Bases de Datos

10. Triggers, Procedimietos y Funciones

IES La Encantá

Índice

- Triggers, Procedimientos y Funciones
- Bloque
- Procedimientos
  - Parámetros
  - Variables
- Estructuras de control
- Funciones

Triggers, Procedimientos y Funciones

Son objetos que contienen código SQL y se almacenan asociados a una base de datos.

Procedimiento: Es un objeto que se crea con la sentencia CREATE PROCEDURE y se invoca con la sentencia CALL. Un procedimiento puede tener cero o muchos parámetros de entrada y cero o muchos parámetros de salida.

Función: Es un objeto que se crea con la sentencia CREATE FUNCTION y se invoca con la sentencia SELECT o dentro de una expresión. Una función puede tener cero o muchos parámetros de entrada y siempre devuelve un valor, asociado al nombre de la función.

Trigger: Es un objeto que se crea con la sentencia CREATE TRIGGER y tiene que estar asociado a una tabla. Un *trigger* se activa cuando ocurre un evento de inserción, actualización o borrado, sobre la tabla a la que está asociado.

Usos

- Facilita las tareas administrativas (copias de seguridad, control de usuarios, ….)
- Validación y verificación de usuarios
- Consultas muy avanzadas
- Centralizar operaciones de negocio/datos.
- Seguridad: por ejemplo, sólo los procedimientos son los que modifican los datos

Estructura de un bloque


```
[ DECLARE
-- declaraciones ]
BEGIN
-- sentencias ejecutables
[ EXCEPTION
-- tratamiento de excepciones ]
END;
```

Procedimiento / Función

- A los bloques se les puede poner nombre.
  - Si no devuelve datos → **Procedimiento**

    ```
    PROCEDURE nombre
    bloque
    ```

  - Si devuelve datos → **Función**

    ```
    FUNCTION nombre RETURNS tipoDato
    IS bloque
    ```

- Si no llevan nombre son bloques anónimos

CREATE PROCEDURE

- https://mariadb.com/kb/en/create-procedure/
  - `CREATE [OR REPLACE] PROCEDURE nombre (parámetros)`

    `bloque`

  - parámetros:
    - `[ IN | OUT | INOUT ] nombreParametro tipo`

- Los tipos de los parámetros y las variables declaradas en los bloques son los mismos empleados en el DDL:
  - https://mariadb.com/kb/en/data-types/

Ejemplo procedimiento

```
DELIMITER //

CREATE PROCEDURE cantidadEmpleados()
BEGIN
  SELECT count(*) FROM empleado;
END

//

DELIMITER ;
```

- Para que el cliente de MariaDB no interprete los punto y coma como separador de instrucciones, antes y después de la declaración, hay que indicar el delimitador mediante `DELIMITER`
  - https://mariadb.com/kb/en/delimiters/

CALL

- Una vez creado un procedimiento, hay que invocarlo mediante **CALL**
  - `CALL nombreProcedimiento(parámetros)`

- `CALL cantidadEmpleados();`
- `CALL cantidadEmpleados;`
  - Como no tiene parámetros, podemos omitir los paréntesis

Consultando los procedimientos

- `SHOW PROCEDURE STATUS`
  - https://mariadb.com/kb/en/show-procedure-status/
  - Muestra todos los procedimientos del sistema.
    - nombre, creador, fecha de creación, ...

