BASES DE DATOS ACTIVAS

Plpgsql Triggers and Stored Procedure

Docentes : Ing. Fernando Sato

A.S. Sebastian Trossero

Versión: 201905041800

RESUMEN

- Introducción
- Funciones o Stored Procedure
- Triggers
- Casos de Estudio.

Plpgsql Trigger y Stored Procedure
Introducción

Las bases de datos activas incorporan principalmente 2 tipos de objetos a las BD Pasivas:

- **Trigger**

```
CREATE TRIGGER Actualizo_Clientes FOR Clientes
BEFORE  UPDATE AS
BEGIN
   New.Cliente_Usuario = CURRENT_USER;
   New.Cliente_Fecha   = CURRENT_DATE;
END
```

Nota: El ej anterior esta escrito en la implementación de Pl/Sql de Oracle, siendo una primera aproximación al tema.

Plpgsql Trigger y Stored Procedure
Introducción

Las bases de datos activas incorporan principalmente 2 tipos de objetos a las BD Pasivas:

- **Trigger**

```
CREATE TRIGGER Actualizo_Clientes . . .
```

- **Stored Procedure y/o Funciones:** Nos permiten implementar programas especificos en el servidor BD.

```
Select radio,  superficie(radio)  from circulos;
```

```
CREATE FUNCTION superficie(p_radio double precision)
RETURNS double precision  AS
$BODY$
begin
        return pi() * p_radio ^ 2;
end
$BODY$
  LANGUAGE plpgsql VOLATILE;
```

Plpgsql Trigger y Stored Procedure
Introducción

Las circunstancias de ejecución de estos tipos de objetos es completamente diferente:

- **Trigger:**

NUNCA se ejecutan explicitamente, se ejecutan automaticamente cuando se produce el evento asociado.

- **Stored Procedure:**

Se EJECUTAN explicitamente por una invocación directa como cualquier función, o como parte de un select, o invocado por otro SP.

Nota: *Un trigger puede invocar un Stored Procedure para realizar la acción completa o parte de ellas.*

Plpgsql Trigger y Stored Procedure
Lenguajes

Lenguages para **Trigger** y **Stored Procedure**:

- La codificación se realiza en una sintaxis propietaria formada por las DMLs y un conjunto de extensiones que permiten: definir variable, control-flujo, manejo de errores, enviar información a quien lo invoco (parámetros de salida).
- Estos son siempre del tipo Procedural/Imperativa, similar a c, java o pascal.

En PostgreSQL	***PL/PgSQL***	En firebird	***PSQL***.
En Sql server	***Transact SQL***	En Oracle	***PL/SQL.***

Problema: *Escasa o Nula estandarización.* → ***PROBLEMA***

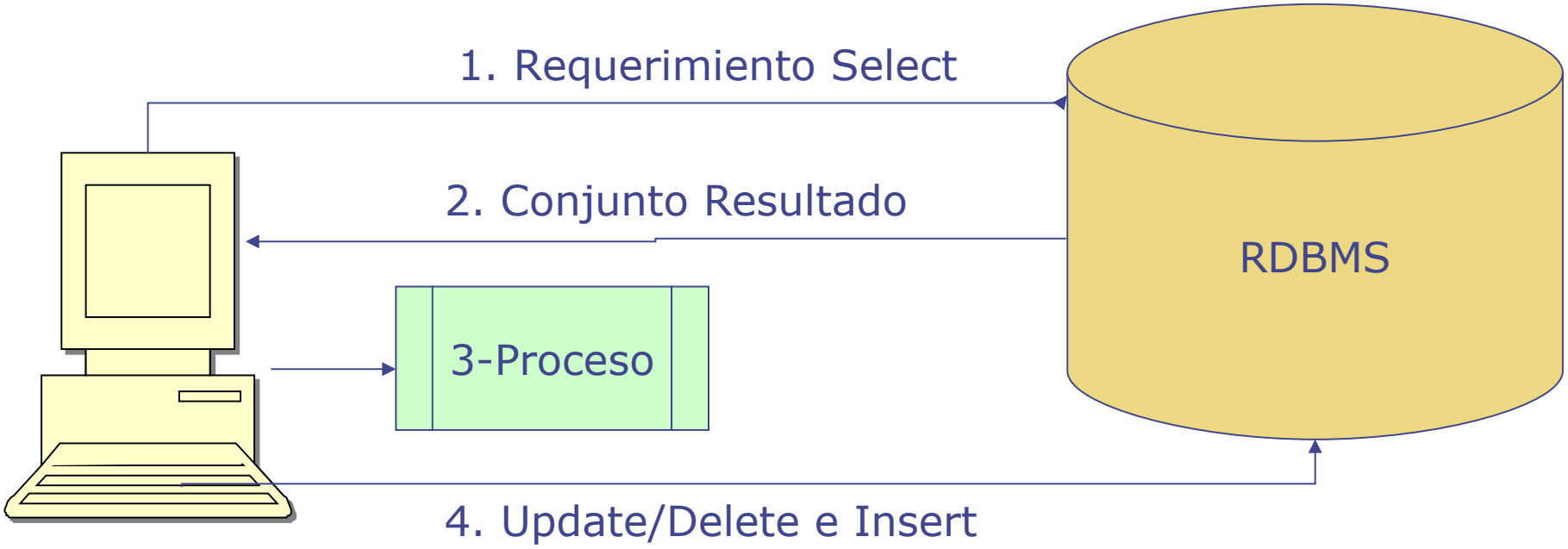
RESUMEN

- Introducción
- Funciones o Stored Procedure
- Triggers

Plpgsql Trigger y Stored Procedure

Stored Procedure – Como nacieron?

Hipótesis: Proceso actualización masivo típico.



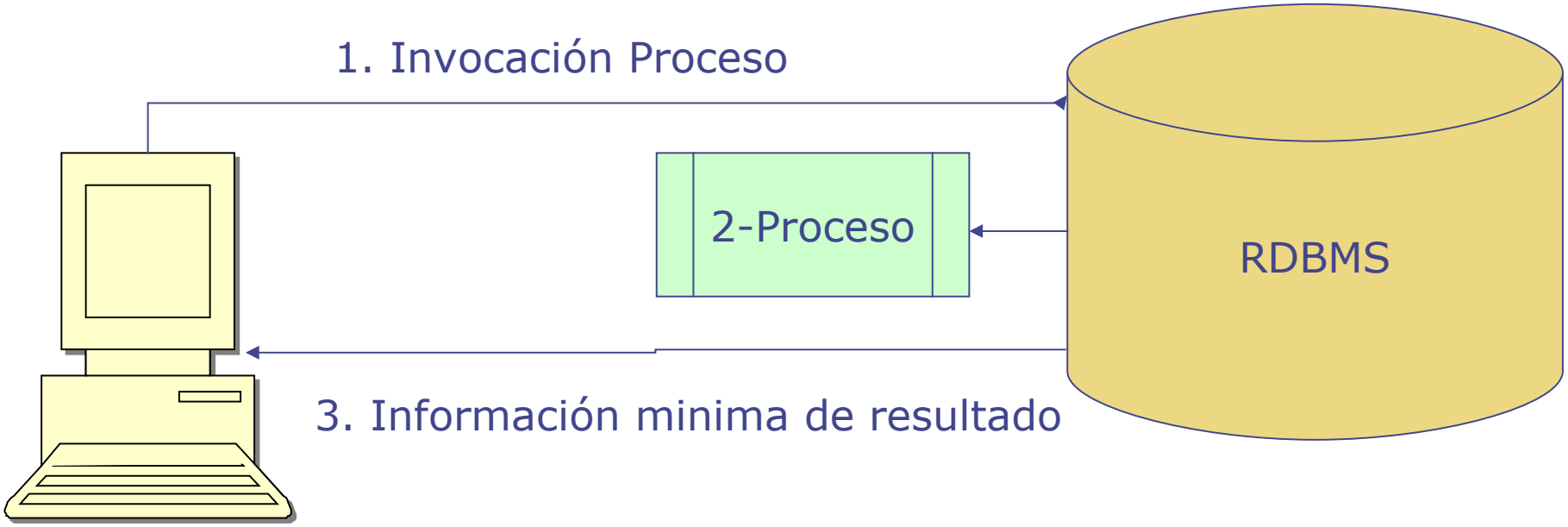
Que punto conlleva mas demora

Que punto se puede evitar

Plpgsql Trigger y Stored Procedure

Stored Procedure – Como nacieron?

