Introducción a las Bases de Datos y Bases de Datos

SQL PROCEDURAL-PL/PGSQL

TEORÍA 7

2017

Tecnicaturas TUPAR y TUDAI

SQL PROCEDURAL

Posibilita el uso de código procedural Conjuntamente con sentencias SQL que son almacenadas dentro de la BD. El código procedural es ejecutado por el DBMS cuando es invocado (directa o indirectamente) por el usuario de la BD.

Ventajas:

- Aísla partes comunes existentes en las aplicaciones, delegándolas en el DBMS.
- Eficiencia (en general, cerca de un orden de magnitud)

Desventajas:

- *Cada proveedor de BD tiene su propio lenguaje procedural.*
- *El SQL3 incorporó estas características, pero poco de lo definido anteriormente se ajustaba a un estándar.*

SQL PROCEDURAL

Se puede utilizar SQL Procedural para definir:

- **Trigger**: Es un procedimiento que es invocado **automáticamente** por el DBMS en respuesta a un evento sobre una tabla.
- **Stored Procedure**: Es un procedimiento que es invocado explícitamente por el usuario.
- **Función**: puede ser predefinida o definida por el usuario para realizar operaciones específicas sobre los datos, y pueden ser invocadas desde un trigger, stored procedure o explícitamente.

PROCEDIMIENTOS Y FUNCIONES EN POSTGRESQL

Para Postgres todos son funciones, sólo que hay funciones que devuelven void (Procedimientos).

```
CREATE [ OR REPLACE ] FUNCTION nombre_funcion(
[ [ argmodo ] [ argnombre ] argtipo [, ...] ] )
RETURNS tipo AS $$
[ DECLARE ] [ declaraciones de variables ]
BEGIN
  codigo
END;
$$ LANGUAGE plpgsql
[IMMUTABLE | STABLE | VOLATILE]
[CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT]
[EXTERNAL SECURITY INVOKER | EXTERNAL SECURITY DEFINER];
```

PROCEDIMIENTOS Y FUNCIONES EN POSTGRESQL

```
CREATE OR REPLACE FUNCTION Sumador(integer)
RETURNS integer AS $$
BEGIN
    RETURN $1 + 1;
END; $$ LANGUAGE plpgsql;
```

```
CREATE OR REPLACE FUNCTION Sumador(unNumero
integer)
RETURNS integer AS $$
BEGIN
    RETURN unNumero + 1;
END; $$ LANGUAGE plpgsql;
```

DECLARACIÓN DE VARIABLES

```
CREATE FUNCTION Ejemplo2(integer, integer) ……
DECLARE
    numero1     ALIAS FOR $1;        // Primer parámetro
    numero2     ALIAS FOR $2;        // Segundo parámetro
    constante   CONSTANT integer := 100;
    resultado     INTEGER;
    resultado_txt   TEXT DEFAULT   'Texto por defecto';
    tipo_reg   voluntario%rowtype;
                // variable del tipo registro
    tipo_col   voluntario.nombre%type;
                // variable del tipo columna
```

FUNCIONES – TIPOS DE RETORNO

- **void** – This is for when the trigger should not return anything. It is just doing some backgroud process for you.
- **trigger** – esta opción debe ser seteada para las de tipo trigger
- **boolean, text, etc** – esto es para retornar sólo valores
- **SET OF schema.table** – Esto es para retornar varias filas de datos.

FUNCIONES TIPO TRIGGER

En Postgres los triggers constan de dos partes, la función denominada como **TRIGGER** y la declaración del Trigger en si.

Cuando una función PL/pgSQL es denominada como Trigger, se crean **automáticamente** las siguientes variables especiales en el bloque de nivel superior:

NEW Tipo de dato RECORD; variable que almacena la nueva fila para las operaciones INSERT/UPDATE en Triggers a nivel ROW, en los Triggers a nivel STATEMENT es NULL .

OLD Tipo de datos RECORD; variable que almacena la antigua fila para operaciones las UPDATE/DELETE en Triggers de nivel ROW, en Triggers de nivel STATEMENT es NULL

FUNCIONES TIPO TRIGGER

Algunas otras (hay muchas mas) variables especiales son:

TG_NAME Tipo de dato text; variable que contiene el nombre del trigger actualmente disparado.

TG_WHEN Tipo de dato text; una cadena conteniendo el string BEFORE o AFTER dependiendo de la definición del trigger.

TG_LEVEL Tipo de dato text; una cadena conteniendo el string ROW o STATEMENT dependiendo de la definición del trigger.

TG_OP Tipo de dato text; una cadena conteniendo el string INSERT, UPDATE o DELETE indicando por cuál operación se disparó el trigger.

TG_RELID Tipo de dato oid; el ID (identificador del objeto) de objeto de la tabla que causó la invocación de trigger.

