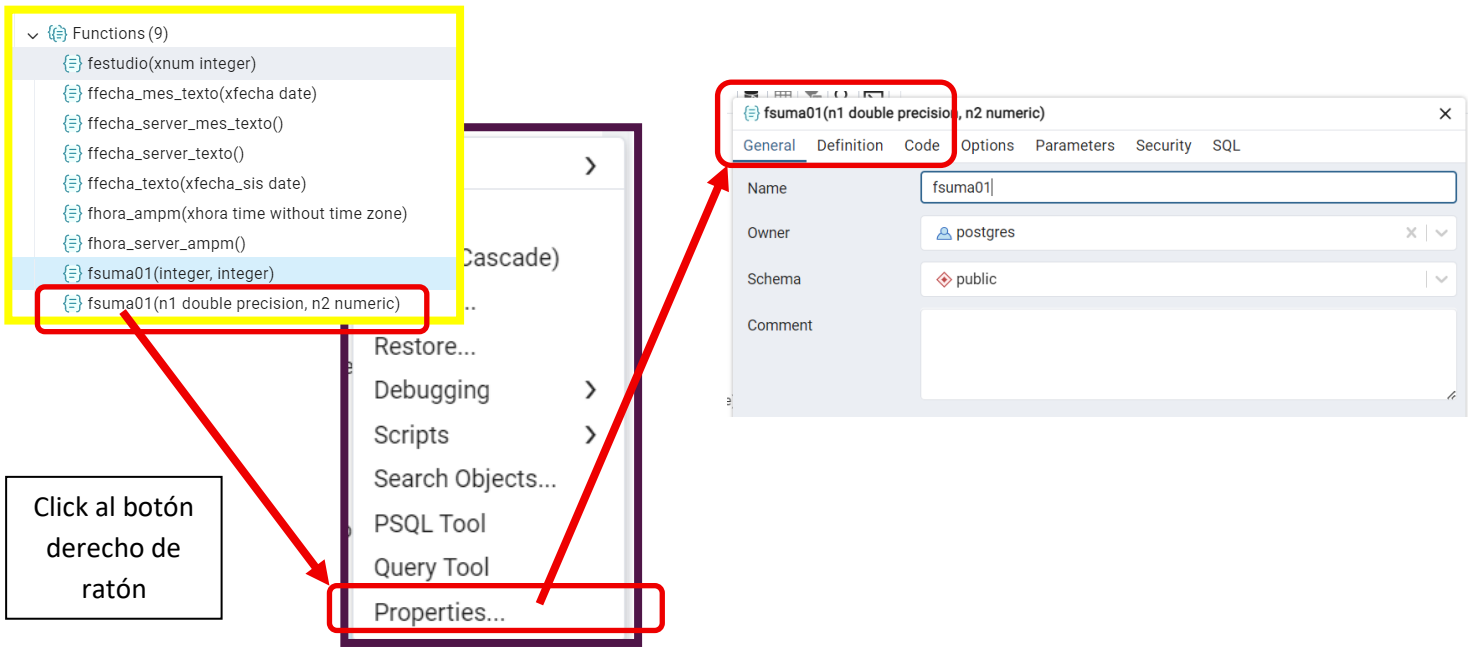
.

```
=# SELECT fsuma01(4,6) ; ----> 10
```



Ejemplo 02: Sobre carga de funciones - Parámetros

```
CREATE  OR  REPLACE  FUNCTION  fsuma01(n1  float,   n2  numeric(7, 2)  )
                                RETURNS  decimal(7,2)  AS  $$
-- fsuma01() --- > Ejemplo de sobre carga de la Función fsuma01()
-- float  --→  Doble precisión  --→  numeric --→  decimal
-- Autor: Sergio Capacho
-- Fecha: Lunes 30/Marzo/2026 – 04:08 Pm
BEGIN
      RETURN  n1 + n2  +  3.5 ;
END  ;
$$ LANGUAGE  plpgsql  ;
.
=# SELECT  'Respuesta: '  ||  to_char(fsuma01(4,  6.4), '99D99 ')  ;  ----> Respuesta:  13,90
```



Ejemplo 03: Declaración de Variables dinámicas ($1 ---→ n1 ; $2 --→ n2)

```
CREATE OR REPLACE FUNCTION fmul(integer , integer) RETURNS integer AS $$
/*fmul() -→ Función para multiplicar n1 por n2 y sumar 100
   Autor: Sergio Capacho
   Fecha: Lunes 30/Marzo/2026 04:18 Pm
*/
DECLARE
        n1 ALIAS FOR $1 ;
        n2 ALIAS FOR $2 ;
        xconstante CONSTANT integer := 100 ;
        respuesta integer ;
BEGIN
        Respuesta := (n1 * n2) + xconstante ;

        RETURN respuesta ;
END ;
$$ LANGUAGE plpgsql ;
.
.
.
=# SELECT 'Respuesta: ' || to_char(fmul(4, 2), '999G999D99 ') ; ----> Respuesta: 108.00
```

Ejemplo 04: Area_Circulo = Pi * (Radio ^ 2)

```
CREATE OR REPLACE FUNCTION farea_circulo(radio numeric(7,2) )
                                    RETURNS numeric(7,2) AS $$
-- farea_circulo(); ---> Función para calcular el área de un circulo
-- Autor: Sergio Capacho
-- Fecha: Lunes 30/Marzo/2026 – 04:38 Pm
DECLARE
        pi      float  :=  3.14  ;
        area   float  := 0.0 ;
BEGIN
        area  := pi * POWER(radio , 2)  ;    /* ---> Elevar el Valor ^ N */
        RETURN  area  ;
END  ;
$$ LANGUAGE   plpgsql   ;
.
.
.
=# SELECT  'El área del circulo es: ' ||   TO_CHAR(farea_circulo(15.5),  '999G999G999D99 ')  ;
El área del circulo es:  ----> 754.39
```

Ejemplo 05: Decisiones IF...

```
CREATE OR REPLACE FUNCTION festudio01(x decimal(10,2) ) RETURNS text AS $$
-- festudio01(); ---> Función para determinar si X es negativo o positivo
-- Autor: Sergio Capacho
-- Fecha: Lunes 30/Marzo/2026– 04:51 Pm
DECLARE
        respuesta text DEFAULT 'Error en el cálculo ...' ;
BEGIN
        IF x < 0.0 THEN
                respuesta := 'El valor es negativo...' ;
        ELSE
                respuesta := 'El valor es positivo...' ;
        END IF ;

        RETURN respuesta ;
END ;
$$ LANGUAGE plpgsql ;
.
.
=# SELECT 'Respuesta: ' || festudio01(15.5) ; ----> Respuesta: El valor es positivo...
=# SELECT 'Respuesta: ' || festudio01(-4.5) ; ----> Respuesta: El valor es negativo...
```