- Si queremos restringir a una bd:
  - `show procedure status where db='jardineria';`

Tipos de parámetros I

- Parámetros de entrada (IN)
  - Si no lo indicamos, por defecto son de entrada

    ```
    CREATE PROCEDURE entrada(p1 INT) …
    CALL entrada(5);
    ```

- Parámetros de salida (OUT)

  ```
  CREATE PROCEDURE salida(IN cantidad INT,
  OUT total INT)
  ```

  Para pasar un parámetro de salida, se antepone una @ a su nombre

  ```
  CALL salida(7, @resultado);
  ```

Tipos de parámetros II

- Parámetros de entrada/salida (IN/OUT)
  - Se lee como dato de entrada, y desde el procedimiento se le asigna un resultado como salida

```
CREATE PROCEDURE contar(INOUT cuenta
INT(4),IN incremento INT(4))
CALL contar(@cantidad, 5);
```

Declaración y uso de variables

- Las variables a utilizar se declaran tras abrir un bloque:

```
DECLARE variable tipo;
```

- Para asignar un valor a una variable o parámetros:

```
SET variable = valor;
```

Ejemplos con parámetros I


```
DELIMITER //
CREATE PROCEDURE saluda (nombre VARCHAR(128))
BEGIN
   SELECT concat("Hola ", nombre);
END;
//
DELIMITER ;

call saluda("Aitor");
```

```
DELIMITER //
CREATE PROCEDURE saludaOut (nombre VARCHAR(128),
                            OUT saludo VARCHAR(128))
BEGIN
   SET saludo = concat("Hola ", nombre);
END;
//

DELIMITER ;
```

Ejemplos con parámetros II


```
-- Ejemplo con parámetro de entrada y de salida
DELIMITER //
CREATE PROCEDURE saludaInOut (INOUT nombre VARCHAR(128),
                              apellido VARCHAR(128),
                              OUT saludo VARCHAR(128))
BEGIN
  declare nombreCompleto varchar(256);
  set nombreCompleto = concat(nombre, " ", apellido);
  set nombre = nombreCompleto;
  set saludo = concat("Hola ", nombreCompleto);
END;
//

DELIMITER ;

set @nombre = "Aitor";
call saludaInOut(@nombre, "Medrano", @saludo);
select @nombre, @saludo;
```

SELECT INTO

- https://mariadb.com/kb/en/selectinto/
- Permite almacenar el resultado de una consulta en [una] variable/s.
  - La consulta sólo debe devolver un registro.

```
select max(total), min(total)
  into @mayor, @menor
  from pago;
```

Estructuras de control

- Condicionales
  - IF-THEN-ELSE
  - CASE
- Instrucciones repetitivas / bucles
  - LOOP
  - REPEAT
  - WHILE

IF-THEN


```
IF condicion
  THEN sentencia
    [ELSEIF condicion THEN sentencia] ...
    [ELSE sentencia]
END IF
```

- https://mariadb.com/kb/en/if/

```
delimiter //
create procedure categoriaEdad (edad integer)
begin
        declare resultado varchar(128);
        if edad < 18 then
                set resultado = "junior";
        elseif edad < 45 then
                set resultado = "senior";
        else
                set resultado = "veterano";
        end if;
        select resultado;
end;
//
delimiter ;
call categoriaEdad(33);
```

CASE

- https://mariadb.com/kb/en/case-statement/

```
CASE condicionQueTomaValor
  WHEN valor THEN sentencia
  [WHEN valor THEN
sentencia] ...
  [ELSE sentencia]
END CASE
```

Se comparan valores

```
CASE
  WHEN condicionEvalua THEN
sentencia
  [WHEN condicionEvalua THEN
sentencia] ...
  [ELSE sentencia]
END CASE
```

Se comparan expresiones

Ejemplo CASE


```
DELIMITER //
CREATE PROCEDURE categoriaEdadCaseValor (edad
integer)
BEGIN
      declare resultado varchar(128);
      case edad
            when 17 then
                  set resultado = "junior";
            when 18 then
                  set resultado = "junior";
            when 19 then
                  set resultado = "senior";
            else
                  set resultado = "desconocido";
      end case;
      SELECT resultado;
END; //


DELIMITER ;
call categoriaEdadCaseValor(33);
```

```
DELIMITER //
CREATE PROCEDURE categoriaEdadCase (edad
integer)
BEGIN
      declare resultado varchar(128);
      case
            when edad < 18 then
                  set resultado = "junior";
            when edad < 45 then
                  set resultado = "senior";
            else
                  set resultado = "veterano";
      end case;
      SELECT resultado;
END; //


DELIMITER ;
call categoriaEdadCase(33);
```