Solución: Proceso actualización masivo típico.



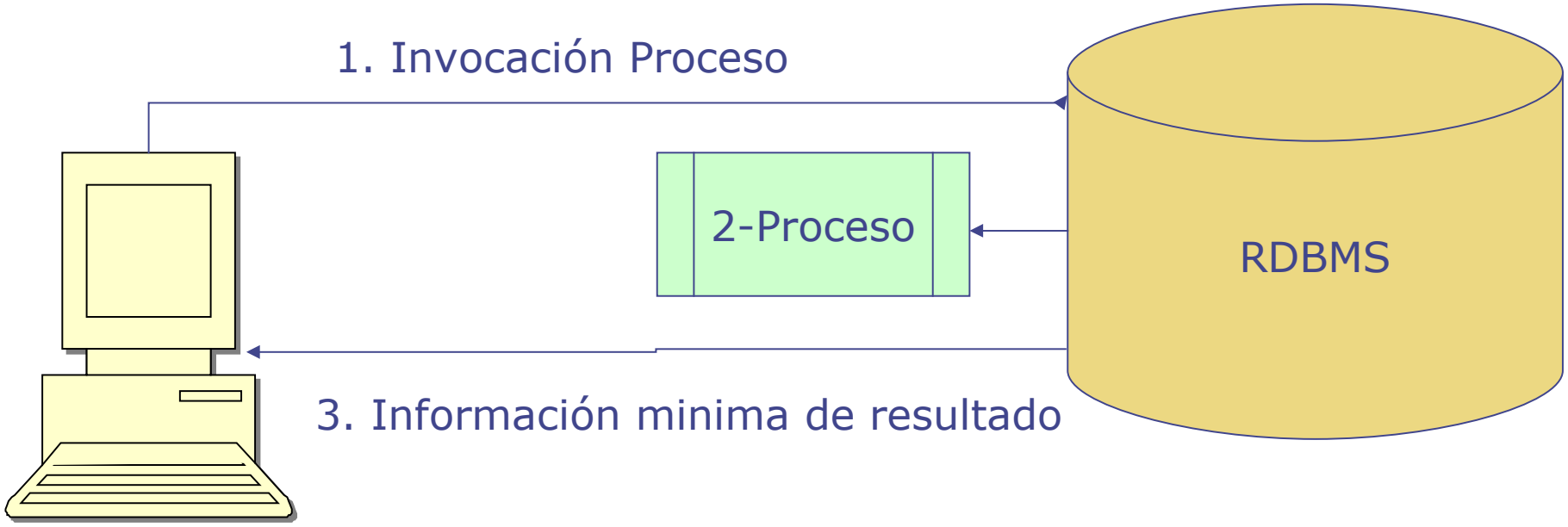
Ventajas

Desventajas

Plpgsql Trigger y Stored Procedure

Stored Procedure – Como nacieron?

Solución: Proceso actualización masivo típico.



Ventajas Evita el costo de doble transferencia.

Desventajas Se duplica código con la plataforma de prog, usa recursos del servidor de BD.

Plpgsql Trigger y Stored Procedure

Stored Procedure **Casos de Uso Tipico**

- Procesos masivos, periódicos o no.
- Producción de salidas muy particulares para herramientas externas que poseen escasas posibilidades de programación (generador de reportes, etc).
- Algoritmos muy utilizados (pueden tomarse estos casos como funciones).
- Lógica de negocio que se utiliza en múltiples interfaces (web, windows Form, consola, etc).

Plpgsql Trigger y Stored Procedure

Stored Procedure **Caracteristicas**

Poca standarizacion: Encabezado semi-standard, cuerpo y lenguaje propio de cada RDBMS.

Invocacion: Un SP siempre se invoca explicitamente.

Centralizacion: Código implementado en el esquema independiente de las aplicaciones.

SP Internos: Los rdbms usan esta tecnología para diversas cuestiones de mantenimiento (compilacion de objetos), (reconstruccion de indices),etc.

Parametros: Un SP puede recibir parametros de entrada y/o de salida o puede no tener entradas ni generar salidas.

Importancia en Select: Se pueden combinar con tablas.

Perfomance: . Al evitar el tiempo de transferencia mejoran procesos de transformación de datos y persistencia de las transformaciones.

Plpgsql Trigger y Stored Procedure

Stored Procedure - **Sintaxis**

Un SP esta formado por:

Encabezado:

Nombre

Parámetros de entrada

Parámetro de salida

```
CREATE FUNCTION superficie(p_radio
   double precision)
   RETURNS double precision  AS
```

Cuerpo:

Programa en si.

```
$BODY$
   begin
        return pi() * p_radio ^ 2;
   end
$BODY$
```

Lenguaje →

```
LANGUAGE plpgsql;
```

Plpgsql Trigger y Stored Procedure

Stored Procedure - **Sintaxis**

Un SP esta formado por:

Encabezado:

Nombre

Parámetros de entrada

Parámetro de salida

```
CREATE FUNCTION superficie(@radio
   double precision)
   RETURNS double precision  AS
```

Cuerpo:

Programa en si.

```
$BODY$
   begin
    return pi() * power(@radio,2);
   end
$BODY$
```

Lenguaje → **Versión Transact-Sql;**

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Un SP esta formado por:

Encabezado:

Nombre

Parámetros de entrada

Parámetro de salida

```
CREATE FUNCTION superficie(p_radio
   double precision)
   RETURNS double precision  AS
```

Cuerpo:

Programa en si.

```
$BODY$
   begin
        return pi() * p_radio ^ 2;
   end
$BODY$
```

Lenguaje →

```
LANGUAGE plpgsql;
```

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Un SP esta formado por:

Encabezado:

Nombre

Parámetros de entrada

Parámetro de salida

```
CREATE FUNCTION superficie(@radio
   double precision)
   RETURNS double precision  AS
```

Cuerpo:

Programa en si.

```
$BODY$
   begin
    return pi() * power(@radio,2);
   end
$BODY$
```

Lenguaje → **Versión Transact-Sql;**

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Nombre:

Hasta 64 caracteres (letras, Nros y "_")

Deben comenzar con una letra.

No pueden ser palabras reservadas.

Para nuestros caso

→ Superficie

```
CREATE FUNCTION superficie(p_radio
   double precision)
   RETURNS double precision  AS
$BODY$
   begin
        return pi() * p_radio ^ 2;
   end
$BODY$

LANGUAGE plpgsql;
```

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Parametros Entrada:

Conjunto de pares
nombre y tipo dato.

- Hasta 64 caracteres (letras, Nros y "_")
- Deben comenzar con una letra.
- No pueden ser palabras reservadas.

Para nuestros caso

→ p_radio double pre..

```
CREATE FUNCTION superficie
    (p_radio double precision)
    RETURNS double precision  AS
$BODY$
    begin
        return pi() * p_radio ^ 2;
    end
$BODY$

LANGUAGE plpgsql;
```

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Parametros Salida:

Conjunto de pares
nombre y tipo dato.

```
CREATE FUNCTION superficie
   (p_radio double precision)
   RETURNS double precision  AS
$BODY$
   begin
        return pi() * p_radio ^ 2;
   end
$BODY$

LANGUAGE plpgsql;
```

Para nuestros caso

→ double precision

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Firma: Se denomina de esta manera a la tripla:

- nombre
- lista de parametros de entrada
- parametro de salida

Para el caso

```
CREATE FUNCTION mayor(nro1 integer, nro2 integer)
 RETURNS integer AS
$body$
  begin
        if (nro1 > nro2) then return nro1;
           else return nro2;
         end if;
  end
$body$      LANGUAGE plpgsql VOLATILE;
```

La firma es

{ mayor, {integer,integer}, integer }

Nota: *La firma no se repite dentro de cada esquema →* ***Sobrecarga***

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Referencias Genéricas de Argumentos:

Los argumentos se pueden referenciar por su nombre o por **$i**, donde **i** es la posición, **$1** primer arg.

```
CREATE FUNCTION mayor(integer, integer) RETURNS integer AS
$body$
 begin
  if ($1 > $2) then
   return $1;
  else
   return $2;
  end if;
 end
$body$              LANGUAGE plpgsql VOLATILE ;
```

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Return / Returns:

- Returns define el tipo de dato del SP.