Ejemplo 06: Decisiones IF...

```
CREATE OR REPLACE FUNCTION fresta( a decimal(10,2) , b numeric(10,2) )
                                  RETURNS text AS $$
-- fresta(); ---> Función para mostrar solo restas negativas
-- Autor: Sergio Capacho
-- Fecha: Lunes 30/Marzo/2026 – 04:58 Pm
DECLARE
        resta       float := 0.0 ;
        mensaje  text DEFAULT 'Error...' ;
BEGIN
        resta := a - b ;
        IF resta < 0.0 THEN
                mensaje := resta ;
        ELSE
                mensaje := 'Existe un Error en el cálculo...' ;
        END IF ;
        RETURN mensaje ;
END ;
$$ LANGUAGE plpgsql ;
.
.
.
=# SELECT 'La resta es: ' || fresta(08.2, 10.8) ; ----> La resta es: -2.6...
=# SELECT 'La resta es: ' || fresta(10.8, 08.2) ; ----> La resta es: Error en el cálculo...
```

Ejemplo 07: Decisiones Anidadas.. Área de la Figura ----> cuadrado / circulo

```
CREATE OR REPLACE FUNCTION  farea_figura( figura  varchar , valor  numeric(10,2) )
                                    RETURNS  character varying  AS  $$
-- farea_figura() --- > Función para mostrar el Área de una Figura según texto: cuadrado / circulo
-- Autor: Sergio Capacho   /   Fecha: Lunes  30/Marzo/2026 – 05:07 Pm
--  varchar --->  character varying  --> text
DECLARE
        area        decimal(10,2)   := 0.0 ;
        mensaje   text   DEFAULT   'Error en el cálculo ...'   ;
BEGIN
        figura := lower(figura)  ;
        IF figura =  'cuadrado' THEN
                area := valor * valor  ;
                mensaje := area  ;
        ELSE
                IF  figura = 'circulo'   THEN
                        area = 3.14 * valor * valor  ;
                        mensaje := area  ;
                ELSE
                        mensaje := 'Error en la figura...'  ;
                END  IF ;
        END IF ;
        RETURN 'El Área de la figura ' || figura || ' es:  ' || mensaje  ;
END  ;
$$ LANGUAGE    plpgsql   ;
.
=# SELECT   farea_figura('cuadrado' ,   2.4)  ;  ----> 5.76
=# SELECT   farea_figura('circulo' ,   10.8)  ;  ----> 366.25
=# SELECT   farea_figura('otra' ,   10.8)  ;  -------> Error en la figura...
```

Ejemplo 08: Decisiones anidadas

```
CREATE OR REPLACE FUNCTION fsueldo01(xnombre text , xhoras integer , xcargo char)
                                        RETURNS integer AS $$
-- Función para calcular el Sueldo Neto de un empleado
-- Autor: Sergio Capacho   /   Fecha: Lunes30/Marzo/2026 – 05:21 Pm
DECLARE
        sb    float          := 0.0  ;
        bo    numeric(7,2)   := 0.0  ;
        de    decimal(7,2)   := 0.0  ;
        sn    float          := 0.0  ;
BEGIN
        IF   xcargo ='A'   THEN
                sb = 100.0 * xhoras  ;
                bo = sb * 0.05  ;
                de = sb * 0.02  ;
        ELSE
                IF   xcargo = 'B'   THEN
                        sb = 200.0 * xhoras  ;
                        bo = sb * 0.07  ;
                        de = sb * 0.03  ;
                ELSE
                        sb = 300.0 * xhoras  ;
                        bo = sb * 0.09  ;
                        de = sb * 0.04  ;
                END IF  ;
        END IF  ;
        sn:= sb + bo -de  ;
        RAISE NOTICE '% tiene el sueldo base: % ,  con el Bono:% ,
                        La Deducción:%  y Sueldo Neto de:%',  xnombre,   sb,   bo,   de,   sn  ;
        RETURN   0  ;
END  ;
$$ LANGUAGE   plpgsql  ;
.
=# SELECT   fsueldo01('Sergio',10,'A')  ;  ---->
NOTICE:  Sergio tiene el sueldo base: 1000 , con el Bono:50.00 , la Deducción:20   y el Sueldo
Neto:1030
```

Ejemplo 09: Decisiones Múltiples...

```
CASE variable (Entera / Char)
        WHEN   valor   THEN

                codigo... ;

        WHEN   valor   THEN

                codigo... ;

        WHEN   valor   THEN

                codigo... ;

        ELSE

                codigo... ;


END CASE  ;
```

```
CREATE   OR   REPLACE   FUNCTION   fsueldo02(xnombre text ,  xhoras integer ,  xcargo char)
                                        RETURNS   integer    AS    $$
-- Función para calcular el Sueldo Neto de un empleado
-- Autor: Sergio Capacho
-- Fecha: Lunes 30/Marzo/2026  – 05:35 Pm
DECLARE
        sb   float              := 0.0 ;
        bo   numeric(7,2)   := 0.0  ;
        de   float              := 0.0 ;
        sn   decimal(7,2)    :=0.0  ;
        sw   integer            := 1  ;
```

```
BEGIN
        CASE   UPPER( xcargo )
                WHEN   'A'   THEN
                        sb = 100.0 * xhoras   ;
                        bo = sb * 0.05   ;
                        de = sb * 0.02   ;
                WHEN   'B'   THEN
                        sb = 200.0 * xhoras   ;
                        bo = sb * 0.07   ;
                        de = sb * 0.03   ;
                WHEN   'C'   THEN
                        sb = 300.0 * xhoras   ;
                        bo = sb * 0.09   ;
                        de = sb * 0.04   ;
                ELSE
                        sw:= 0   ;
        END CASE ;

        IF   sw   =   1   THEN
                sn:= sb + bo -de   ;
                RAISE NOTICE '% tiene el sueldo base: %,   con el Bono:%,   la Deducción:%
                                        y el Sueldo Neto de :%', xnombre,sb,bo,de,sn   ;
        ELSE
                RAISE NOTICE 'Calculo imposible …..'   ;
        END IF   ;

        RETURN 0   ;
END   ;
$$ LANGUAGE   plpgsql   ;
.
.
=# SELECT   fsueldo02('Sergio',10,'A')   ;   ---->
NOTICE:  Sergio tiene el sueldo base: 1000,   el Bono:50.00,   la Deducción:20
  y Sueldo Neto:1030.00
```

