

# Taller de Bases de Datos

# IN1078C

Prof: Mariella Gutiérrez Valenzuela

# Contenidos

2. Procedimientos Almacenados y Triggers
  - a. Disparadores (Triggers)
  - b. Funciones en lenguaje Pl/pgSql

# Funciones en lenguaje PL/pgSql

- Es un programa almacenado en la base de datos que permite ejecutar un conjunto de sentencias.
- Puede estar escrito en un lenguaje procedural como C, Perl, Phyton, entre otros.
- También puede estar escrito en un lenguaje procedural propio de postgresql llamado PL/pgSQL.
- Existen tanto funciones como procedimientos en Postgresql, la diferencia está en que un procedimiento puede no tener un retorno y no usa la sentencia return.
- Se puede utilizar en ellos todas las estructuras de control vistas para los triggers.
- Para utilizar el lenguaje PL/pgSQL primero debe crearse:
  - CREATE PROCEDURAL LANGUAGE plpgsql; o bien,
  - CREATE TRUSTED PROCEDURAL LANGUAGE plpgsql; si se quiere que cualquier usuario lo utilice, aun cuando no sea el administrador.

# Funciones en lenguaje Pl/pgSql

PL/pgSQL es un lenguaje estructurado en bloques. Como mínimo se tiene un bloque principal en el procedimiento almacenado y dentro de este se puede tener subbloques que se delimitan con las palabras reservadas BEGIN-END.

Estructura General de un bloque:

```
[ << etiqueta >> ]  
[ DECLARE  
    declaraciones de variables ]  
BEGIN  
    codigo  
END [ etiqueta ];
```

# Funciones en lenguaje Pl/pgSql

La estructura General de una función es:

```
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
| COST execution_cost
| ROWS result_rows
| SET configuration_parameter { TO value | = value | FROM CURRENT };
```

# Funciones en lenguaje Pl/pgSql

Donde:

## Argumentos:

- **argmodo:** El modo de un argumento puede ser IN, OUT, or INOUT. Por defecto se usa IN.
- **argtipo:** Los tipos que podemos utilizar son todos los disponibles en PostgreSQL y todos los definidos por el usuario.

**Retorno:** El retorno de la función puede ser de cualquier tipo de datos e incluso registros. Si no retorna nada útil se puede poner Return void.

**Declaraciones de variables:** Las declaraciones de variables se pueden realizar de la siguiente manera:

nombre\_variable ALIAS FOR \$n; (\$n = orden de declaración del argumento.)

nombre\_variable [CONSTANT] tipo [NOT NULL] [ { DEFAULT | := } expresion ];

## Otros parámetros:

- **IMMUTABLE:** La función no puede alterar a la base de datos y siempre devolverá el mismo resultado, dados los mismos valores como argumentos. Este tipo de funciones no pueden realizar consultas en la base de datos.
- **STABLE:** La función no puede alterar a la base de datos y que siempre devolverá el mismo resultado en una consulta individual de una tabla, dados los mismos valores como argumentos. El resultado podría cambiar entre sentencias SQL.
- **VOLATILE:** La función puede devolver diferentes valores, incluso dentro de una consulta individual de una tabla (valor por defecto)

# Funciones en lenguaje Pl/pgSql

Donde:

- **CALLED ON NULL INPUT:** Indica que la función se ejecutará aunque algunos de los argumentos sean NULL. *Valor por defecto.*
- **RETURNS NULL ON NULL INPUT / STRICT:** Indican que la función no se ejecutará y devolverá el valor NULL si alguno de los argumentos es NULL.
- **SECURITY INVOKER:** Indica que la función se ejecutará con los privilegios del usuario que la ejecuta (valor por defecto)
- **SECURITY DEFINER:** Indica que la función se ejecutará con los privilegios del usuario que la creó.

# Funciones en lenguaje Pl/pgSql

**Ejemplo1:** El siguiente código crea una función que recibe dos parámetros de tipo entero y retorna un entero.

```
CREATE OR REPLACE FUNCTION ejemplo(integer, integer) RETURNS integer AS $$
```

```
DECLARE
```

```
numero1 ALIAS FOR $1; -- le da un nombre al argumento 1
```

```
numero2 ALIAS FOR $2; -- le da un nombre al argumento 2
```

```
constante CONSTANT integer := 100; -- define una constante de tipo entero
```

```
resultado integer; -- define variable de tipo entero que se utilizara como retorno
```

```
BEGIN
```

```
resultado := (numero1 * numero2) + constante;
```

```
RETURN resultado;
```

```
END;
```

```
$$ LANGUAGE plpgsql;
```

Para llamar a esta función se ejecuta:

```
Select ejemplo(2,2);
```

Lo cual retorna el valor 104.