  [RETURNS ***rettype*** **| RETURNS** ***tabletype*** **]**

- Return termina la ejecución devolviendo el valor al objeto que lo llamo.

```
CREATE FUNCTION mayor(integer, integer) RETURNS integer AS
$body$
 begin
  if ($1 > $2) then
   return $1;
  else
   return $2;
  end if;
 end
$body$              LANGUAGE plpgsql VOLATILE ;
```

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Return / Returns:

- Returns define el tipo de dato del SP.

  [RETURNS *rettype* | RETURNS *tabletype*]

- Return termina la ejecución devolviendo el valor al objeto que lo llamo.

```
CREATE FUNCTION mayor(@numero1 int, @numero2 int)
    RETURNS int
 AS
  BEGIN
   DECLARE @aux int
     IF @numero1 > @numero2
             SET @aux =  @numero1
       ELSE  SET @aux = @numero2
     RETURN @aux      ← en Transact-SQL debe ser la última Linea
  END
```

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Bloques: PL/pgSQL es un lenguaje orientado a bloques.

```
[<<label>>]
[DECLARE
    declarations]
BEGIN
    statements
END;
```

Puede haber cualquier numero de subbloques.

Las variables declaradas en la sección de declaraciones se inicializan a su valor por defecto cada vez que se inicia el bloque, y no cada vez que se realiza la llamada a la función.

Nota: *Declarar variables en realidad es opcional, y las variables declaradas aquí son locales al bloque y a sus subbloques.*

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Bloques: Transact-SQL admite bloques para if, while, etc. De hecho el cuerpo se define por el bloque mas exterior.

```
BEGIN
[DECLARE          declarations]
    statements
END;
```

Puede haber cualquier numero de subbloques.

En transact-SQL la sentencia declare permite declarar dentro de cada bloque las variables temporales a usar.

Nota: *Declare es opcional, y las variables declaradas aquí son locales al bloque y a sus subbloques.*

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Declaración de variables locales, tipo valor, inicialización y default:

declarations

[declaracion_de_variables]

Ej:

```
vsalida                 varchar(256);
e                       doble precision = 2.718281;
vcantidad_ordenes       integer default 4;
vid                     clientes.id%TYPE;
```

Nota: *`clientes.id%TYPE` significa que la variable vid asumirá la definición de la columna `id` de la tabla `clientes`.*

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Declaración de variables locales, tipo valor ~~y default,~~ para asignación se utiliza "=":

Los nombres de variable siempre compienzan con **@**.

declarations

[declaracion_de_variables]

Ej:

```
DECLARE    @var1 int = 0, @clase varchar(20)
```

Las distintas variables se separan por coma.

Nota: *~~clientes.id%TYPE significa que la variable vid asumira la definición de dominio de la columna id de la tabla clientes.~~*

Nota: *DECLARE @cursorcliente AS cliente; --tabla clientes*

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Tipos de Datos en PlpgSql.

Los tipos de una variable pueden ser cualquiera de los tipos básicos existentes en la base de datos.

- Postgres-basetype ← Unica sintaxis admitida en SqlServer
- variable%TYPE
- tabla.atributo%TYPE

.

Variable es el nombre de una variable, previamente declarada en la misma función, que es visible en este momento.

Nota: *tabla.atributo%TYPE significa que la variable asumirá la definición de dominio de la atributo de la tabla.*

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

SINTAXIS DECLARE COMPLETA(I)

Todas las variables, filas y columnas que se usen en un bloque o subbloque ha de ser declarado en la sección de declaraciones del bloque, excepto las variables de control de bucle en un bucle FOR que se itere en un rango de enteros. Los parámetros dados a una función PL/pgSQL se declaran automáticamente con los identificadores usuales, $n. Las declaraciones tienen la siguiente sintaxis:

name **[CONSTANT] <*typ*> [NOT NULL] [DEFAULT | := *value*];**

Esto declara una variable de un tipo base especificado. Si la variable es declarada como CONSTANT, su valor no podrá ser cambiado. Si se especifica NOT NULL, la asignación de un NULL producirá un error en tiempo de ejecución. Dado que el valor por defecto de todas las variables es el valor NULL de SQL, todas las variables declaradas como NOT NULL han de tener un valor por defecto.

El valor por defecto es evaluado cada vez que se invoca la función. Así que asignar '*now*' a una variable de tipo *datetime* hace que la variable tome el momento de la llamada a la función, no el momento en que la función fue compilada a bytecode.

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

SINTAXIS DECLARE COMPLETA(II)

name **RECORD;**

Los registros son similares a los tipos de fila, pero no tienen una estructura predefinida. Se emplean en selecciones y bucles FOR, para mantener una fila de la actual base de datos en una operación SELECT. El mismo registro puede ser usado en diferentes selecciones. El acceso a un campo de registro cuando no hay una fila seleccionada resultará en un error de ejecución.

name **tabla%ROWTYPE;**

Esto declara una fila con la estructura de la clase/tabla indicada. La clase ha de ser una tabla existente, o la vista de una base de datos. Se accede a los campos de la fila mediante la notación de punto. Los parámetros de una función pueden ser de tipos compuestos (filas de una tabla completas). En ese caso, el correspondiente identificador $n será un tipo de fila, pero ha de ser referido usando la orden ALIAS que se describe más adelante. Solo los atributos de usuario de una fila de tabla son accesibles en la fila, no se puede acceder a Oid o a los otros atributos de sistema (dado que la fila puede ser de una vista, y las filas de una vista no tienen atributos de sistema útiles).

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

SINTAXIS DECLARE COMPLETA(III)

name ALIAS for $n;
Para una mejor legibilidad del código, es posible definir un alias para un parámetro posicional de una función.
Estos alias son necesarios cuando un tipo compuesto se pasa como argumento a una función.

Ejemplos: ej: `teoriaSP_TiposDatos(integer);`

```
itb integer = 1;          --Tipo Basico

ivt ri%type = 2;          --Tipos Variable%Type

ict facturas.numero%type = 3;   --Tipos de datos: class.field%type

vtipo facturas.tipo%type;       --Tipos de datos: class.field%type

iparent alias for $1;            --Declare alias

rtfacturas facturas%rowtype;     --Declare row%type

rrfacturas record;               --Declare record
```

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Bloques Anonimos**

Herramienta Practica en **PlpgSql**.

Una forma practica de usar y probar la sintaxis es utilizar **bloques anonimos** de ejecución .

```
DO $$
DECLARE
  vlibuni alumnos.libuni%TYPE;
  vnota dnota;
BEGIN
   SELECT libuni FROM alumnos INTO vlibuni;
   vnota = 8;
   RAISE notice 'el nro de libreta: % obtuvo: %',vlibuni,vnota;
END$$;
```

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Comentarios : PL/pgSQL admite 2 tipos de comentarios.

Tipo 1:

```
 a= 12;    -- Tipo: Hasta fin de linea
```

Tipo 2: **No disponible el Transact-sql**

```
/* Tipo Comentario de bloque
   acá inicializamos variables String
*/
```

Los bloques de comentarios no pueden anidarse pero comentarios de guiones pueden encerrarse en un bloque de comentario.

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Asignación: directa y por select.

Directas:

```
i  = 12;     -- asigna el valor 12 a la var i;
y := 2 * x; -- asigna a y el doble de x
```

Por Select:

```
/* asigna a la variable record rclientes la
ejecución del select
Rclientes clientes%rowtype; */
Select * into rclientes from clientes limit 1;
```

Nota: *la asignación directa se puede hacer con el operador igual o con dos puntos igual.* *Ej: TeoriaSp_tiposdatos(int);*

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Asignación: directa y por select.