Ejemplo 10: Ciclo For

```
FOR  variable  IN  [reverese]  Punto_Inicial  ..  Punto_Final  [BY]  Incremento
LOOP
        .
END LOOP  ;

CREATE  OR  REPLACE  FUNCTION freporte_fibonacci(  tx  integer )  RETURNS  integer  AS $$
-- Función para mostrar los  N primeros términos (tx) de la serie de Fibonacci
-- Autor: Sergio Capacho   /   Fecha: Lunes 30/Marzo/2026 – 05:50 Pm
DECLARE
        a  integer := 0  ;
        b  integer := 1  ;
        c  integer := 0  ;
        i  integer := 3  ;
BEGIN
        IF  tx  =  1  THEN
                RAISE NOTICE  'Termino 1  Valor  ---> %', a  ;
        ELSE
                IF  tx  >=  2  THEN
                        RAISE NOTICE 'Termino 1  Valor ---> %', a  ;
                        RAISE NOTICE 'Termino 2  Valor ---> %', b  ;
                        c := a + b  ;

                        FOR    i   IN    3  ..  tx    BY    +1
                        LOOP
                                RAISE NOTICE 'Termino %  Valor ---> % ',i  ,c  ;
                                a:= b  ;
                                b:= c  ;
                                c := a + b  ;
                        END LOOP  ;
                END IF  ;
        END IF  ;
        RETURN 0  ;
END  ;
$$ LANGUAGE   plpgsql  ;
.
```

.
.

```
=# SELECT   freporte_fibonacci(7)  ;
NOTICE:  Termino 1  Valor ---> 0
NOTICE:  Termino 2  Valor ---> 1
NOTICE:  Termino 3  Valor ---> 1
NOTICE:  Termino 4  Valor ---> 2
NOTICE:  Termino 5  Valor ---> 3
NOTICE:  Termino 6  Valor ---> 5
NOTICE:  Termino 7  Valor ---> 8
freporte_fibonacci
--------------------
                 0
```

Ejemplo 11: ---> **Ciclo While**

```
WHILE   <Condición>
LOOP
.
.
.
END LOOP  ;
```

```
CREATE OR REPLACE FUNCTION fvalor_fibonacci( tx integer ) RETURNS integer AS $$
-- Función para retornar el valor asociado al número del termino X de la serie de Fibonacci
-- Autor: Sergio Capacho
-- Fecha: Lunes 30/Marzo/2026 – 06:06 Pm
DECLARE
        a integer := 0 ;
        b integer := 1 ;
        c integer := 0 ;
        i integer := 2 ;
BEGIN
        IF tx = 1 THEN
                RAISE NOTICE 'Termino 1 Valor ---> %', a ;
        ELSE
                IF tx = 2 THEN
                        RAISE NOTICE 'Termino 2 Valor ---> %', b ;
                ELSE
                        WHILE ( i < tx )
                        LOOP
                                c := a + b ;
                                a:= b ;
                                b:= c ;
                                i = i + 1 ;
                        END LOOP ;
                        RAISE NOTICE 'Termino % Valor ---> %', i, c ;
                END IF ;
        END IF ;
        RETURN 0 ;
END ;
$$ LANGUAGE plpgsql ;
.
.

=# SELECT freporte_fibonacci(7) ;
=# SELECT fvalor_fibonacci(7) ; ----> 8
```

Ejemplo 12: ----> **Ciclo Repita Hasta** (LOOP ... END LOOP)

```
LOOP
        ...
        .......
        ............
        IF  contador  >  0  THEN
                EXIT  ;  -- exit loop
        END IF  ;

END LOOP ;


CREATE  OR  REPLACE  FUNCTION  ftabla( nx  integer  )  RETURNS  integer  AS $$
/* Función para obtener la tabla de multiplicación del número nx
   Autor: Sergio Capacho
   Fecha: Lunes 30/Marzo/2026  – 06:15 Pm  */
DECLARE
        i       integer:= 0  ;
        m       integer:= 0  ;
BEGIN
        RAISE NOTICE 'Tabla de Multiplicar del ----> %',  nx  ;
        LOOP
                m := nx * i  ;
                RAISE NOTICE '% X % ---> %',  nx,  i,  m  ;
                i = i + 1  ;

                IF  i  >  10  THEN
                        EXIT  ;  -- exit loop
                END IF  ;
        END LOOP  ;
        RETURN  1  ;
END  ;
$$ LANGUAGE   plpgsql  ;
.
=# SELECT  ftabla(6)  ;
```

```
NOTICE:  Tabla de Multiplicar del ----> 6
NOTICE:  6 X 0 ---> 0
NOTICE:  6 X 1 ---> 6
NOTICE:  6 X 2 ---> 12
NOTICE:  6 X 3 ---> 18
NOTICE:  6 X 4 ---> 24
NOTICE:  6 X 5 ---> 30
NOTICE:  6 X 6 ---> 36
NOTICE:  6 X 7 ---> 42
NOTICE:  6 X 8 ---> 48
NOTICE:  6 X 9 ---> 54
NOTICE:  6 X 10 ---> 60
 ftabla_multiplicar
--------------------
                  1
(1 fila)
```

Ejemplo 13: ---> La cantidad de caracteres del texto / Longitud texto

```
CREATE OR REPLACE FUNCTION flongitud_texto(xt text ) RETURNS integer AS $$
-- Función para determinar la cantidad de caracteres de un texto
-- Autor: Sergio Capacho
-- Fecha: Lunes 30/Marzo/2026 –  06:21 Pm
DECLARE
        xtexto      ALIAS  FOR  $1  ;
        longitud   integer := 0  ;
BEGIN
        longitud := (select length(xtexto))  ;
        RETURN  longitud  ;
END  ;
$$ LANGUAGE plpgsql  ;
.
.
=# SELECT   flongitud_texto('hola mundo')  ;  ----> 10
```

Ejemplo 14: ---> **Concatenar nombres.** Varchar -----→ text

```
CREATE OR REPLACE FUNCTION fconcatenar(xnom varchar , xape text)
                          RETURNS VARCHAR AS $$
-- Autor: Sergio Capacho
-- Fecha: Lunes 30/Marzo/2026 – 06:27 Pm
DECLARE
        xtemp varchar ;
BEGIN
        xtemp:= xnom || '  ' || xape ; -- || (Se usa para concatenar texto)
        RETURN initcap(xtemp) ; -- initcap(texto) Convierte la primera letra
                                --de cada palabra a mayúsculas
END ;
$$ LANGUAGE plpgsql ;
.
=# SELECT fconcatenar('sergio', 'capacho') ; ----> Sergio Capacho
```

