

PROBLEMES PLPGSQL

Maih 2019

| | |
|----------------------------------|---|
| 1. Introducció | 1 |
| 2. Sintaxi i Semàntica..... | 3 |
| 2.1. Create Function | 3 |
| 2.2. Triggers | 4 |
| 2.3. Variables predefinides..... | 5 |
| 3. Exercicis Function..... | 6 |
| 4. Exercicis Trigger..... | 7 |

1. INTRODUCCIÓ



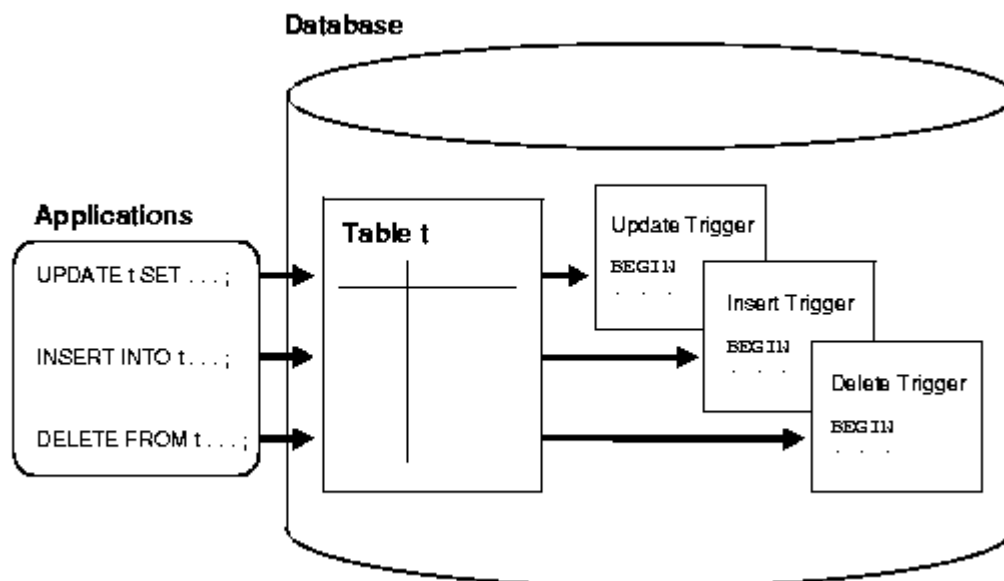
☞ Una base de datos con disparadores es una base de datos activa

Un *disparador* es una “computación” que el sistema ejecuta de manera automática como respuesta a la modificación de la base de datos.



Los disparadores se deberían escribir con sumo cuidado, dado que un error de un disparador detectado en tiempo de ejecución causa el fallo de la instrucción de inserción, borrado

0



actualización que inició el disparador. En el peor de los casos esto podría dar lugar a una cadena infinita de disparos.

Por ejemplo, supóngase que un disparador de inserción sobre una relación realice otra (nueva) inserción sobre la misma relación. La acción de inserción dispara otra acción de inserción, y así hasta el infinito.

Generalmente, los sistemas de bases de datos limitan la longitud de las cadenas de disparadores (por ejemplo, hasta 16 o 32), y consideran que las cadenas mayores son erróneas.

Abans del SQL:1999 no formaven part de l'estàndart. Desgraciadament cada SGBD els va implementar a la seua manera engendrant incompatibilitats entre ells.

2. SINTAXI I SEMÀNTICA

2.1. Create Function

```
CREATE [ OR REPLACE ] FUNCTION
    name ( [ [ argmode ] [ argname ] argtype [, ...] ] )
    [ RETURNS rettype ]
    {
        LANGUAGE langname
        | IMMUTABLE | STABLE | VOLATILE
        | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
        | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY
DEFINER
        | COST execution_cost
        | ROWS result_rows
        | SET configuration_parameter { TO value | = value | FROM
CURRENT}
        | AS 'definition'
        | AS 'obj_file', 'link_symbol'
    } ...
    [ WITH ( attribute [, ...] ) ]
```

Exercici

Reescribe la sintaxis de Create Function de una forma “más amigable”.

2.2. Triggers

```
CREATE TRIGGER nom_trig {BEFORE | AFTER}
{INSERT | DELETE | UPDATE} [OR {INSERT | DELETE | UPDATE} ...]
ON nom_taula
[FOR EACH {ROW | STATEMENT}]
EXECUTE PROCEDURE nom_funció ([paràmetres]);
```



Se debe crear la función antes de escribir el disparador



Para el tipo de salida de las funciones asociadas a disparadores escribiremos *returns trigger*. Además siempre exige sentencia *return* en el cuerpo de la función. A trigger function must return either `NULL` or a record/row value having exactly the structure of the table the trigger was fired for.



If you have no specific reason to make a trigger before or after, the before case is more efficient, since the information about the operation doesn't have to be saved until end of statement.



FOR EACH STATEMENT es la opción por defecto!!.



El return de una función asociada a un disparador FOR EACH STATEMENT siempre es ignorado por eso debería escribirse `return null`. Análogamente podemos decir lo mismo para las funciones asociadas a disparadores AFTER y FOR EACH ROW.