Directas:

```
SET i  = 12     -- asigna  12 a la var i;
SET y  = 2 * x -- asigna a y el doble de x
```

Por Select:

```
/* asigna a la variable record rclientes la
ejecución del select
Asignación desde un select */
  SELECT @stock  = stock    FROM productos AS p
 WHERE codigo = @codigo;
```

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Devolviendo valores en PL/pgSQL.

```
CREATE FUNCTION teorioSP_call1(integer)
RETURNS integer AS
$BODY$
  return $1;
$BODY$
LANGUAGE plpgsql VOLATILE;
```

La función termina y el valor de expresión se devolverá al ejecutor superior (llamador).
Si el control alcanza el fin del bloque de mayor nivel de la función sin encontrar una sentencia RETURN, ocurrirá un error de ejecución.

Ej en: teoriaSP_call1()

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Devolviendo valores en PL/pgSQL.

```
CREATE FUNCTION teorioSP_call1(integer)
RETURNS integer AS
$BODY$
  return $1;
$BODY$
LANGUAGE plpgsql VOLATILE;
```

La función termina y el valor de expresión se devolverá al ejecutor superior (llamador).
Si el control alcanza el fin del bloque de mayor nivel de la función sin encontrar una sentencia RETURN, ocurrirá un error de ejecución.

Ej en: teoriaSP_call1()

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Hay tres tipos de llamadas a funciones en PL/pgSQL.

```
perform teoriaSP_call(2);   -- descarta el resultado

i = teoriaSP_call(4); --recupera en i el valor
retornado

select teoriaSP_call(5) into i;     --idem anterior
```

Perform pierde el retorno generado por la función llamada.

Los otros casos lo recuperan en i.

```
Ej en: teoriaSP_call()
```

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Devolviendo valores en Transact-SQL

Ejemplo:

```
Create table productos (codigo int not null primary key,
  nombre varchar(50),   stock numeric(10,2));
```

```
CREATE PROCEDURE stockproductos
@codigo integer,
@stock numeric(15,2) OUTPUT
AS      SELECT @stock  = stock
          FROM productos AS p
         WHERE codigo = @codigo;
RETURN
GO
```

```
DECLARE @productosstock numeric(15,2);
EXECUTE dbo.stockproductos  10,@stock = @productosstock OUTPUT;
-- Muestra el valor retornado
PRINT 'stock del producto: ' + cast(@productosstock as varchar(20));
GO
```

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

IF: Condicionales simples en PL/pgSQL.

```
IF expresion THEN
  sentencias
[ELSIF   expresion THEN sentencias]
[ELSE   sentencias]
END IF;
```

expresión tiene que ser booleano

Ej en: `teoriaSP_divide(float, float)`

Nota: Ver la particularidad de ELSIF.

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

IF: Condicionales simples en Transact-SQL.

```
IF expresion
  sentencias
[ELSEIF   expresion
   sentencias]
[ELSE   sentencias]
```

expresión tiene que ser booleano

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Case: If multiples formato SIMPLE en PL/pgSQL.

```
CASE
   WHEN expression [, expresion […]] THEN statements
 [ WHEN expression [, expresion […]] THEN statements]
 [ ELSE statements ]
END CASE;
```

La forma simple de CASE proporciona ejecución condicional basada expresiones booleanas.

Ej en: teoriaSP_nombredia(date)

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Case: If multiples formato SIMPLE en Transact-SQL.

```
CASE
   WHEN expression [, expresion […]] THEN statements
 [ WHEN expression [, expresion […]] THEN statements]
 [ ELSE statements ]
END CASE;
```

La forma simple de CASE proporciona ejecución condicional basada en expresiones booleanas.

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Select Simple en PL/pgSQL: Cualquier sentencia de resultado cardinalidad = 1, sino toma el primero.

```
SELECT estado  INTO v_estado
  FROM PRODUCTOS
 WHERE codigo = 1;
```

Nota: Debe tener una clausula INTO con un juego de variables acorde a las columnas proyectadas de manera de asignar el resultado a variables validas PlpgSQL.

Ej en: teoriaSP_TiposDatos(integer)

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Select Simple en PL/pgSQL: Cualquier sentencia de resultado cardinalidad = 1, sino toma el primero.

```
SELECT @estado = estado
  FROM PRODUCTOS
 WHERE codigo = 1;
```

Nota: Debe tener asignación por cada elemento proyectado de manera de copiar el resultado a variables validas Transact-SQL.

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Cursor Select en PL/pgSQL: Para cualquier sentencia SQL.

```
FOR variableCursor IN SELECT cantidad, precio
                      FROM PRODUCTOS
  LOOP
      sentencias…
  END LOOP;
```

Ej en: teoriaSP_facturas()

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Cursor Select en Transact-SQL: Para cualquier sentencia SQL.

```
DECLARE vendor_cursor CURSOR FOR
   SELECT VendorID, Name
     FROM Purchasing.Vendor
    WHERE PreferredVendorStatus = 1;

OPEN vendor_cursor

FETCH NEXT FROM vendor_cursor INTO @vendor_id, @vendor_name

WHILE @@FETCH_STATUS = 0
BEGIN
--    Aca estan disponibles @vendor_id, @vendor_name...
      FETCH NEXT FROM vendor_cursor
            INTO @vendor_id, @vendor_name
END
```

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Actualizaciones Cualquier sentencia de actualización.

```
DELETE FROM CLIENTES WHERE nro = v_cliente;

UPDATE productos SET estado = 'sin stock' where stock
<= v_stock_minimo;

INSERT INTO pedido_proveedores (producto_id,cantidad)
 VALUES (v_id, 1000);
```

- Sin Ejemplo

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Actualizaciones Cualquier sentencia de actualización.

```
DELETE FROM CLIENTES WHERE nro = @cliente;

UPDATE productos SET estado = 'sin stock' where stock
<= @stock_minimo;

INSERT INTO pedido_proveedores (producto_id,cantidad)
 VALUES (@id, 1000);
```

Sin Ejemplo

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Actualizaciones Actividad

```
CREATE TABLE parametros
                (id serial,                  nombre varchar(60),
                  valor varchar(60),  tipo varchar(30))
```

```
Cargue una fila con nombre     =  'Nota Minima de aprobación',
                    Nota       = '6',
                    Parametro = 'integer'
```

Desarrolle un sp llamado ejsp_dml que permita cambiar el valor de nota para el parámetro cargado.

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

LOOP define un bucle incondicional que se repite indefinidamente hasta que sea terminado por una instrucción EXIT o RETURN.

```
[<<label>>]
LOOP
  statements
END LOOPS [<<label>>];
```

Las etiquetas opcionales pueden ser usadas para salir y continuar los bucles anidados para especificar qué loop de esas declaraciones se refieren.

```
EXIT [ label ] [ WHEN expresion];
CONTINUE [ label ] [ WHEN expression];
  - para todos los tipos de bucles
```

Ej en: teoriaSP_loops(date)

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

While: La instrucción WHILE repite una secuencia de instrucciones mientras la expresión booleana se evalúa como verdadera. La expresión se comprueba justo antes de cada entrada en el cuerpo del bucle.

```
[ <<label>> ]
WHILE expresion LOOP
   statements
END LOOP [label];
```

Sin ejemplo

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

While: La instrucción WHILE repite una secuencia de instrucciones mientras la expresión booleana se evalúa como verdadera. La expresión se comprueba justo antes de cada entrada en el cuerpo del bucle.

```
WHILE expresion
    statement
```

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

En **LOOP, WHILE, FOR** y las declaraciones **FOREACH** se pueden usar para repetir una serie de comandos.

Internamente puede usar:

CONTINUE: finaliza la iteración corriente.

EXIT: Sale del bucle.