TRIGGER EJEMPLO

```
CREATE TABLE tabla1 (
  clave SERIAL,
  valor INTEGER,
  valor_tipo VARCHAR,
  PRIMARY KEY(clave)
) ;

CREATE TABLE tabla2 (
  clave INTEGER,
  valor INTEGER,
  valor_tipo VARCHAR,
  user_name NAME,
  accion VARCHAR,
  accion_hora TIMESTAMP,
  PRIMARY KEY(clave)
) ;
```

TRIGGER EJEMPLO

```
CREATE FUNCTION copia_tabla1 ( )
RETURNS trigger AS $body$
BEGIN
        IF TG_OP = 'INSERT' THEN
                INSERT INTO tabla2
                VALUES (NEW.clave, NEW.valor, NEW.valor_tipo, current_user, TG_OP, now());
                RETURN NEW;
        END IF;
        IF TG_OP = 'UPDATE' THEN
                INSERT INTO tabla2
                VALUES (NEW.clave, NEW.valor, NEW.valor_tipo, current_user, TG_OP, now());
                RETURN NEW;
        END IF;
        IF TG_OP = 'DELETE' THEN
                INSERT INTO tabla2
                VALUES (OLD.clave, OLD.valor, OLD.valor_tipo, current_user, TG_OP, now());
                RETURN OLD;
        END IF;
END; $body$
LANGUAGE 'plpgsql'
VOLATILE ;
```

TRIGGER EJEMPLO

```
CREATE TRIGGER tabla1_tr
  BEFORE INSERT OR UPDATE OR DELETE
  ON tabla1 FOR EACH ROW
  EXECUTE PROCEDURE copia_tabla1();

INSERT INTO tabla1 (valor, valor_tipo) VALUES ('30',
‘metros');
INSERT INTO tabla1 (valor, valor_tipo) VALUES ('10',
‘pulgadas');
UPDATE tabla1 SET valor = '20' WHERE valor_tipo =
‘pulgadas ';
DELETE FROM tabla1 WHERE valor_tipo = 'pulgadas ';
INSERT INTO tabla1 (valor, valor_tipo) valorS ('50',
‘pulgadas');
```

PL/PGSQL -ASIGNACIÓN

Una asignación de un valor a una variable o campo de fila o de registro se escribe:

identifier := expression;

- Si el tipo de dato resultante de la expresión no coincide con el tipo de dato de las variables, el interprete de PL/pgSQL realiza el cast implícitamente.

PL/PGSQL – SELECT INTO

Una asignación de una selección completa en un registro o fila puede hacerse del siguiente modo:

SELECT expressions INTO target FROM ...;

Por ejemplo:

```
SELECT * INTO myrec
FROM EMP
WHERE nombre_emp = mi_nombre;
```

Si la selección devuelve múltiples filas, solo la primera fila se mueve a la variable, todas las demás se descartan.

PL/PGSQL –ESTRUCTURAS DE CONTROL

IF-THEN
IF boolean-expression **THEN**
sentencias ;
END IF;

IF-THEN-ELSE
IF boolean-expression **THEN**
sentencias ;
ELSE
sentencias ;
END IF;

IF-THEN-ELSIF-ELSE
IF boolean-expression **THEN**
sentencias;
[**ELSIF** boolean-expression **THEN**
sentencias;
[**ELSIF** boolean-expression **THEN**
sentencias ...;]]
[**ELSE**
sentencias ;]
END IF;

PL/PGSQL –ESTRUCTURAS DE CONTROL

WHILE expresión **LOOP**
    Sentencias;
END LOOP;

Repite una secuencia de sentencias mientras que la condición se evalúe a verdadero

FOR nombre **IN** [REVERSE] expresión .. expresión **LOOP**
    Sentecias;
END LOOP;

Crea un ciclo que itera sobre un rango de valores enteros. La variable *nombre* es definida automáticamente como de tipo *integer* y existe sólo dentro del ciclo. Las dos expresiones determinan el intervalo de iteración y son evaluadas al entra. Por defecto el intervalo comienza en 1, excepto cuando se especifica REVERSE que es -1.

INTRODUCCIÓN

- Cuando se trabaja con SQL Procedural las consultas pueden devolver resultados que involucren una o más de una filas.
- Para todos los casos es necesario contar con variables o estructuras de retorno para procesar dichos resultados.
  - Si el resultado devuelve una fila entonces se puede utilizar una variable.
  - Si el resultado devuelve más de una fila entonces es necesario utilizar cursores

CURSORES: DEFINICIÓN

- Son una estructura de control utilizada para el recorrido (y potencial procesamiento) de los registros del resultado de una consulta.
- Se utiliza para el procesamiento individual de las filas devueltas ante una consulta. Los lenguajes de programación son procedurales (no disponen de ningún mecanismo para manipular conjuntos de datos en una sola instrucción) → las filas deben ser procesadas de forma secuencial.
- Puede verse como un mecanismo de iteración sobre la colección de filas que habrá en el conjunto resultado.
- Lo ciclos FOR utilizan internamente un cursor.
- Evita problemas de memoria cuando es necesario trabajar con conjuntos de datos muy grandes.

CURSORES

- Los cursores pueden ser definidos en forma explícita o implícita.
  - Forma explícita: es necesario declararlo, abrirlo, recorrerlo y cerrarlo

    Sentencias OPEN – FETCH – CLOSE
  - Forma implícita: FOR-IN-SELECT

CURSORES

- FOR-IN-SELECT itera a través de los resultados de una consulta:

```
[<<etiqueta>>]
FOR registro | fila IN select_query LOOP
   sentencias
END LOOP;
```

  - A la variable *registro* o *fila* le son sucesivamente asignadas todas las filas resultantes de la consulta SELECT y el cuerpo del bucle es ejecutado para cada fila. Aquí tiene un
  - Si el bucle es terminado por un estamento EXIT, el valor de la última fila asignada es todavía accesible tras la salida del bucle.

EJEMPLO

Ver ejemplo_cursores.sql

Documentación

- ✓ *http://www.postgresql.org/docs/current/static/plpgsql-cursors.html*