Row-level triggers fired `BEFORE` can return null to signal the trigger manager to skip the rest of the operation for this row (i.e., subsequent triggers are not fired, and the `INSERT/UPDATE/DELETE` does not occur for this row). If a nonnull value is returned then the operation proceeds with that row value. Returning a row value different from the original value of `NEW` alters the row that will be inserted or updated (but has no direct effect in the `DELETE` case). To alter the row to be stored, it is possible to replace single values directly in `NEW` and return the modified `NEW`, or to build a complete new record/row to return.

2.3. Variables predefinides



Variables predefinides de PostgreSQL que es poden fer servir en funcions associades a triggers.

| <u>Nom</u> | <u>Tipus</u> | <u>Descripció</u> |
|------------|--------------|------------------------------|
| NEW | %ROWTYPE | Nous valors (INSERT UPDATE) |
| OLD | %ROWTYPE | Valors Antics(UPDATE DELETE) |
| TG_NAME | NAME | Nom del propi TRIGGER |
| TG_WHEN | TEXT | BEFORE AFTER |
| TG_LEVEL | TEXT | ROW SENTENCIA (SQL) |
| TG_OP | TEXT | INSERT, UPDATE DELETE |
| TG_RELID | OID | Identificador de la taula |
| TG_RELNAME | NAME | Nom de la taula |
| TG_NARGS | INT | Nombre d'arguments |
| TG_VARS | TEXT[] | Els propis arguments |

3. EXERCICIS FUNCTION


1. Funció anomenada Max2 que conteste amb el màxim de dos números introduïts com paràmetres.

```
CREATE OR REPLACE FUNCTION Max2(numeric,numeric) RETURNS numeric AS
'
declare
    aux numeric;
begin
    aux:=$1;
    if $2>$1 then aux:=$2;
end if; /* end if observeu que va separat */
    return aux;
end;
'
LANGUAGE plpgsql;
```

2. Fes ara la Max3 utilitzant la darrer Max2 

3. Funció que conteste amb un boolean si cert número passat com paràmetre és PAR.

```
create or replace function EsPar1(numeric) returns boolean as
'
declare resultat boolean;
begin
    if mod($1,2)=0 then resultat:=true;
                        else resultat:=false;
    end if;
    return resultat;
end;
' language plpgsql;
```

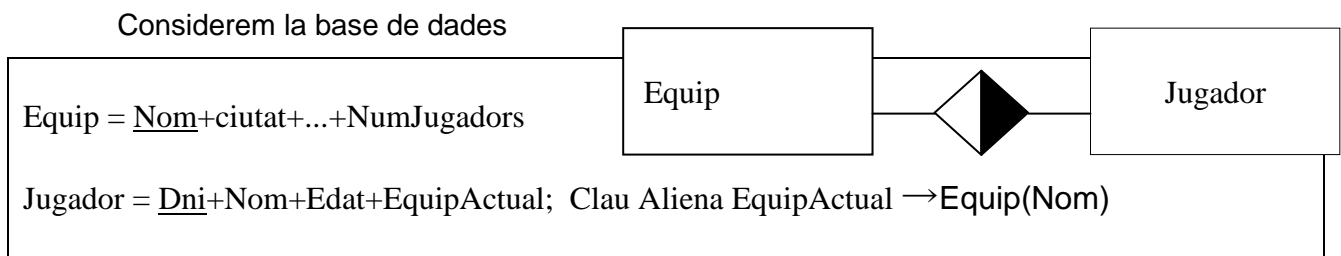
4. Funció que traga els números senars del 1 al 50 

5. Escriu una funció que elimina TOTS els espais en blanc d'una cadena que li passarem: 

4. EXERCICIS TRIGGER

6. Escriu un trigger que implemente el següent comportament: "després de cada inserció en la taula R = a0+a1 feu una còpia de la tupla inserida en la taula Rcopia"
7. Trigger que implemente el següent comportament actiu: "després de qualsevol actualització de la taula R deseu una tupla en Rlog=Codi+Usuari+Data+DiaSetmana+Hora+Event on controlarem l'històric de les manipulacions.
8. Sobre la taula Empleados=CodEmp+Nombre+Sueldo diseñad un disparador que guarde en HistóricoSueldos=CodEmp+Num + Fecha + SueldoAnterior + NuevoSueldo + PorcentajeCambio una tupla para cada cambio de sueldo. El campo Num será un número correlativo para cada empleado.

Pràctica Mantenir consistència informació derivada en binària 1:N



Cal obviar consideracions d'anàlisi com ara especificar si hi ha o no alguna restricció d'existència tot i que la pràctica consisteix en **dissenyar un conjunt de disparadors i funcions** que garantitzen la consistència de l'atribut derivat (redundància inter-taula) *NumJugadors*.

Per exemple, si esborrem un jugador d'un determinat equip aleshores hem de restar una unitat al NumJugadors d'eixe equip.



Dissenya els triggers necessaris i les respectives funcions per tal de mantenir la consistència de la BD.