OJO PROBAR ESTE CAMBIO Y ANALIZARLO

```
CREATE OR REPLACE FUNCTION fconcatenar(xnom varchar , xape text)
                          RETURNS TEXT AS $$
-- Autor: Sergio Capacho
-- Fecha: Lunes 30/Marzo/2026 – 06:27 Pm
DECLARE
        xtemp varchar ;
BEGIN
        xtemp:= xnom || '  ' || xape ; -- || (Se usa para concatenar texto)
        RETURN initcap(xtemp) ; -- initcap(texto) Convierte la primera letra
                                --de cada palabra a mayúsculas
END ;
$$ LANGUAGE plpgsql ;
```

Ejemplo 15: ---> **Determinar si un valor entero x positivo es Deficiente, Perfecto o Abundante**

```
CREATE OR REPLACE FUNCTION festudio02(xnum integer)  RETURNS  TEXT  AS $$
DECLARE
      xsuma  integer := 0 ;          xsw    integer := 0 ;
      i      integer := 0 ;          texto  text := 'Mensaje de retorno....' ;
BEGIN
      IF xnum <= 0  THEN
            Texto := 'Calculo imposible.....Valor Negativo...'  ;
      ELSE
            FOR  i  IN  1 .. (xnum-1)  BY +1
            LOOP
                  IF mod ( xnum , i ) = 0  THEN
                        xsuma :=  xsuma  + i  ;
                        RAISE NOTICE 'Divisor ---> %  Suma ---> %',  i,  xsuma  ;
                  END IF  ;
            END LOOP  ;
            IF xsuma < xnum THEN
                  Texto  := 'El valor ' || xnum :: text || ' es Definiente.'  ;
            ELSE
                  IF xsuma = xnum THEN
                        texto :=  'El valor ' || xnum :: text || ' es Perfecto.'  ;
                  ELSE
                        texto :=  'El valor ' || CAST(xnum AS text) || ' es Abundante.'  ;
                  END IF  ;
            END IF  ;
      END IF  ;
      RETURN  texto  ;
END  ;
$$ LANGUAGE  plpgsql  ;
```

```
=# SELECT   festudio02(10) ;
=# SELECT   festudio02(6)  ;
=# SELECT   festudio02(12) ;
```

5.- Ejemplos de funciones con tablas (Select …from….Where ….)

Ejemplo 01: ---> Función para **Insertar** Registros en TMEmpleados / **RETURNS VOID**

```
CREATE OR REPLACE FUNCTION
finsertar_empleado01(xcc varchar(12) ,  xnom varchar(40) ,  xfecha date ,  xfkcodcar integer)
RETURNS  VOID  AS  $$
-- Autor: Sergio Capacho
-- Fecha: Domingo 05/Abril/2026 – 06:08 Pm

-- OJO En este caso No hay DECLARE
BEGIN
        INSERT INTO  tmempleados(pkcedemple, nomemple, fecha, fkcodcar)
                        VALUES (xcc, xnom, xfecha, xfkcodcar)  ;
END  ;
$$ LANGUAGE  plpgsql  ;
.
=# SELECT  * from tmempleados  ;
=# SELECT  finsertar_empleado01('9021', 'Greisy', '2024-11-22', 1)  ;
=# SELECT  * from tmempleados  ;
```

Ejemplo 02: ---> Función para **Insertar** Registros en TMCargos / **RETURNS VOID**

```
CREATE OR REPLACE FUNCTION  finsertar_cargos01( varchar(40) ,  numeric(10,2)  )
RETURNS  VOID  AS  $$
-- OJO En este  caso No hay DECLARE
BEGIN
        INSERT INTO tmcargos(dcargo,  sueldo,  fkcods)  VALUES  ($1 ,  $2 ,  1 )  ;
END  ;
$$ LANGUAGE  plpgsql  ;
.
=# SELECT  * from tmcargos  ;
=# SELECT  finsertar_cargos01('Programador Junior', 1200000.01)  ;
=# SELECT  pkcodcar, dcargo, to_char(sueldo, '999G999G999D99')  from  tmcargos  ;
```

Ejemplo 03: ---> **Insertar** datos en la TMCARGOS / SQL SELECT / **RETURNS VOID**

```
CREATE OR REPLACE FUNCTION finsertar_cargos02(xdcargo varchar , xsueldo numeric(12,2) )
RETURNS    VOID    AS  $$
-- OJO En este  caso No hay DECLARE
BEGIN
      INSERT  INTO  tmcargos(dcargo , sueldo ,  fkcods) values (xdcargo , xsueldo ,  1 ) ;
END ;
$$ LANGUAGE plpgsql ;
.
=# SELECT  pkcodcar , dcargo , to_char(sueldo , '999G999G999D99')  from  tmcargos ;
=# SELECT  finsertar_cargos02('Programador Semi Senior' , 1800000.02) ;
=# SELECT  pkcodcar , dcargo , to_char(sueldo , '999G999G999D99')  from  tmcargos ;
```

Ejemplo 04: ---> **Insertar** datos en la TMCARGOS / SQL SELECT / **RETURNS VOID**

```
CREATE OR REPLACE FUNCTION   finsertar_cargos03(varchar , numeric(12,2) )
RETURNS   VOID   AS   $$
DECLARE    -- OJO AQUÍ  TENEMOS  DECLARACIÓN
      c1  ALIAS  FOR  $1 ;
      c2  ALIAS  FOR  $2 ;
BEGIN
      INSERT  INTO  tmcargos(dcargo ,  sueldo ,  fkcods )  values  (c1 ,  c2 ,  1) ;
END ;
$$ LANGUAGE   plpgsql ;
.
=# SELECT  pkcodcar , dcargo , to_char(sueldo , '999G999G999D99')  from  tmcargos ;
=# SELECT  finsertar_cargos03('Programador  Senior' , 1800000.02) ;
=# SELECT  pkcodcar , dcargo , to_char(sueldo , '999G999G999D99')  from  tmcargos ;
```

Ejemplo 05: ---> **Eliminar** físicamente todos los retardos de los empleados que llegaron tarde en noviembre 2022 / Select / **RETURNS VOID** …

```
CREATE OR REPLACE FUNCTION   feliminar_retardos01()   RETURNS  VOID   AS  $$
-- Función para eliminar físicamente todos los  retardos de los empleados de noviembre 2022
-- Autor: Sergio Capacho
-- Fecha: Domingo 05/Abril/2026 –  06:45 Pm
DECLARE
        xtotal   integer  := 0 ;
BEGIN
        SELECT  COUNT(fecha)   INTO    xtotal
        FROM    tdretardos
        WHERE   fecha   BETWEEN   '2022-11-01'  AND  '2022-11-30'  ;

        UPDATE   tdretardos   SET   fkcods = 0
                  WHERE   fecha   BETWEEN   '2022-11-01'  AND  '2022-11-30'  ;
        DELETE   FROM   tdretardos   WHERE   fkcods = 0  ;

        RAISE NOTICE 'Total de registros eliminados ---> % ',   xtotal  ;
END  ;
$$ LANGUAGE   plpgsql  ;
.
=# SELECT   *   FROM   tdretardos
    WHERE   fecha   BETWEEN  '2022-11-01'   AND   '2022-11-30'
    ORDER BY  fecha  ;
=#  SELECT   feliminar_retardos01()  ;
=# SELECT   *   FROM   tdretardos
   WHERE  fecha  BETWEEN  '2022-11-01'   AND   '2022-11-30'
   ORDER BY  fecha  ;
```