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Ej **FOREACH** con **LOOP**.

```
CREATE FUNCTION sum(int[]) RETURNS int8 AS $$
DECLARE
  s int8 := 0;
  x int;
BEGIN
  FOREACH x IN ARRAY $1
  LOOP
    s := s + x;
  END LOOP;
  RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

En **WHILE** se puede repetir una serie de comandos.

Internamente puede usar:

CONTINUE: finaliza la iteración corriente.

BREAK: Sale del bucle.

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

While

Actividad

Desarrolle un sp o función llamado ejsp_ultimodiames de tipo integer que reciba un nro de mes y devuelva el último día del mes (de un año no vicie-sto).

Desarrolle versión sobrecargada de ejsp_ultimodiames que reciba ademas el año y devuelva el último día de ese mes. (ejsp_ultimodiames2 para sqlserver).

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

FOR: La instrucción FOR se usa cuando hay una cantidad de repeticiones conocida de antemano. Existen varios formatos, vamos al primero:

```
[ <<label>> ]
FOR name IN [ REVERSE ]
   expression .. expression [ BY expression ] LOOP

statements
END LOOP [label];
```

Ejemplo

```
FOR i IN REVERSE 10..1 BY 2 LOOP
  -- i tomará los valores: 10,8,6,4,2
END LOOP;
```

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

FOR query: La instrucción FOR desde un query:

```
[ <<label>> ]
FOR target IN query LOOP
  statements
END LOOP [label];
```

Ejemplo

```
FOR rec_clientes IN
(SELECT cuit, nombre FROM clientes)  LOOP
  -- por cada fila del select estará disponible
  -- rec_clientes.cuit y rec_clientes.cuit ;
END LOOP;
```

-- Nota: rec_clientes se define en declare *rec_clientes record;*

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

FOR - IN - EXECUTE: Bucles a dinámicos.

```
[ <<label>> ]
FOR target
 EXECUTE expresion_string [USING expression [,
... ]]
  statements
END LOOP [label];
```

Similar a la forma anterior, excepto que la consulta de origen se especifica como una expresión de cadena que se evalúa y hace un nuevo planquery en cada entrada al bucle FOR.

FOR-IN-EXECUTE

→ hace un plan query cada vez que se ejecuta

→ +++ flexibilidad --- velocidad

FOR-IN-QUERY

→ hace un plan query cuando se compila por primera vez

→ +++ velocidad --- flexibilidad

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

El FOR-IN-EXECUTE iteraciones dinámicas:

```
[ <<label>> ]
FOR target   EXECUTE expresion_string [USING
expression [, ... ]] LOOP
 statements
END LOOP [label];
```

Ejemplo

```
DO $$
  DECLARE
      tabla  VARCHAR DEFAULT 'empleados';
      cur RECORD;
  BEGIN
    FOR cur IN    EXECUTE
     'SELECT * FROM '||tabla
     LOOP
          RAISE NOTICE 'tratando: %',cur.dni;
      END LOOP;
  END; $$ LANGUAGE plpgsql;
```

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

El FOR-IN-ARRAY a través de un **array**:

```
[ <<label>> ]
FOREACH target  IN ARRAY expresion LOOP
 statements
END LOOP [label];
```

Ejemplo

```
CREATE FUNCTION sum(int[]) RETURNS int8 AS $$
DECLARE
 s int8 := 0;        x int;
BEGIN
  FOREACH x IN ARRAY $1
  LOOP
      s := s + x;
  END LOOP;
  RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

RETURN NEXT Envía una nueva fila al proceso invocador .con los contenidos asignados a los atributos de salida (table).

```
RETURNS
TABLE(rdetalle varchar(40), ri integer) AS
...
    rdetalle = 'Salida (' || i || '):';
    ri := cfacturas.numero;

   RETURN NEXT;
```

Ej: prácticamente todos

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Invocar un SP o Función desde un select: Un SP se puede combinar en un select:

- Como una tabla (con algunas restricciones).
- En Proyección
- En Restricción

```
SELECT * FROM SPFACTURASVIP('2009/01/01','2009/06/30');
SELET NUMERO, FACTURA, SPPERANT(PERIODO) FROM FACTURAS;
SELET * FROM FACTURAS WHERE PERIODO = SPPERANT(PERIODO);
```

Plpgsql Trigger y Stored Procedure

Ayuda Funciones y Operadores

Documentación Oficial:

http://www.postgresql.org/docs/9.3/static/functions.html

Chapter 9. Functions and Operators

Table of Contents

9.1. Logical Operators

9.2. Comparison Operators

9.3. Mathematical Functions and Operators

9.4. String Functions and Operators

Ej: http://www.postgresql.org/docs/9.3/static/functions-string.html

9.5. Binary String Functions and Operators

9.6. Bit String Functions and Operators

9.7. Pattern Matching

9.8. Data Type Formatting Functions

9.9. Date/Time Functions and Operators

Plpgsql Trigger y Stored Procedure

Ayuda Funciones y Operadores

Documentación Oficial:

http://www.postgresql.org/docs/9.3/static/functions.html

Chapter 9. Functions and Operators

Table of Contents

9.10. Enum Support Functions

9.11. Geometric Functions and Operators

9.12. Network Address Functions and Operators

…..

…..

9.27. Trigger Functions

9.28. Event Trigger Functions

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

RAISE: Envía información de Mensajes y Errores.

```
RAISE [ level ] 'format' [, expression [, ... ]] [ USING
option = expression [, ... ] ];
RAISE [ level ] condition_name [ USING option = expression
[, ... ] ];
RAISE [ level ] SQLSTATE 'sqlstate' [ USING option =
expression [, ... ] ];
RAISE [ level ] USING option = expression [, ... ];
RAISE ;

Ej: RAISE NOTICE '6:Terminación el proceso: %', v_cantidad;
```

% se reemplaza por el argumento correspondiente a la posición.

Los niveles ***level*** permitidos son DEBUG, LOG, INFO, NOTICE, WARNING y EXCEPTION(default).

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Abortando la ejecución en PL/pgSQL.

```
RAISE EXCEPTION
    for'' [, identifier [...]];

/* Ejemplo concreto */
if divisor = 0 then
   raise 'Error intento de división por cero';
end if;
return dividendo / divisor;
```

RAISE que puede enviar mensajes al sistema de registro de Postgres

Ej en: teoriaSP_divide(float, float)

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

Procesos masivos, periódicos o no.

Producción de salidas muy particulares para herramientas

Externas con pocas posibilidades de programación (generador de reportes, etc).

Algoritmos muy utilizados (pueden tomarse estos casos como funciones)

Lógica de negocio que se utiliza en múltiples interfaces (web, windows Form, consola, etc).

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Sintaxis**

DROP FUNCTION name

Elimina el SP cuyo nombre es name

Ej:

DROP FUNCTION sp_calcula_intereses_cc();

Recordar:

CREATE OR REPLACE FUNCTION

** Muy Practico **

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Ejemplo**

1) Genere una función que reciba como parámetro los lados de un triangulo y calculo su superficie.

2) Genere una función que reciba como parámetro un string con el apellido/s y nombre/s y genere una abreviatura formada por el apellido y la inicial del primer nombre. El apellido y nombre se encuentra separado por una coma.

Ej: recibe “PEREZ DE CUELLAR, JUAN ANDRES”

genera “PEREZ DE CUELLAR J.”

3) Genere una función que reciba un nombre de tabla y devuelva cuantas filas posee. Nota: no usar funciones internas, realizar la resolución con Loop o algún otro bucle.

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Ej Retorno Tabla**

```
CREATE or REPLACE FUNCTION dup2(int)
  RETURNS TABLE(f1 int, f2 varchar)
     AS $$ SELECT $1,
  cast('20170446667' as varchar) $$
     LANGUAGE SQL;


-- Prueba
SELECT * FROM dup2(42) d join clientes
  c on c.cuit = f2;
```

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Ejemplo**

```
CREATE OR REPLACE FUNCTION foo(a int)
 RETURNS TABLE(b int, c int) AS $$
 BEGIN
   RETURN QUERY SELECT i, i+1 FROM
generate_series(1, a) g(i);
 END;
 $$ LANGUAGE plpgsql;
 Call:


SELECT * FROM foo(10);
```

Plpgsql Trigger y Stored Procedure

Stored Procedure – **Ejemplo**

```
CREATE OR REPLACE FUNCTION obtenerclientes()
  RETURNS TABLE(rcuit varchar(11), rnombre varchar(40)) AS
$BODY$
declare
   record_clientes record;
begin
     FOR record_clientes IN(SELECT clientes.cuit::varchar(11),
   clientes.nombre::varchar(40)
                  FROM clientes)  LOOP
            rcuit := record_clientes.cuit ; rnombre :=
   record_clientes.nombre;
            if (record_clientes.nombre != 'NOBLES JUAN')  then
   RETURN NEXT; end if;
     END LOOP;
end
$BODY$                  LANGUAGE plpgsql VOLATILE;
```

RESUMEN

- Introducción
- Funciones o Stored Procedure
- Triggers
- Casos de Estudio.