WHILE

- https://mariadb.com/kb/en/while/

```
WHILE condicion DO
     sentencias
END WHILE
```

Ejemplo WHILE


```
DELIMITER //
DROP PROCEDURE IF EXISTS bucleWhile //
CREATE PROCEDURE bucleWhile(IN tope INT, OUT suma INT)
BEGIN
	DECLARE contador INT;
	SET contador = 1;
	SET suma = 0;
	WHILE contador <= tope DO
		SET suma = suma + contador;
		SET contador = contador + 1;
	END WHILE;
END
//
DELIMITER ;
CALL bucleWhile(10, @resultado);
SELECT @resultado;
```

REPEAT

- https://mariadb.com/kb/en/repeat-loop/
- Similar a un `do-while` de *Java*

```
REPEAT
     sentencias
UNTIL condicion
END REPEAT
```

Ejemplo REPEAT


```
DELIMITER //
DROP PROCEDURE IF EXISTS ejemploRepeat //
CREATE PROCEDURE ejemploRepeat(IN tope INT, OUT suma INT)
BEGIN
    DECLARE contador INT;
    SET contador = 1;
    SET suma = 0;
    REPEAT
        SET suma = suma + contador;
        SET contador = contador + 1;
    UNTIL contador > tope
    END REPEAT;
END
//
DELIMITER ;
CALL ejemploRepeat(10, @resultado);
SELECT @resultado;
```

LOOP + LEAVE

- https://mariadb.com/kb/en/loop/
- https://mariadb.com/kb/en/leave/
- LOOP crea un bucle infinito, el cual hay que romper con LEAVE

```
etiqueta: LOOP
  sentencias;
  IF condicion THEN
      LEAVE etiqueta;
  END IF;
END LOOP
```

Ejemplo LOOP


```
DELIMITER //
DROP PROCEDURE IF EXISTS ejemploLoop//
CREATE PROCEDURE ejemploLoop(IN tope INT, OUT suma INT)
BEGIN
    DECLARE contador INT;
    SET contador = 1;
    SET suma = 0;
    bucle: LOOP
        IF contador > tope THEN
            LEAVE bucle;
        END IF;

        SET suma = suma + contador;
        SET contador = contador + 1;
    END LOOP;
END
//
DELIMITER ;
CALL ejemploLoop(10, @resultado);
SELECT @resultado;
```

Funciones

- Similar a un procedimiento pero devuelven un valor.
- Se pueden usar en las consultas
- Normalmente se utilizan para realizar cálculos
- Comandos:
  - `CREATE FUNCTION`
  - `SHOW CREATE FUNCTION`
  - `SHOW FUNCTION STATUS`
  - `DROP FUNCTION`

CREATE FUNCTION

- https://mariadb.com/kb/en/create-function/
  - `CREATE FUNCTION nombre (parámetros)`
    `RETURNS tipo`
    `bloque con RETURN`
  - parámetros:
    - `[ IN | OUT | INOUT ] nombreParametro tipo`
- El bloque de instrucciones debe contener alguna instrucción RETURN que devuelva el `tipo` esperado.

Ejemplo función


```
delimiter //
create or replace function precioConIVA(precio decimal(10,2))
       returns decimal(10,2)
begin
    declare piva decimal(10,2);
    set piva = precio * 1.21;
    return piva;
end;
//

delimiter ;

select total, precioConIVA(total) from pago;

select precioConIva(1234);
```

Referencias

- Apuntes José Juan Sánchez Hernández - IES Celia Viñas (Almería): https://josejuansanchez.org/bd/unidad-12-teoria/index.html

- Documentación MaríaDB: https://mariadb.com/kb/en/programming-customizing-mariadb/

¿Alguna pregunta?