Ejemplo 06: Determinar la cantidad de registros eliminados. Para ello se desea eliminar los retardos desde el 01/Marzo/2022 al 31/Marzo/2022 / RETURNS **integer**

```
CREATE OR REPLACE FUNCTION  feliminar_retardos02()  RETURNS  integer  AS  $$
DECLARE
        Xtotal_retardos   integer := 0  ;
        xeliminados       integer := 0  ;
        xresta            integer := 0  ;
BEGIN
        Xtotal_retardos := (SELECT count(fkcodexcu) FROM  tdretardos)  ;
        xeliminados := (SELECT count(fkcodexcu) FROM  tdretardos
                        WHERE  fecha BETWEEN  '2022-03-01'  AND  '2022-03-31')  ;
        DELETE FROM tdretardos WHERE fecha BETWEEN  '2022-03-01'  AND  '2022-03-31'  ;
        xresta := xtotal_retardos  -  xeliminados  ;
        RETURN  xresta  ;
END  ;
$$ LANGUAGE plpgsql  ;
.
=# SELECT  Count(fkcedemple)  FROM  tdretardos  ;
=# SELECT  Count(fkcedemple)  FROM  tdretardos
            WHERE  fecha  BETWEEN  '2022-03-01'  AND  '2022-03-31' ;
=# SELECT  feliminar_retardos02()  ;
=# SELECT  Count(fkcedemple)  FROM  tdretardos  ;
=# SELECT  Count(fkcedemple)  FROM  tdretardos
            WHERE  fecha  BETWEEN  '2022-03-01'  AND  '2022-03-31' ;
```

Ejemplo 07: Determinar la sumatoria de todos los sueldos / **RETURNS float**

```
CREATE OR REPLACE FUNCTION fsuma_sueldos01() RETURNS float AS $$
DECLARE
        xsuma float := 0.0 ;
BEGIN
        --- Aquí son importantes colocar los paréntesis () para que se pueda ejecutar
        --- el comando SELECT y la función de agregación
        xsuma := ( SELECT sum(sueldo) FROM tmcargos ) ;
        RETURN xsuma ;
END ;
$$ LANGUAGE plpgsql ;
=# SELECT pkcodcar , dcargo , to_char(sueldo , '999G999G999D99') from tmcargos ;
=# SELECT to_char( fsuma_sueldos01(), '999G999G999D99' ) ;
```

Ejemplo 08: Se desea determinar la sumatoria de los sueldos mensuales asignados a un cargos X cuyo nombre es enviado como parámetro / **RETURNS decimal**

```
CREATE OR REPLACE FUNCTION fsuma_sueldos02(xcargo varchar) RETURNS decimal(12,2) AS $$
DECLARE
        xsuma float := 0.0 ;
BEGIN
        xsuma := ( SELECT sum(sueldo) FROM tmcargos WHERE dcargo ILIKE xcargo ) ;
        RETURN xsuma ;
END ;
$$ LANGUAGE plpgsql ;
.
=# SELECT pkcodcar , dcargo , to_char(sueldo , '999G999G999D99') from tmcargos ;
=# SELECT to_char( fsuma_sueldos02('SUPERVISOR') , '999G999G999D99' ) ;
=# SELECT to_char( fsuma_sueldos02('Supervisor') , '999G999G999D99' ) ;
=# SELECT to_char( fsuma_sueldos02('Pro%') , '999G999G999D99' ) ; → Comiencen con Pro
=# SELECT to_char( fsuma_sueldos02('%i%') , '999G999G999D99' ) ; -→ Que tenga la "i"
```

Ejemplo 09: Se desea determinar la cantidad de retardos que tiene una persona X, de la cual solo conocemos el nombre; es decir la función debe recibir el nombre como parámetros / RETURNS integer

```
CREATE OR REPLACE FUNCTION  ftotal_retardos(xnom varchar)  RETURNS  integer  AS  $$
DECLARE
        xtotal   integer := 0 ;
BEGIN

        --- Aqui son importantes los paréntesis () para ejecutar el SELECT
        xtotal := ( SELECT   count(fkcodexcu)   FROM  tdretardos
                              WHERE fkcedemple   =   (SELECT  pkcedemple    FROM  tmempleados
                                                                  WHERE  nomemple ILIKE xnom)  )  ;

        RETURN xtotal  ;
END  ;
$$ LANGUAGE plpgsql  ;
.
.
.
=# SELECT   *   FROM   tmempleados   ;
=# SELECT   count(fkcodexcu)   FROM   tdretardos   WHERE   fkcedemple  =  '6000'     ;
=# SELECT   ftotal_retardos('NATASHA')  ;
=# SELECT   ftotal_retardos('Natasha')  ;
```

Ejemplo 10.A (RETURN QUERY - TABLE): Crear una función que retorne **una tabla temporal**; la cual muestre de los los empleados los siguientes datos: la cédula, el nombre, la fecha de nacimiento, la descripción del cargo y el sueldo / **RETURNS TABLE**

```
CREATE OR REPLACE FUNCTION ftabla_empleados01()
RETURNS
TABLE (ced varchar , nom varchar , fe date , dcar varchar , suel numeric(12,2)) AS $$
BEGIN
    RETURN QUERY (
                    SELECT A.pkcedemple , A.nomemple , A.fecha ,
                                B.dcargo , B.sueldo
                    FROM tmempleados AS A , tmcargos AS B
                    WHERE A.fkcodcar = B.pkcodcar
                ) ;
END ;
$$ LANGUAGE plpgsql ;
.
.
.
=# SELECT ftabla_empleados01() ;
=# SELECT (ftabla_empleados01()).* ;
=# SELECT * FROM ftabla_empleados01() ;
```

Ejemplo 10.B (RETURN QUERY – TABLE):

```
CREATE OR REPLACE FUNCTION  ftabla_empleados02()
RETURNS  TABLE (ced  varchar ,  nom  varchar ,  fe  date ,  dcar  varchar ,  suel  varchar)
LANGUAGE  sql  AS
$$
      SELECT  A.pkcedemple ,   A.nomemple ,   A.fecha ,   B.dcargo ,
                  to_char(  B.sueldo ,  '999G999G999D99')
      FROM tmempleados AS A ,    tmcargos  AS  B
      WHERE  A.fkcodcar  =  B.pkcodcar ;
$$;
.
.
.
=# SELECT  ftabla_empleados02()  ;
=# SELECT  (ftabla_empleados02()).*  ;
=# SELECT  *  FROM  ftabla_empleados02() ;
```

