

People matter, results count.

For internal use only

Modulo SQL

Temas avanzados de PL-SQL




Stored Procedures (Procedimientos almacenados)

- Funciones creadas por el desarrollador que se compilan y guardan en la base de datos
 - Procedure vs. Function
 - Un procedimiento no devuelve ningún valor, pero una función si
 - Hasta PostgreSQL v10, un procedimiento era una función que simplemente devolvía un `void`
 - Desde PostgreSQL v11 ya existen como objeto independiente
-

Ejemplo 1: Hola Mundo

```
create function hello() returns void as $$  
begin  
    raise notice 'Hola mundo con funcion';  
end;  
$$ language plpgsql;
```

```
create procedure hello() as $$  
begin  
    raise notice 'Hola mundo con procedimiento';  
end;  
$$ language plpgsql;
```



```
CREATE FUNCTION `hola` ()  
RETURNS varchar(4) DETERMINISTIC  
BEGIN  
RETURN "hola";  
END
```

```
USE `indragt2`;  
DROP function IF EXISTS `hola`;  
DELIMITER $$USE `indragt2`  
$$CREATE FUNCTION `hola` ()  
RETURNS varchar(4) DETERMINISTIC  
BEGIN  
RETURN "hola";  
END$$DELIMITER ;
```

```
SELECT hola(), first_name FROM  
indragt2.employees;
```

```
CREATE FUNCTION `generoRND` ()  
RETURNS INTEGER deterministic  
BEGIN  
RETURN round(rand()*10000,0);  
END
```

```
SELECT hola() ,generoRND() as saludo, first_name FROM indragt2.employees;
```

POSTGRES

Ejemplo 2: Suma10

```
create function suma10( a integer ) returns integer as $$  
declare  
    b integer;  
begin  
    b := 10;  
    return a + b;  
end;  
$$ language plpgsql;
```

MYSQL

```
CREATE DEFINER=`root` @`%`  
FUNCTION `sumar10`(a int)  
RETURNS int DETERMINISTIC  
BEGIN  
DECLARE b int;  
set b = 10;  
RETURN a + b;  
END
```

```
SELECT hola() ,generoRND() as saludo,  
first_name, sumar10(5) as suma FROM  
indragt2.employees;
```



Ventajas de los Stored Procedures

- Performance

- Codigo compilado y optimizado para la base de datos
- Se pueden ejecutar varias sentencias sin tener que enviar estados de ejecucion al usuario.

- Seguridad

- Se puede definir controles de acceso
- Se envian mucho menos datos por la red

- La aplicacion es mas simple

- Se pueden usar Triggers para mantener la integridad de los datos



Desventajas de los Stored Procedures

- Portabilidad
 - El lenguaje PL es generalmente mas difícil de implementar y mantener que un lenguaje convencional
 - Tiene muchos menos features
 - El soporte es solo del proveedor de base de datos
-

Ejemplo Store Procedure

```
CREATE DEFINER=`root` @`%`  
PROCEDURE `chau`(IN nombre VARCHAR(10))  
BEGIN  
Select first_name, last_name from employees  
where first_name = nombre;  
END
```

call chau("Neena");

PostgreSQL PL/pgSQL

• Que cosas si tenemos en PL/SQL de los lenguajes de programacion convencionales:

- Variables y tipos
- Sentencias de Control (If, else , etc)
- Functiones

• <https://www.postgresql.org/docs/current/plpgsql.html>

• <https://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx>

Sintaxis de una Funcion

```
CREATE [OR REPLACE] FUNCTION name ( parameters )  
RETURNS type AS $$  
DECLARE  
declarations  
BEGIN  
statements  
END;  
$$ LANGUAGE plpgsql;
```

```
DROP FUNCTION name ( argtype [, ...]);
```

Sintaxis Basica

- Operador de asignacion **:=**
- Declaracion de Variables
 - **local_a text := "Hola";**
 - **v_string ALIAS FOR \$1;**
- **RAISE**
 - Niveles: **DEBUG, LOG, INFO, NOTICE, WARNING, EXCEPTION**
- Formateo del mensaje con **%**
- <https://www.postgresql.org/docs/current/plpgsql-errors-and-messages.html>

Convenciones de nombres

- El objetivo de impedir conflictos con los nombre de las tablas , columnas y otros objetos de la base de datos
- El mas sencillo es con prefijos :
 - Parametros con **p_**
 - Variables locales con **l_**
 - Variables globales con **g_**