Plpgsql Trigger y Stored Procedure

Trigger – **Sintaxis**

```
CREATE [ CONSTRAINT ] TRIGGER name { BEFORE | AFTER |
          INSTEAD OF } { event [ OR ... ] }
ON table
[ FOR [ EACH ] { ROW | STATEMENT } ]
[ WHEN ( condition ) ]
EXECUTE PROCEDURE function_name


where event can be one of:

  INSERT
  UPDATE [ OF column_name [, ... ] ]
  DELETE
  TRUNCATE
```

Plpgsql Trigger y Stored Procedure

Trigger – **Sintaxis SqlServer**

```
CREATE TRIGGER [ schema_name . ]trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS { sql_statement  [ ; ] [ ,...n ] | EXTERNAL NAME
<method specifier [ ; ] > }

<dml_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clause ]

<method_specifier> ::=
    assembly_name.class_name.method_name
```

Plpgsql Trigger y Stored Procedure

Trigger – **Sintaxis (II)**

Tipos de Trigger – Tiempo de Ejecución

Before → Antes de que se realice a la actualización

After → Despues de que se realice a la actualización

Instead of → En lugar de… (Practico para vistas)

Plpgsql Trigger y Stored Procedure

Trigger – **Sintaxis (II) sqlServer**

Tipos de Trigger – Tiempo de Ejecución

For → Realice a la actualización

After → Despues de que se realice a la actualización

Instead of → En lugar de… (Practico para vistas)

Plpgsql Trigger y Stored Procedure

Trigger – **Sintaxis (III)**

Tenemos los trigger

```
Create trigger trg1 BEFORE UPDATE ON tabla FOR EACH ROW
    EXECUTE PROCEDURE fxb1;
Create trigger trg2 AFTER UPDATE ON tabla FOR EACH ROW
    EXECUTE PROCEDURE fxb2;
```

Ante la ejecución de la sentencia:

```
update table set campo1 = 12 where tipo = '8';
    → 3 rows affected;
```

Sucesos:

Por cada una de las tres filas:

- Se activa trg1 → se ejecuta fxb1
- Se produce la actualización del campo1
- Se activa trg2 → se ejecuta fxb2

Plpgsql Trigger y Stored Procedure

Trigger – Acciones → **Funciones**

Funciones: El procedimiento propiamente dicho de la/s acción/es del trigger se implementa en una función de tipo “trigger”.

```
CREATE FUNCTION dec_aux () RETURNS trigger AS
BEGIN
  UPDATE aux SET cont=cont-1 WHERE dmd5 = OLD.dmd5;
  DELETE FROM aux where cont < 1;
  RETURN NULL;
END; '
LANGUAGE 'plpgsql';
```

En Transact-Sql NO SON OBLIGATORIAS

Plpgsql Trigger y Stored Procedure

Trigger – Funciones → **return**

Return: Que debe retornar cada tipo de ocurrencia?

Si el trigger es de ***before*** debe encontrar un return con *new,* si el registro *new* fue modificado.

Si el trigger es de ***after*** debe encontrar un return con null. Situación mas razonable por que la operación ya se hizo. Se reveera luego de new y old.

Returns opaque: Se usaba en versiones anteriores a la 8, fue deprecada.

Concepto solo para plPgSql

Plpgsql Trigger y Stored Procedure
Trigger – Variables de Transición

NEW Nuevos valores de columnas según la sentencias INSERT o UPDATE del trigger.

OLD Valores antiguos de columnas de la fila que disparó el trigger.

- Para DELETE o UPDATE.

Se analizan a continuación.

Plpgsql Trigger y Stored Procedure

Trigger – Variables de Transición

OLD.columna: Valor de la tabla.

NEW.columna: Valor que se modifica en SET[UPDATE]
o se ingresa en VALUES[INSERT]

Ej: Tabla **Personas**

DNI	NOMBRE	FEC_NAC
170	Salas Martín	28-06-1984

```
UPDATE personas SET dni = 17044666 WHERE dni = 170;
```

→ **OLD.DNI**=170 **NEW.DNI** = 17044666

```
DELETE FROM personas WHERE dni = 17044666;
```

→ **OLD.DNI**=1744666 **OLD.NOMBRE**=SALA MARTIN **OLD.FEC_NAC**=28-06-1984

```
INSERT PERSONAS VALUES (18444264,'SA JUAN',null);
```

→ **NEW.DNI**=18444264 **NEW.NOMBRE**='SA JUAN' **NEW.FEC_NAC**=NULL

Plpgsql Trigger y Stored Procedure

Trigger – Variables de Transición

Transact-Sql NO cuenta con New y Old. Cuenta con los conjuntos:

INSERTED Nuevos valores de columnas según la sentencias INSERT o UPDATE del trigger.

DELETED Valores antiguos de columnas de la fila que disparó el trigger. Para DELETE o UPDATE.

Plpgsql Trigger y Stored Procedure

Trigger – Granularidad

FOR EACH ROW Se dispara una instancia por cada fila afectada.

FOR EACH STATEMENT Se dispara una vez por la instrucción que desencadeno el trigger.

Nota: Transact-sql todos los trigger son statement.

Plpgsql Trigger y Stored Procedure

Trigger Polivalentes

Admiten una ***colección de eventos*** separadas por “OR”

. ej: INSERT OR DELETE OR UPDATE

Función para determinar la DML lanzadora:

tg_op devuelve

INSERT

DELETE

UPDATE

→ `trg_poli_clientes y fx_poli_clientes`

Plpgsql Trigger y Stored Procedure
Trigger Polivalentes

Admiten una ***colección de eventos*** separadas por ~~"OR"~~ "," (coma).

. ej: INSERT , DELETE , UPDATE

Para saber si fue disparado por Insert, Delete y Update se tienen las tablas Inserted y Deleted:

```
IF EXISTS(SELECT * FROM inserted)
      and EXISTS(SELECT * FROM deleted)          -- En este caso es Update

IF EXISTS(SELECT * FROM inserted)
     and NOT EXISTS(SELECT * FROM deleted)       -- En este caso es Insert

IF NOT EXISTS(SELECT * FROM inserted)
     and EXISTS(SELECT * FROM deleted)          -- En este caso es Delete
```

- → Ej de versiones anteriores.

Plpgsql Trigger y Stored Procedure

Trigger – Casos de Uso Típico

Propagación de actualizaciones

- TOO: **After**, Acción: actualizar tablas relacionadas.

Manejo de valores de Default. generadores (secuencias) especiales.

- TOO: **Before**, Acción: Modificar estos atributos de new.

Reglas de Validación complejas:

- TOO: **Before**, Acción: invalidar lanzando un excepción.

Manejo de Versiones:

- TOO: **After**, Acción: Insertar en tabla de Historia.

Pistas para auditar actividad:

- TOO: **After**, Acción: Registrar la actividad para análisis posterior.

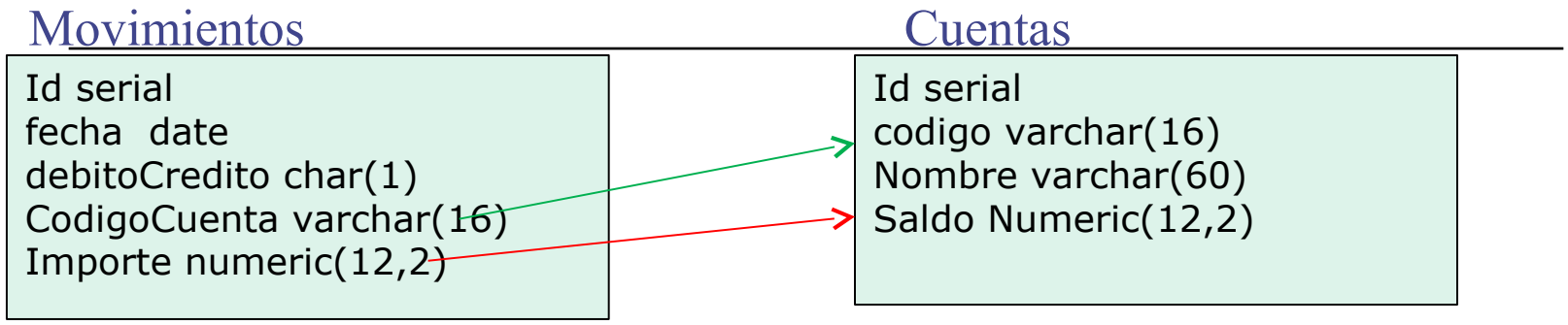
Crear Vistas materializadas (en forma manual)

- TOO: **After**, Acción: actualizar la tabla que implementa la VM.