Ejemplo 10.C (RETURN QUERY – TABLE): Construir una función que reciba como parámetros de entrada el código de cargo y el código de estatus; y retorne de los empleados que cumplan la condición los siguientes datos: Número de cédula, nombre, descripción del cargo y descripción del status.

```
CREATE OR REPLACE FUNCTION   ftabla_empleados03(
                                        xcodcar  integer,   xcods integer
                                        )
RETURNS    TABLE (cedula varchar ,   nombre varchar ,     cargo varchar ,    status varchar )   AS
$$
BEGIN
      RETURN QUERY (  SELECT  pkcedemple ,
                              nomemple ,
                              dcargo ,
                              dstatus
                      FROM    tmempleados  AS   E
                      JOIN    tmcargos   ON   pkcodcar   =   fkcodcar
                      JOIN    tmstatus   ON   pkcods  =  E.fkcods
                      WHERE    ( fkcodcar =  xcodcar )  AND  ( E.fkcods =  xcods )
                      ORDER BY   CAST(pkcedemple AS BIGINT)
                  ) ;
END  ;
$$ LANGUAGE plpgsql   ;


=# SELECT   *   FROM   ftabla_empleados01(4,   1  )  ;
```

Ejemplo 10.D (RETURN QUERY – TABLE): Generar un reporte que muestre los retardos de un empleado con el número de cédula X, los siguientes datos: Número de retardo, número de cédula, nombre, la excusa, la fecha y la hora.
Nota: El número de cédula se debe recibir como parámetro.

```
CREATE OR REPLACE FUNCTION   ftabla_retardos01(xcedula   varchar)
RETURNS   TABLE (nr  integer ,    cedula varchar ,   nombre varchar ,   excusa varchar ,
                                  fecha  date ,    hora time)   AS   $$
BEGIN
      RETURN QUERY (SELECT  Izq.pknr ,
                                  Izq.fkcedemple AS cedula ,
                                  Der1.nomemple AS nombre ,
                                  Der2.dexcusa AS excusa ,     Izq.fecha ,     Izq.hora
                        FROM    tdretardos  AS Izq
                        JOIN    tmempleados AS Der1  ON  Izq.fkcedemple = Der1.pkcedemple
                        JOIN    tmexcusas   AS Der2  ON  Izq.fkcodexcu = Der2.pkcodexcu
                        WHERE   Izq.fkcedemple =   xcedula
                        ORDER BY   Izq.fkcedemple,   Izq.fkcodexcu
                        ) ;
END  ;
$$ LANGUAGE plpgsql   ;
.
.
.
=# SELECT   *   FROM   ftabla_retardos01('6000' )  ;
```

Ejemplo 10.E (RETURN QUERY – TABLE): Se desea una función que reciba el nombre o parte de éste; y retorne un listado con: la Cedula, el nombre del empleado y el total de los retardos de los empleados que cumplan con dicha condición

```
CREATE OR REPLACE FUNCTION   ftabla_retardos02(xnom   varchar)
RETURNS   TABLE (cedula   varchar ,   nombre   varchar ,   Total_Retardos   bigint)   AS   $$
BEGIN
      RETURN QUERY (SELECT   B.fkcedemple AS   cedula,
                            A.nomemple AS   nombre,
                            COUNT(B.fkcodexcu)   AS   Total_Retardos
                     FROM  tmempleados   AS   A,   tdretardos   AS   B
                     WHERE ( A.pkcedemple  =  B.fkcedemple ) AND
                            ( B.fkcedemple  IN
                                    (SELECT  C.pkcedemple
                                     FROM  tmempleados AS C
                                     WHERE  C.nomemple   ILIKE   xnom)
                            )
                     GROUP BY  B.fkcedemple,   A.nomemple
                     ORDER BY  B.fkcedemple
                     ) ;
END  ;
$$ LANGUAGE plpgsql  ;

=# SELECT   ftabla_retardos02('Natasha');
=# SELECT   ftabla_retardos02('Natasha');
=# SELECT  *  FROM  ftabla_retardos02('Ma%');
=# SELECT  *  FROM  ftabla_retardos02('%tas%');
=# SELECT  *  FROM  ftabla_retardos02('%i%');
```

Ejemplo 11: (**VARSQL - VOID**) ---> Generar un reporte que muestre los retardos de todos los empleados / SQL SELECT.

```
CREATE OR REPLACE FUNCTION ftabla_retardos03() RETURNS VOID AS $$
-- Autor: Sergio Capacho
-- Fecha: Domingo 05/Abril/2026 – 08:48 Pm
DECLARE
        varsql TEXT ;
        datos RECORD ;
BEGIN
        --- Se ejecuta la consulta y el resultado se guarda en la variable varsql
        varsql:= 'SELECT A.pknr, A.fkcedemple, A.fecha, A.fkcodexcu, B.dexcusa
                  FROM tdretardos AS A, tmexcusas AS B
                  WHERE A.fkcodexcu = B.pkcodexcu AND A.fkcods =1
                  ORDER BY A.fkcedemple, A.fkcodexcu, A.fecha' ;

        -- La variable varsql en la práctica es un arreglo de registros
        FOR datos IN EXECUTE varsql
        LOOP
                RAISE NOTICE 'Fila ---> % % % % %', datos.pknr, datos.fkcedemple,
                                        datos.fkcodexcu, datos.fecha, datos.dexcusa ;
        END LOOP ;
END ;
$$ LANGUAGE plpgsql ;
.
=# SELECT ftabla_retardos03() ; ----> Muestra el listado de los reportes
```

Ejemplo 12: (**VARSQL – VOID**) ---> Generar un reporte con los retardos que tiene un empleado con el número de cédula. El número de cédula es enviado por un parámetro / SQL SELECT.

```
CREATE OR REPLACE FUNCTION ftabla_retardos04(xced varchar) RETURNS VOID AS
$$
-- Fecha: Domingo 05/Abril/2026 – 08:53 Pm
DECLARE
        varsql TEXT ;
        datos RECORD ;
BEGIN
        RAISE NOTICE 'Xced ---> % ', xced ;
        --- Se ejecuta la consulta y el resultado se guarda en la variable varsql
        varsql:= 'SELECT A.pknr, A.fkcedemple, A.fecha, A.fkcodexcu, B.dexcusa
                  FROM tdretardos AS A, tmexcusas AS B
                  WHERE A.fkcodexcu = B.pkcodexcu AND A.fkcods =1 AND fkcedemple = $1
                  ORDER BY A.fkcedemple, A.fkcodexcu, A.fecha' ;
        -- La variable varsql en la práctica es un arreglo de registros
        -- USING permite que la variable dinámica XCED sea reconocida por EXECUTE
        FOR datos IN EXECUTE varsql USING xced
        LOOP
                RAISE NOTICE 'Fila ---> % % % % %', datos.pknr, datos.fkcedemple,
                                datos.fkcodexcu, datos.fecha, datos.dexcusa ;
        END LOOP ;
END ;
$$ LANGUAGE plpgsql ;
```