SELECT...INTO

- Asignar una consulta a variables (resultado de una fila)

```
SELECT select_list INTO variable_list  
FROM table_list  
[WHERE condition]  
[ORDER BY order_list];
```

SE USA EN LUGAR DEL CREATE TABLE

CREATE TEMPORARY TABLE

- Asignar multiples filas a una nueva tabla temporal

```
CREATE TEMPORARY TABLE nombre IAS
    SELECT select_list
FROM table_list
[WHERE condition]
[ORDER BY order_list];
```



IF THEN ELSE

```
IF condition1 THEN  
    statements1  
ELSIF condition2 THEN  
    statements2  
ELSE  
    statements3  
END IF;
```

NOTA: No olvidarse del punto y coma (;) al final del
END IF.

Loops

LOOP

statements

EXIT WHEN *condition*;

statements

END LOOP;

WHILE *condition* LOOP

statements

END LOOP;

FOR *loop_variable* IN [REVERSE]

lower_bound..*upper_bound* LOOP

statements

END LOOP;



Triggers

- Procedimientos que son ejecutados automaticamente cuando se cambia algun dato (INSERT, DELETE, UPDATE)
 - Comunmente se utilizan para :
 - Hacer que se cumplan restricciones de datos
 - Auditoria
 - Replicacion
-

Ejemplo de Trigger

- Crear un trigger que audite los cambios de las calificaciones en la tabla `enrollment` y guardarlos en `grade_changes`

```
CREATE TABLE grade_changes (  
    enrollment_id integer,  
    old_grade_id integer,  
    new_grade_id integer,  
    timestamp timestamp  
);
```

Paso 1: Creamos el Trigger

```
Create trigger grade_audit  
after update  
on enrollment  
for each row  
execute procedure grade_audit();
```

Sintaxis de Trigger

```
CREATE TRIGGER name
{ BEFORE | AFTER } { event [ OR ... ] }
ON table
[ FOR EACH { ROW | STATEMENT } ]
EXECUTE PROCEDURE funcname ( arguments );
```

```
DROP TRIGGER name ON table;
```



Eventos de Triggering

- INSERT
 - DELETE
 - UPDATE
-



Before o After

- **BEFORE:** El trigger se ejecuta antes del evento elegido
 - **AFTER:** El trigger se ejecuta despues del evento
-



Statement Trigger vs. Row Trigger

- Statement Trigger

- Default
- Se ejecuta una vez por sentencia

- Row Trigger

- Se ejecuta una vez por fila
-

Paso 2: Creamos la Funcion


```
create or replace function grade_audit()  
returns trigger as $$  
begin  
    if new.id = old.id and new.grade_id <> old.grade_id then  
        insert into grade_changes values (  
            new.id, old.grade_id, new.grade_id,  
            current_timestamp );  
    end if;  
    return null;  
end;  
$$ language plpgsql;
```


Funciones de Triggers

- No tienen parametros de entrada
 - El tipo de dato de respuesta es `trigger`
 - Tiene variables especiales
 - NEW, OLD
 - Others: <https://www.postgresql.org/docs/current/plpgsql-trigger.html>
-

Valores de respuesta de un Trigger

- Los triggers de Statement y los de tipo AFTER deben devolver siempre NULL
- Triggers de tipo BEFORE
 - Pueden devolver NULL , eso hace que la operacion que se ejecutaba de cancele
 - Para los eventos insert y update, las filas que se devuelven con el return pasan a ser las que se insertaran o actualizaran en la tabla (ej: return NEW;)



```
ALTER TABLE employees  
  ADD CONSTRAINT emp_job  
  FOREIGN KEY(job_id)  
  REFERENCES jobs(job_id)|
```

VIEWS

```
CREATE VIEW `emp_it` AS  
Select first_name, last_name, job_id  
from employees  
where job_id = "IT_PROG";
```

```
SELECT * FROM emp_it;
```

```
1 • CREATE  
2     ALGORITHM = UNDEFINED  
3     DEFINER = `root`@`%`  
4     SQL SECURITY DEFINER  
5 VIEW `emp_it` AS  
6     SELECT  
7         `employees`.`first_name` AS `first_name`,  
8         `employees`.`last_name` AS `last_name`,  
9         `employees`.`job_id` AS `job_id`  
10    FROM  
11        `employees`  
12    WHERE  
13        (`employees`.`job_id` = 'IT_PROG')
```
