RUTINAS: PROCEDIMIENTOS Y FUNCIONES ALMACENADAS EN MySQL

USO DE CURSORES

CREACIÓN DE TRIGGERS O DISPARADORES

Un programa almacenado es un conjunto de instrucciones almacenadas dentro del servidor de Bases de Datos y que se ejecutan en él. El programa almacenado se identifica por un nombre y es un objeto más de la Base de Datos.

Tipos de programas almacenados:

-**Procedimientos almacenados**, realizan el proceso diseñado en ellos,se ejecutan cuando son llamados, pueden aceptar varios argumentos de entrada y devolver varios datos salida.

-Funciones almacenadas, retornan o devuelven un valor como parámetro de salida. **Las funciones pueden ser utilizadas dentro de instrucciones sql** .

-**Triggers, desencadenadores o disparadores**, son rutinas o programas que se activan se “disparan” ante un determinado suceso/evento ocurrido en la base de datos.

Las rutinas (procedimientos, funciones y triggers) son un conjunto de sentencias/comandos SQL , que pueden guardarse en el servidor, de modo que, una vez guardados, el cliente no necesita lanzar cada comando o sentencia individualmente, sino que podremos lanzar el procedimiento almacenado como un único comando.

PROCEDIMIENTOS:

Un procedimiento **es un programa** que se almacena físicamente en una tabla dentro del sistema de bases de datos. Este programa esta hecho con un lenguaje, propio de cada Gestor de BD, y está compilado, por lo que la velocidad de ejecución será muy rápida.

SINTAXIS creación procedimiento:

CREATE PROCEDURE nombre_procedimiento ([parameter[,...]])
[characteristic ...]
cuerpo_del_procedimiento

parameter:

[IN | OUT | INOUT] nombre_parametro **type**

IN es el tipo de parámetro por defecto, el parámetro declarado de tipo IN es sólo de entrada, permite introducir datos de entrada en el proceso que realiza el procedimiento.

OUT es el tipo de parámetro de salida, un parámetro así declarado sirve para **"retornar"** valores o datos de salida, resultantes del proceso ejecutado por el procedimiento.

```
DELIMITER $$
DROP PROCEDURE IF EXISTS valorinicial;
CREATE PROCEDURE valorinicial (OUT dato int)
BEGIN
       SET dato=123;
      /*  otra posibilidad SELECT COUNT(*) INTO dato FROM
EMPLEADOS;*/
END $$
DELIMITER;
```

INOUT, el parámetro es declarado de tipo Entrada-Salida, de modo que, nos permite pasar valores de entrada al procedimiento y devolver datos o valores de salida obtenidos en la ejecución del procedimiento.

```
DELIMITER $$
DROP PROCEDURE IF EXISTS triplicador;
CREATE PROCEDURE triplicdor (INOUT dato int)
BEGIN
      SET dato=dato*3;
END $$
DELIMITER;
```

type:
Cualquier tipo de dato valido de MySQL

characteristic:
[LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | **NO SQL** | **READS SQL DATA** | **MODIFIES SQL DATA** }
| SQL SECURITY {DEFINER|INVOKER}

]
cuerpo_del_procedimiento:

Cuerpo de la rutina, formado generalmente por sentecias SQL, en caso de existir más de una debe ir en un bloque delimitado por sentencias **BEGIN..... END**

Para lanzar o ejecutar un procedimiento:

CALL nombre_procedimiento([datos,...]);

// lista de datos o argumentos en la llamada

SINTAXIS borrado procedimiento:

DROP PROCEDURE IF EXISTS nombre_procedimiento;

SINTAXIS para ver información de nuestros procedimientos almacenados:

SHOW CREATE PROCEDURE nombre_procedimiento;

FUNCIONES:

Una función almacenada es un programa que **devuelve un valor.**

Las funciones, a diferencia de los procedimientos, **se pueden utilizar en expresiones, se pueden incluir en el cuerpo de otras funciones o de procedimientos , se pueden incluir en el interior de sentencias SQL como SELECT, UPDATE, INSERT, DELETE.**

Además de las funciones que nos proporciona el SGBD, vamos a crear nuestras propias funciones para hacer tareas más especializadas…

Vamos a ver cómo crear funciones en MySQL:

SINTAXIS creación función:

CREATE FUNCTION nombre_función ([parameter[,...]])
RETURNS type
[characteristic ...]
cuerpo_de_la_función

parameter: nombre_parametro type

En MySql **todos los parámetros de una función son de tipo IN, no pueden ser de otro tipo, el nombre del parámetro no va precedido por IN**

type:
Cualquier tipo de dato valida de MySQL

characteristic:
[LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | **NO SQL** | **READS SQL DATA** | **MODIFIES SQL DATA** }
| SQL SECURITY {DEFINER|INVOKER}]

cuerpo_de_la_función:
Cuerpo de la rutina, formado generalmente por sentencias SQL, en caso de existir más de una debe ir en un bloque delimitado por sentencias BEGIN..... END