=# **SELECT** fretardos04('6000') ; ----> Muestra el listado de los retardos del empleado con el número de cédula '6000'

Ejemplo 13: (VARSQL - VOID) ---> Generar un reportes con los retardos de un empleado con un número de cédula A y un código de excusa B; donde los datos son dados a través de parámetros / SQL SELECT.

```
CREATE OR REPLACE FUNCTION  ftabla_retardos05(xced varchar(12),  xcodexcu integer)
                                                    RETURNS  VOID  AS $$
DECLARE
        varsql   TEXT      ;
        datos  RECORD  ;
BEGIN
        RAISE NOTICE 'Xced ---> % | Xcodexcu ---> %', xced, xcodexcu ;
        ---  Se ejecuta la consulta y el resultado se guarda en la variable varsql
        varsql := 'SELECT A.pknr,  A.fkcedemple,  A.fecha,  A.fkcodexcu,  B.dexcusa
                   FROM tdretardos  AS  A,  tmexcusas  AS  B
                   WHERE  ( A.fkcodexcu = B.pkcodexcu )  AND  ( A.fkcods =1 )
                          AND ( fkcedemple = $1  ) AND ( fkcodexcu = $2 )
                   ORDER BY  A.fkcedemple,  A.fkcodexcu,  A.fecha' ;
        -- La variable  varsql  en la práctica es un arreglo de registros
        -- USING permite que la variable dinámica XCED, XCODEXCU sea reconocida por EXECUTE
        FOR  datos  IN  EXECUTE  varsql  USING  xced,  xcodexcu
        LOOP
                RAISE NOTICE 'Fila ---> %  %  %  %  %',  datos.pknr,  datos.fkcedemple,
                                                datos.fkcodexcu,  datos.fecha,  datos.dexcusa  ;
        END LOOP  ;
END  ;
$$ LANGUAGE  plpgsql  ;
.
```

=# **SELECT** ftabla_retardos05('6000', 4) ; ----> Muestra el listado de los retardos del empleado con el número de cédula '6000' donde como código de excusa sea el 4: “Pelie con el Aguila”

Ejemplo 14 (**RETURNS Estructura** / **SETOF** ---→ Conjunto de datos): Generar un reporte usando una **tabla TYPE (Estructura - Cursor)** con los siguientes datos de los retardos:

- El número del retardo
- La cédula
- El nombre
- La excusa
- La fecha
- La hora del retardo /

Paso 1: Crear una estructura de datos base

```
CREATE TYPE estructura AS (
        nr          integer,
        cedula      varchar,
        nombre      varchar,
        excusa      varchar,
        fecha       date,
        hora        time
) ;
```

=# \dT --→ Muestra las estructuras de datos creadas con Create Type

=# \d **estructura** o \d+ **estructura** ---→ Muestra la descripción

Nota: La estructura de datos, creada previamente con CREATE TYPE no almacena datos, solo sirve para guardar datos en forma temporal; es decir, se comporta como un arreglo.

SETOF ---→ Conjunto de datos

Paso 2: Crear la función que ejecuta la consulta.

```
CREATE OR REPLACE FUNCTION festructura01()  RETURNS  SETOF  estructura  AS  $$
DECLARE
        registro        RECORD      ;
        regis_temp      estructura  ;
        i               integer := 0  ;
        cursor_tabla    CURSOR  IS
                        SELECT pknr ,  pkcedemple  AS cedula ,  nomemple  AS nombre ,
                               dexcusa  AS excusa ,  Izq.fecha ,  Izq.hora
                        FROM  tdretardos  AS  Izq
                        JOIN  tmempleados  AS  Der1  ON  Izq.fkcedemple = Der1.pkcedemple
                        JOIN  tmexcusas  AS  Der2  ON  Izq.fkcodexcu = Der2.pkcodexcu
                        ORDER BY Izq.fkcedemple,  Izq.fkcodexcu  ;
BEGIN
        OPEN   cursor_tabla  ;
        LOOP
                FETCH NEXT FROM  cursor_tabla  INTO  regis_temp  ;  -- FETCH Recupere una fila
                /*   i := i + 1  ;
                RAISE NOTICE ' Paso % - Numero % - Registro %',  i,  registro.pknr,  registro;
                regis_temp.nr = registro.pknr ;  ....  regis_temp.hora = registro.hora  ;   */

                --  Retorne el siguiente regis_temp  /  Ir al siguiente registro
                RETURN NEXT   regis_temp   ;
                EXIT  WHEN  NOT  FOUND  ;  /*  Salga del ciclo cuando no existan registros  */
        END LOOP ;
        CLOSE  cursor_tabla  ;
        RETURN  ;
END  ;
$$ LANGUAGE plpgsql  ;
```

```
=# SELECT  *  FROM  festructura01 () ;
```

```
=# DROP  TYPE  estructura ;  -→ Borra la estructura
```

Pregunta si existe la estructura y si existe la elimina

```
=# DROP  TYPE  IF  EXISTS  Nombre_Estructura ;
```

Borra la estructura con todos los objetos asociados en cascada.

```
=# DROP TYPE  Nombre_Estructura  CASCADE  ;
```