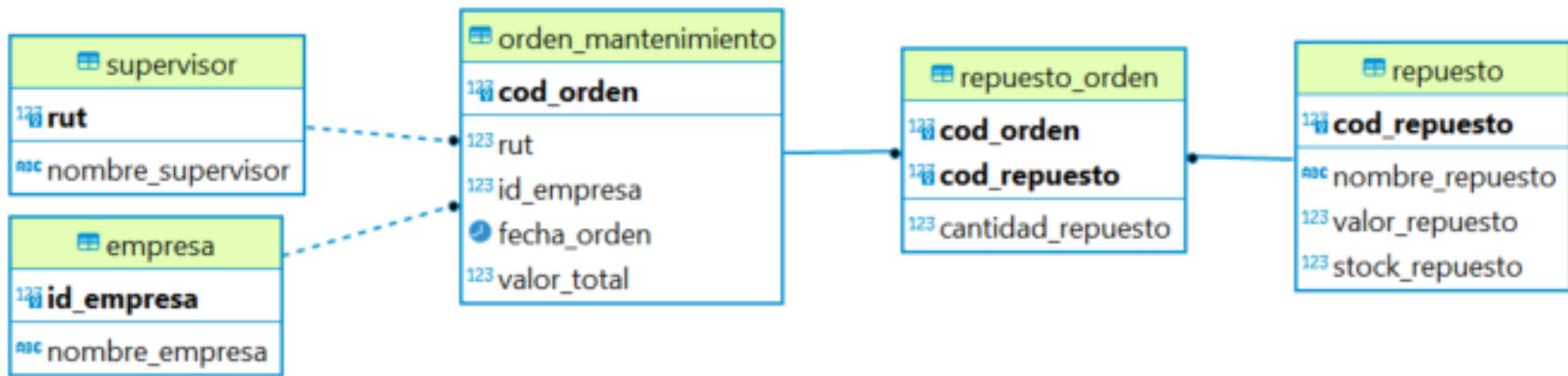
# Funciones en lenguaje Pl/pgSql

**Ejercicio 1:** Escriba un procedimiento almacenado que calcule la potencia de un número.

Es decir, si se le pasa como parámetro (3,3) debe entregar como resultado 27 ( $3^3$ )

# Funciones en lenguaje Pl/pgSql

Consideremos nuevamente la base de datos de ordenes de mantenimiento:



# Funciones en lenguaje Pl/pgSql

**Ejemplo2:** La siguiente función verifica en la tabla supervisor de la base de datos si un supervisor existe o no.

```
CREATE OR REPLACE FUNCTION buscar_supervisor (varchar) -- Se le va a pasar por parámetro
                                                       -- un argumento de tipo varchar
RETURNS bool AS $$                                -- Retorna un booleano (verdadero o falso)
DECLARE
    nombre_buscar alias for $1;      -- le da un nombre al argumento 1
    registro supervisor%ROWTYPE;    -- Define la variable registro de tipo fila de la
                                    -- tabla supervisor a través del símbolo % y luego la
                                    -- palabra reservada ROWTYPE
BEGIN
    SELECT INTO registro * FROM supervisor -- recupera el primer registro donde aparezca
    WHERE nombre_supervisor = nombre_buscar; -- el nombre_buscar o Null si no lo encuentra.
    IF FOUND THEN -- Se utiliza la variable FOUND para verificar si el select
        RETURN true; -- retornó una fila. En ese caso retorna verdadero
    END IF;
    RETURN false; -- Retorna Falso si el select no retornó una fila, es decir, no entró al IF.
END;
$$ LANGUAGE 'plpgsql';
```

# Funciones en lenguaje Pl/pgSql

## IMPORTANTE

Cuando se utiliza un Select para asignar un valor a una variable la sintaxis es la siguiente:

SELECT select\_expressions INTO [STRICT] variable FROM ...; o

SELECT INTO [STRICT] variable select\_expressions FROM ...;

Si se utiliza STRICT el select debe retornar un (1) valor si no reportará un error, el cual podrá ser manejado como una excepción.

Si no se utiliza STRICT el select retornará el primer valor que coincida con el criterio de selección, o NULL si no encuentra ningún registro. El resultado se podrá manejar con FOUND.

Con STRICT	Sin STRICT
<pre>BEGIN   SELECT * INTO STRICT registro FROM profesor   WHERE nombre = nombre_buscar;   RETURN true; EXCEPTION   WHEN NO_DATA_FOUND THEN     RETURN false;   WHEN TOO_MANY_ROWS THEN     RETURN false; END;</pre>	<pre>BEGIN   SELECT * INTO registro FROM profesor   WHERE nombre = nombre_buscar;   IF FOUND THEN     RETURN true;   END IF;   RETURN false; END;</pre>

# Funciones en lenguaje Pl/pgSql

**Ejercicios:** Dada la base de datos de órdenes de mantenimiento

- a) Escriba una función que dado el nombre de un supervisor retorne la cantidad de órdenes de mantenimiento que ha supervisado, o cero si no ha supervisado ninguna.
  
- b) Escriba una función que, dada nombre de repuesto retorne en un texto en cuantas órdenes de mantenimiento se ha utilizado y cuál es la cantidad total utilizada.  
Ej. ‘El repuesto ‘||nombre repuesto||’ se ha utilizado en ‘||XX||’ órdenes de mantenimiento y la cantidad total usada es’||YY;
  
- c) Escriba una función que dado el nombre de una empresa entregue verdadero si la empresa tiene órdenes de mantenimiento (al menos una) y falso si no tiene.
  
- d) Escriba una función que, dado un nombre de supervisor y un nombre de empresa, retorne en un texto si la empresa tiene una o más órdenes de mantenimiento supervisadas por el supervisor con la cantidad de órdenes y el valor total. Si no, debe decir en un texto que la empresa no tiene órdenes de mantenimiento supervisadas por el supervisor.