CURSORES EN POSTGRESQL

En PostgreSQL existen 2 tipos de cursores: explícitos o implícitos.

Los cursores explícitos son unas variables especiales que almacenan datos respecto de consultas a una o más tablas, por lo tanto, un cursor esta necesariamente formado por una instrucción SELECT y como tal, debe ser declarado como una variable.

Los cursores implícitos están insertos directamente en el código como una instrucción SELECT, sin necesidad de ser declarados previamente, generalmente requieren alguna variable del tipo RECORD o ROWTYPE para capturar las filas.

```
--  Uso simple de cursores EXPLÍCITOS
CREATE OR REPLACE FUNCTION expl_cursor1() RETURNS SETOF clientes AS
$$
DECLARE
   cur_clientes CURSOR FOR SELECT * FROM clientes;      -- Declaración EXPLICITA del cursor
   registro clientes%ROWTYPE;
BEGIN
            -- Procesa el cursor
  FOR registro IN cur_clientes LOOP
     RETURN NEXT registro;
  END LOOP;
  RETURN;
END
$$ LANGUAGE 'plpgsql'
```

```
--  Uso simple de cursores IMPLICITOS
CREATE OR REPLACE FUNCTION impl_cursor2() RETURNS SETOF clientes AS
$$
DECLARE
   registro clientes%ROWTYPE;
BEGIN
  FOR registro IN SELECT * FROM clientes LOOP            -- Cursor IMPLICITO en el ciclo FOR
     RETURN NEXT registro;
  END LOOP;
  RETURN;
END
$$ LANGUAGE 'plpgsql'
```

Por comodidad resulta más simple usar cursores implícitos ya que se declaran directamente en el ciclo FOR, sin embargo, cuando las consultas son más complejas o bien es necesario reutilizar el cursor en más de una oportunidad lo conveniente sería usar cursores explícitos,

DECLARACION DE CURSORES

-- SINTAXIS DE UN CURSOR
DECLARE name [BINARY] [INSENSITIVE] [[NO] SCROLL]
    CURSOR [{ WITH | WITHOUT } HOLD] FOR query

-- Para recorrerlo obteniendo un registro
FETCH [direction [FROM | IN]] cursor_name

where direction can be empty or one of:
    NEXT
    PRIOR
    FIRST
    LAST
    ABSOLUTE count
    RELATIVE count
    count
    ALL
    FORWARD
    FORWARD count
    FORWARD ALL
    BACKWARD
    BACKWARD count
    BACKWARD ALL

-- Para modificar la posicion actual (esta accion no retorna registros)
MOVE [direction [FROM | IN]] cursor_name

where direction can be empty or one of:
    NEXT
    PRIOR
    FIRST
    LAST
    ABSOLUTE count
    RELATIVE count
    count
    ALL
    FORWARD
    FORWARD count
    FORWARD ALL
    BACKWARD
    BACKWARD count
    BACKWARD ALL

-- Para cerrarlo liberando sus recursos
CLOSE { name | ALL }

Dentro de la declaración de un cursor disponemos de 3 formas de definirlo:

```
DECLARE
   curs1 refcursor;
   curs2 CURSOR FOR SELECT * FROM tenk1;
   curs3 CURSOR (key integer) FOR SELECT * FROM tenk1 WHERE unique1 = key;
```

Donde el primero es una simple declaración sin asociación a ninguna consulta por lo que puede referenciarse a cualquier resultado. El segundo esta asociado a una consulta y el tercero admite parámetros para la ejecución de la consulta

Podemos abrir un cursor del primer tipo asociándolo en ese momento a una consulta, incluso utilizando formatos de consulta dinámica

```
OPEN curs1 FOR EXECUTE format('SELECT * FROM %I WHERE col1 = $1',tabname) USING
keyvalue;
```

Los otros 2 formatos como ya están asociados a una consulta los abrimos de otra forma:

```
OPEN curs2;
OPEN curs3(42);
```