Hacer actulizable una Vista

- TOO: **Instead of**, Acción: actualizar las tablas subyacentes.

Caso Propagación Actualiza(1)



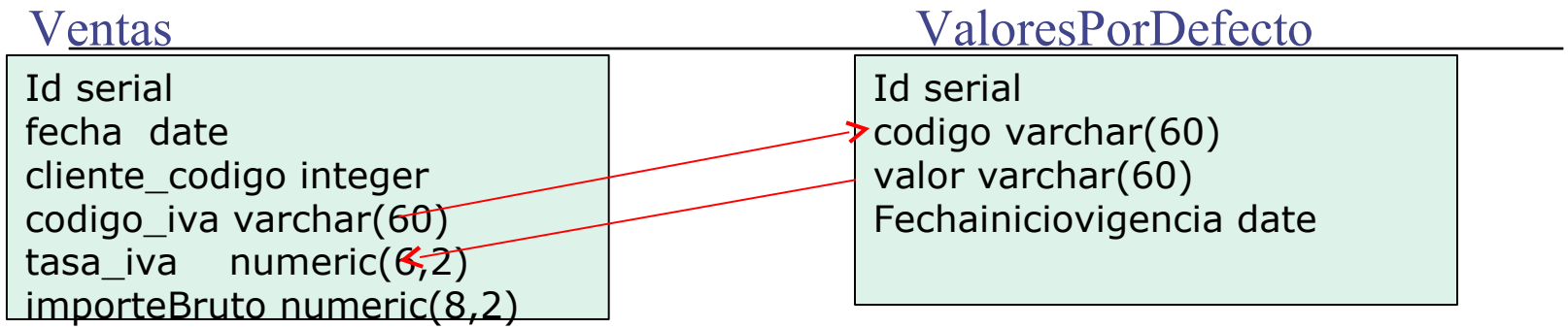
La idea es que cuando se ingresa, modifica o da de baja un Movimiento, se actualice el saldo de la cuenta asociada consecuentemente. Un credito incrementa el saldo y un debito decrementa el saldo.

¿Trigger sobre que tablas crearía?

¿sobre que Eventos?

¿Con sus palabras que haría cada trigger?

Caso Manejo de Valores Default(2)



La idea es que cuando se ingresa una venta se ingrese solo el codigo de iva y con este que se grabe siempre el valor del iva vigente.

```
create trigger trgValDefectoIva
 before insert  on ventas
 for each row
execute procedure fxValDefectoIva();
```

```
create function fxValDefectoIva()
returns trigger AS $$  begin
select cast(valor as numeric(8,2))
  from valorespordefecto
  where codigo = new.codigo_iva
  into new.tasa_iva;

  return new;
end;  $$    language 'plpgsql';
```

→ triggervaloresdefecto

Caso Validaciones Complejas (3)

ClientesTrg

Id serial codigo integer Nombre varchar(40) Cliente_estatal varchar(40)

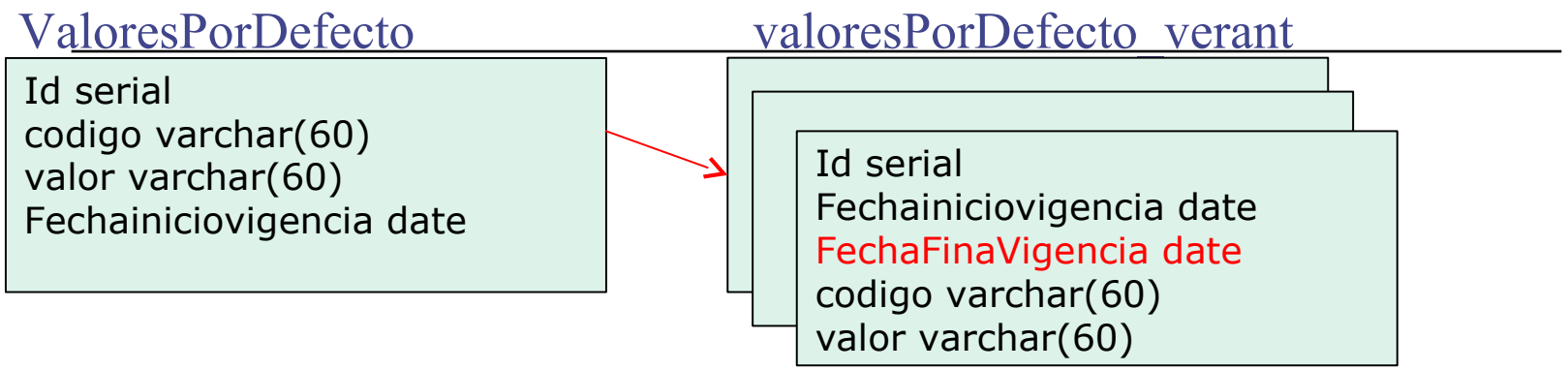
La idea es que un cliente estatal no puede comprar productos importados.

```
create trigger trgValidaEstatal
before insert  on ventas
for each row
execute procedure fxValidaEstatal();
```

```
create function fxValidaEstatal()
  returns trigger AS $$  declare
  begin
   if (new.codigo_iva = 'IvaProdImp‘)
     select cliente_estatal
       from clientestrg
       where codigo=new.cliente_codigo
        into vestatal;
     if vestatal then
        raise exception 'Cliente Est’;

      end if;
    end if;
    return new;
  end;  $$    language 'plpgsql';
```

→ triggervalidaestatal

Caso Manejo de Versiones (4)



La idea es que cada vez que se actualice se guarde la versión anterior.

```
create trigger trgVersionesAnteriores_ins()
 after update or delete
  on valoresPorDefecto
   for each row
execute procedure fxVersionesAnt_ins();
```

```
create function fxVersionesAnt_ins()
  returns trigger AS $$  begin

 -- insert into valpordef_verant
  -- desde transición old
 end;  $$    language 'plpgsql';
```

Solo se muestra el de Insert (el mas Simple). Ver Resto en el script.

→ triggerversionesanteriores

Caso Pistas Auditoría (5)

clienteTrg

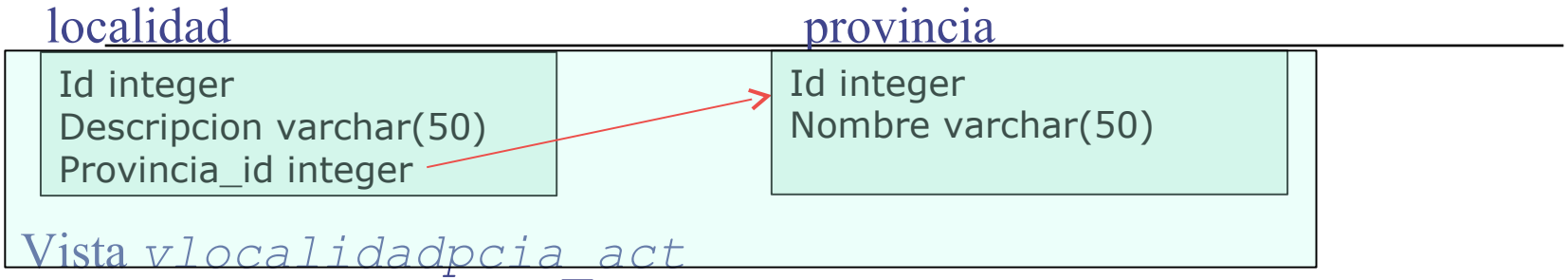
Id serial
codigo integer
Nombre varchar(40)
Cliente_estatal varchar(40)
fecha_alta timestamp
fecha_ultimamodificacion timestamp
usuario_alta varchar(64)
usuario_ultdificacion varchar(64)