Ejemplo 15 (RETURNS SETOF ---→ Conjunto de datos):. Construir una función que retorne los siguientes datos de los retardos: Número del retardo, cedula del empleado, nombre del empleado, la descripción de la excusa, la fecha del retardo y la hora del retardo.

```
CREATE OR REPLACE FUNCTION
ftabla_virtual01(OUT xnr  int ,  OUT xced  varchar ,  OUT xnom  varchar ,
                 OUT xexcusa  varchar ,  OUT xfecha  date ,  OUT xhora  time)
RETURNS  SETOF  RECORD  AS $$
DECLARE
    registro  RECORD  ;
    i         integer := 1  ;
```

```
BEGIN
        FOR  registro  IN  (SELECT  pknr,  fkcedemple AS cedula,  nomemple AS nombre,
                                        dexcusa AS excusa,  Izq.fecha,  Izq.hora
                            FROM  tdretardos  AS Izq
                            JOIN  tmempleados AS Der1  ON  Izq.fkcedemple = Der1.pkcedemple
                            JOIN  tmexcusas  AS Der2  ON  Izq.fkcodexcu = Der2.pkcodexcu
                            ORDER BY Izq.fkcedemple, Izq.fkcodexcu )
        LOOP
                /*
                i :=  i  +  1  ;
                RAISE NOTICE ' Paso % - Numero % - Registro %',  i,  registro.nr,  registro ;
                */
                /* Mover los datos de la variable registro a los campos OUT  */
                xnr :=  registro.pknr  ;
                xced :=  registro.cedula  ;
                xnom :=  registro.nombre  ;
                xexcusa :=  registro.excusa  ;
                xfecha =  registro.fecha  ;
                xhora :=  registro.hora  ;
                RETURN NEXT  ;   -- Ir al siguiente registro
        END LOOP  ;
        RETURN  ;
END  ;
$$ LANGUAGE   plpgsql  ;
.
=# SELECT  *  FROM  ftabla_virtual01 ()  ;
=# SELECT  *  FROM  ftabla_virtual01 ()  WHERE  xced  =  '6000'  ;
```

Ejemplo 16 (RETURNS SETOF ---→ Conjunto de datos): Se desea construir una función que retorne los siguientes datos de los retardos: Número del retardo, cedula del empleado, nombre del empleado, la descripción de la excusa, la fecha del retardo y la hora del retardo

```
CREATE OR REPLACE FUNCTION
ftabla_virtual02(OUT xnr int , OUT xced varchar , OUT xnom varchar ,
                 OUT xexcusa varchar , OUT xfecha date , OUT xhora time)
RETURNS SETOF RECORD AS $$
DECLARE
    Var_sql text ;
    registro RECORD ;
    i integer := 1 ;
BEGIN
    var_sql := 'SELECT pknr, fkcedemple AS cedula, nomemple AS nombre,
                dexcusa AS excusa, Izq.fecha, Izq.hora
            FROM tdretardos AS Izq
            JOIN tmempleados AS Der1 ON Izq.fkcedemple = Der1.pkcedemple
            JOIN tmexcusas AS Der2 ON Izq.fkcodexcu = Der2.pkcodexcu
            ORDER BY Izq.fkcedemple, Izq.fkcodexcu ' ;

    FOR registro IN EXECUTE var_sql
    LOOP
        /*
        i := i + 1 ;
        RAISE NOTICE ' Paso % - Numero % - Registro %', i, registro.pknr, registro ;
        */
        xnr := registro.pknr ;
        xced := registro.cedula ;
```

```
            xnom :=   registro.nombre  ;
            xexcusa :=   registro.excusa  ;
            xfecha =   registro.fecha  ;
            xhora :=   registro.hora  ;
            RETURN NEXT  ;      -- Ir al siguiente registro
      END LOOP  ;

      RETURN  ;
END  ;
$$ LANGUAGE   plpgsql  ;

=# SELECT  *  FROM  ftabla_virtual02 ()  ;
=# SELECT  *  FROM  ftabla_virtual02()  WHERE  xced  =  '6000' ;
```

Ejemplo 17: Tablas temporales

```
CREATE OR REPLACE FUNCTION leer1()  RETURNS  integer  AS  $$
BEGIN
      DROP TABLE IF EXISTS temporal01;
      DROP TABLE IF EXISTS temporal02;

      CREATE TEMP TABLE temporal01 AS SELECT * FROM tmempleados;

      CREATE TEMP TABLE temporal02 AS SELECT * FROM tmcargos;

      RETURN   1;
END;
$$ LANGUAGE plpgsql;
```

```
=# select leer1();
```

Ejemplo 18: concatenación de cedula + nombre en una fila

```
CREATE OR REPLACE FUNCTION leer2()   RETURNS   text   AS   $$
DECLARE
        retorno   TEXT;
        fila         RECORD;
BEGIN
        retorno := '[';

        FOR fila IN SELECT pkcedemple FROM tmempleados LIMIT 10
        LOOP
                retorno := retorno || fila.pkcedemple || ', ';
        END LOOP;
        retorno := substring(retorno, 1, length(retorno)-1);
        retorno := retorno || '],[';

        FOR fila IN SELECT nomemple FROM tmempleados LIMIT 10
        LOOP
                retorno := retorno || '*' || fila.nomemple || '*, ';
        END LOOP;
        retorno := substring(retorno, 1, length(retorno)-1);
        retorno := retorno || ']';

        RETURN retorno;
END;
$$ LANGUAGE plpgsql;
```

```
=# select leer2();
```

6.- Renombrar la función

=# **ALTER FUNCTION** Nombre_Función(Parametro_Tipo1, ….. , Parametro_TipoN)
RENAME TO Nuevo_Nombre_Función;

Ejemplo 01: Cambiar el nombre de la función
=# **ALTER FUNCTION** fsuma01(int, int) **RENAME TO** fsuma001;

7.- Borrar una función

=# **DROP FUNCTION** FNombre (Tipo_campo1, ..., Tipo_campoN) ;

Referencias ---> Videos sobre Teorías de Funciones

1.- Qué es una función
1.1.- https://www.youtube.com/watch?v=It3tHF9EZfk
1.2.- https://www.youtube.com/watch?v=9ePQp0LMYOk
1.3.- https://sqlearning.com/es/funciones/funciones-usuario/
1.4.- https://www.postgresql.org/docs/current/xfunc-sql.html
1.5.- https://www.youtube.com/watch?v=MdfqWrTIHug
1.6.- https://www.youtube.com/watch?v=t9IFeozZ-YI

2.- Funciones que retornan Tablas
2.2.- https://neon.tech/postgresql/postgresql-plpgsql/plpgsql-function-returns-a-table
2.2.- https://sqltemuco.wordpress.com/2016/10/06/retornar-multiples-registros-en-postgresql/
2.3.- https://www.youtube.com/watch?v=jxpaFcmDCnY
2.4.- https://www.youtube.com/watch?v=s9xaGNAI0KQ
2.5.- https://www.youtube.com/watch?v=VbvWbqpolXw
2.6.- https://www.youtube.com/watch?v=jxpaFcmDCnY
2.6.- https://www.youtube.com/watch?v=WOsjB0zjryk&list=PLhnOQ_ypZS5ViZsK28otsfImA56Gm0lG_&index=2

3.- Tablas temporales
3.1.- https://www.youtube.com/watch?v=5khN3QrCXyM
3.2- Crear tablas temporales , cursores y bucles en PL/pgSQL
https://www.youtube.com/watch?v=DYTWvjtcFPw