Podemos llegar a retornar cursores, como se plantea en el siguiente ejemplo de forma simplificada:

```
--creacion de una tabla ejemplo
CREATE TABLE test (col text);
INSERT INTO test VALUES ('123');

--creacion de la función de retorna un cursor
CREATE FUNCTION reffunc(refcursor) RETURNS refcursor AS
$$
BEGIN
   OPEN $1 FOR SELECT col FROM test;
   RETURN $1;
END;
$$ LANGUAGE plpgsql;

--prueba de funcionamiento
BEGIN;
SELECT reffunc('funccursor');
FETCH ALL IN funccursor;
COMMIT;
```

FORMAS DE RECORRER UNA CONSULTA A TRAVES DE UN CURSOR

El uso de la instrucción WHILE requiere un cursor Explícito, abrir el cursor y finalmente cerrarlo, para operar sobre éste usamos la instrucción FETCH que va capturando los datos fila a fila hasta el final del cursor, es decir, mientras encuentre datos (FOUND) :

```
CREATE OR REPLACE FUNCTION cursor_while() RETURNS VOID AS
$$
DECLARE
   reg        RECORD;
   cur_clientes CURSOR FOR SELECT * FROM clientes
            ORDER BY nombre_cliente;
BEGIN
  OPEN cur_clientes;
  FETCH cur_clientes INTO reg;
  WHILE( FOUND ) LOOP
     RAISE NOTICE ' PROCESANDO  %', reg.nombre_cliente;
     FETCH cur_clientes INTO reg;
  END LOOP ;
  CLOSE    cur_clientes;
  RETURN;
END
$$ LANGUAGE 'plpgsql';
```

LOOP es una instrucción muy básica de iteración, también requiere el uso de un cursor explícito, a diferencia del WHILE, es necesario dar una salida mediante EXIT, en este caso, hasta que ya no se encuentren datos en el cursor (NOT FOUND) lo cual indica que se ha llegado al final del recorrido.

```
CREATE OR REPLACE FUNCTION cursor_loop() RETURNS VOID AS
$$
DECLARE
   reg        RECORD;
   cur_clientes CURSOR FOR SELECT * FROM clientes
            ORDER BY nombre_cliente;
BEGIN
  OPEN cur_clientes;
  LOOP
         FETCH cur_clientes INTO reg;
         EXIT WHEN NOT FOUND;
         RAISE NOTICE ' PROCESANDO  %', reg.nombre_cliente;
  END LOOP;
  CLOSE    cur_clientes;
  RETURN;
END
$$ LANGUAGE 'plpgsql';
```

FOR es el favorito para los que usan PostgreSQL por su manera simple de recorrer los datos, sin importar si el cursor es implícito, explicito, y sin abrir cursores para su procesamiento

```
CREATE OR REPLACE FUNCTION cursor_for() RETURNS VOID AS
$$
DECLARE
   reg         RECORD;
   cur_clientes CURSOR FOR SELECT * FROM clientes
            ORDER BY nombre_cliente;
BEGIN
  FOR reg IN cur_clientes LOOP
   RAISE NOTICE ' PROCESANDO  %', reg.nombre_cliente;
  END LOOP;
  CLOSE     cur_clientes;
  RETURN;
END
$$ LANGUAGE 'plpgsql';
```

Aunque se usan mucho para operaciones de lectura y extracción de resultados, una vez posicionados podemos ejecutar también operaciones de actualización y borrado usando la clausula CURRENT OF

```
UPDATE table_name
SET column = value, ...
WHERE CURRENT OF cursor_variable;

DELETE FROM table_name
WHERE CURRENT OF cursor_variable;
```

La complejidad del trabajo con cursores llega a la posibilidad de crear funciones que retornan cursores o bien un conjunto de cursores a través del indicador de retorno SETOF

En el siguiente enlace tenemos algunos ejemplo con los que trabajar

http://www.sqlines.com/postgresql/how-to/return_result_set_from_stored_procedure