La idea es que cada vez que un cliente se agregue o modifique se guarda esta información para seguir la pista.

```
create trigger trgPistaAudita()
 before insert or update on clienteTrg
 for each row
execute procedure fxPistaAudita();
```


```
create function fxPistaAudita()
  returns trigger AS $$  begin
    IF (tg_op = 'INSERT') THEN
       new.f_alta =current_timestamp;
       new.usuario_alta  = user;
    end if;

    IF (tg_op = 'UPDATE') THEN
      …Idem Insert con *modificacion
   END IF;
 end;  $$    language 'plpgsql';
```

Caso Vistas Actualizables (6)



Se cuenta con una vista vlocalidad provincia, la cual se quiere hacer actualizable para que ingresada una provincia o localidad inexistente en insert por ej se propague a la tabla base *create view vlocalidadpcia_act AS SELECT l.codigo_postal, l.localidad, p.id AS provincia_id, p.nombre AS provincia_nombre FROM localidades l JOIN provincias p ON p.id = l.provincia_id;*

```
create trigger trgVistaAct
 instead of insert  on vlocalidadpcia_act
 for each row execute procedure fxVistaAct();
```

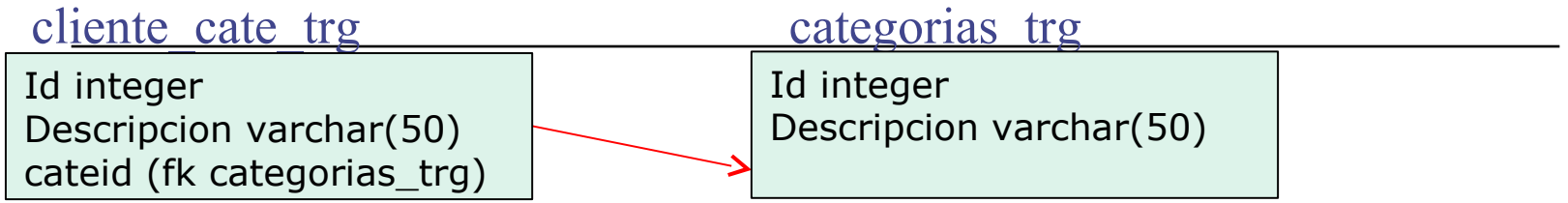
```
create function fxVistaAct()
    returns trigger AS $$  begin
   // inserta loc si no existe

   // codigo completo pag siguiente


    return null;
end;  $$    language 'plpgsql'
```

→ triggervistaact

Caso Vistas Actualizables (6')

```
create function fxVistaAct() returns trigger AS $$
 declare
   cantidad int;
   nombre  localidades.localidad%type;
  begin

   select count(*), max(localidad)                from localidades
   where codigo_postal = new.codigo_postal    into cantidad, nombre;

  if (cantidad = 0 ) then
    insert into localidades (codigo_postal, localidad, provincia_id) values
 (new.codigo_postal, new.localidad, new.provincia_id);
  end if;

   -- Que hacemos si la prov no existe????? DISCUSIÓN

   return null; -- No interesa el valor retornado
end;
$$ language 'plpgsql';
```

Caso Vistas Materializadas (7)



La idea es emular correctamente una vista del tipo

```
create materialized view   vclientecate  as select *
from clientestrg c join categoriastrg t on t.id…
```

vm_cliente_cate_trg →

Cliecodigo integer	catid integer
Clienombre varchar(50)	catdes varchar(50)

Cada vez que se agrega una fila en clientestrg o categoriastrg se debe disparar un trigger que mantenga actualizada la tabla que implementa nuestra vista materializada vm_cliente_cate_trg.

Se explica la idea con clientestrg, mismo concepto aplicar para categoriastrg.

```
create trigger trgVistaMat      after insert
  on clientestrg  for each row
  execute procedure     fxVistaMat();
```

→ trgvistamat

Caso Vistas Materializadas (7')

```
create function fxVistaMat() returns trigger AS $$
   declare
      vestatal bool;
   begin

     insert into    vm_cliente_cate_trg
         select c.codigo, c.nombre,
           t.id,     t.descripcion
           from clientestrg c
           join categoriastrg t on t.id = c.categoria_id
          where c.id = new.id;
--Nota: El trigger es de after se toma la fila recien grabada.

      return null;
   end;
   $$
   language 'plpgsql';
```

Plpgsql Trigger y Stored Procedure
Objetos Asociados

```
CREATE [ TEMPORARY | TEMP ] SEQUENCE name
  [ INCREMENT [ BY ] increment ]
  [ MINVALUE minvalue | NO MINVALUE ]
  [ MAXVALUE maxvalue | NO MAXVALUE ]
  [ START [ WITH ] start ]  [ CACHE cache ]
  [ [ NO ] CYCLE ]
  [ OWNED BY { table.column | NONE } ]
```

Para cambiar el valor

```
ALTER SEQUENCE miseq  RESTART WITH 105;
```

Para invocar el proximo valor:

```
select nextval('auditoria_id_seq')
```

Plpgsql Trigger y Stored Procedure
Alter Trigger

```
ALTER TRIGGER name ON table RENAME TO new_name
```

Renombra el trigger

Ejemplo:

```
ALTER TRIGGER trg1 ON table1 RENAME TO trgnew.
```

Plpgsql Trigger y Stored Procedure
Drop Trigger

```
DROP TRIGGER [ IF EXISTS ] name ON
table [ CASCADE | RESTRICT ]
```

- Elimina el trigger cuyo nombre es name

IF EXISTS *no da error si el trigger no existe.*
CASCADE *elimina objetos dependientes.*
RESTRICT *si hay objetos dependientes no elimina el trigger.*

Ej: **DROP TRIGGER** trg_alumnos_gen_id;

Plpgsql Trigger y Stored Procedure

Standard sobre Trigger

```
Los triggers se estandarizaron en Sql-99,
cubriendo las siguientes especificaciones y
restricciones.
```

Eventos: Insert, Delete y Update, en update se puede especificar hasta el atributo.

Granularidad: fila (FOR EACH ROW) o sentencia (FOR EACH STATEMENT).

Referencing:

Por fila: Renombrado de **new** y **old**.

Por Sentencia: Renombrado de **inserting** y **deleting**

Plpgsql Trigger y Stored Procedure

Ejemplos de Trigger

Ver Rar de la catedra → "Bda ejemplos trigers.rar"

```
CREATE TRIGGER trg_poli_clientes AFTER INSERT OR DELETE OR UPDATE
 ON clientes
 FOR EACH ROW EXECUTE PROCEDURE fx_poli_clientes();

 CREATE OR REPLACE FUNCTION fx_poli_clientes() RETURNS trigger AS $$
    BEGIN
        -- Verifica si esta borrando
        IF tg_op = 'DELETE' THEN
           INSERT INTO AUDITORIA (FECHA, INFORMACION )
              VALUES  (current_timestamp, 'Borrando: ' || NEW.CUIT);
        END IF;
        -- Verifica Update e Insert ...
        RETURN null;
    END;$$
 LANGUAGE 'plpgsql';
```

Analice que actividades realiza

Plpgsql Trigger y Stored Procedure
Referencias

Libro: Fundamentos de Bases de Datos
Autor: Silberschatz / Korth / Sudarshan
Editorial: Mc Graw Hill

Libro: Introducción a los SISTEMAS DE BASES DE DATOS
Autor: C.J. Date
Editorial: Addison Wesley

https://msdn.microsoft.com/es-es/library/ms187926.aspx
https://msdn.microsoft.com/es-es/library/ms186755.aspx
https://msdn.microsoft.com/es-es/library/ms189799.aspx
http://www.postgresql.org/docs/9.5/static/plpgsql-control-structures.html

http://www.postgresqltutorial.com/postgresql-stored-procedures/
http://www.sqlines.com/postgresql/stored_procedures_functions
https://www.postgresql.org/docs/current/static/plpgsql.html