EJEMPLO:

```
DELIMITER //

CREATE FUNCTION holaMundo( ) RETURNS VARCHAR(20)
BEGIN
   RETURN 'HolaMundo';
END //
```

Para comprobar que funciona tecleamos lo siguiente en la consola de MySQL :

```
SELECT holaMundo();
```

SINTAXIS borrado de función:
Para borrar la función que acabamos de crear :

```
DROP FUNCTION IF EXISTS holaMundo;
```

Para ver información de nuestras funciones almacenadas:

```
SHOW CREATE FUNCTION nombre_función;
```

Uso de las VARIABLES en procedimientos y funciones:

Las variables en las funciones se usan de igual manera que en los procedimientos almacenados, se declaran con la sentencia **DECLARE**, y se asignan valores con la sentencia **SET**.

Las variables se declaran al comienzo del proceso, antes de la declaración de cursores y de manejadores de errores (handlers)

La clausula: **DECLARE nombre_var type [DEFAULT VALOR];**

Si no se incluye DEFAULT la/s variable/s tomarán valor null.

type cualquiera de los tipos de datos utilizados en la creación de tablas.

ejemplos:

```
DECLARE a ,b INT DEFAULT 5;

DECLARE a INT;

DECLARE cadena1, cadena2 VARCHAR(30) DEFAULT 'VALOR INICIAL';
```

Sentencia SET

Asignar valores a una o varias variables

set var1=expresión1 [,var2=expresión2,..... varn=expresiónn];

```
DELIMITER //
CREATE FUNCTION holaMundo() RETURNS VARCHAR(30)
BEGIN
  DECLARE salida VARCHAR(30) DEFAULT 'Hola mundo con VARIABLES';

  SET salida = 'Hola mundo con VARIABLES';
  RETURN salida;
END//
```

Esta variable (salida) es de ámbito local, y será destruida una vez finalice la función. Dejará de ser accesible desde fuera del cuerpo de la función.
Cabe destacar el uso de la sentencia DEFAULT en conjunto con DECLARE, que asigna un valor inicial al declarar la variable.

Uso de parámetros en funciones:

```
DELIMITER //
DROP FUNCTION IF EXISTS holaMundo;
CREATE FUNCTION holaMundo(entrada VARCHAR(20)) RETURNS VARCHAR(20)
BEGIN
  DECLARE salida VARCHAR(20);
/*asignamos a la variable salida el valor recibido en la llamada a la función
  SET salida = entrada;
  RETURN salida;
END//
 DELIMITER;
```

Ahora hemos creado una función que devuelve el mismo valor que le pasamos como argumento.

INSTRUCCIONES CONDICIONALES:

A) IF-THEN-ELSE

a) IF expresión THEN
sentencia; … sentencias;
END IF;

b) IF expresión THEN
sentencia; … sentencias;
ELSE
sentencia; … sentencias;
END IF;

c) IF expresión1 THEN
sentencia; … sentencias;
ELSEIF expresión2 THEN
sentencia; … sentencias;
ELSE
sentencia; … sentencias;
END IF;

B) CASE

```
CASE    expresión
      WHEN value1 THEN  sentencias …..
      [WHEN value2 THEN  sentencias ..…]
       …
      [ELSE  sentencias …]
END CASE;
```

INSTRUCCIONES REPETITIVAS O LOOPS (SINTAXIS)

A) SIMPLE LOOP

```
[etiqueta:] LOOP
        instrucciones……..
        if condiciónsalida  THEN LEAVE [etiqueta];
        end if;
        ……. instrucciones
    END LOOP [etiqueta];
```

B) REPEAT UNTIL

```
[etiqueta:] REPEAT
        instrucciones;
        …….....
    UNTIL expresión
    END REPEAT [etiqueta];
```

el conjunto de instrucciones (cuerpo del ciclo) se ejecuta al menos 1 vez, ***1 o varias veces, hasta expresión cierta.***

C) WHILE

```
[etiqueta:] WHILE expresión DO
        instrucciones
        …….....
    END WHILE [etiqueta];
```

el conjunto de instrucciones (cuerpo del ciclo) se ejecuta 0 o varias veces, mientras expresión cierta.

Para ver uso de estructuras de control repetitivas veamos el siguiente ejemplo:

Vamos a crear una función que acepte un dividendo y un divisor y haga una división sin usar el operador división:

```
DELIMITER //
CREATE FUNCTION divide(dividendo int,divisor int) returns int
begin
   declare aux int;
   declare contador int;
   declare resto int;
   SET contador = 0;
   SET aux = 0;
   WHILE (aux + divisor) <= dividendo DO
      SET aux = aux + divisor ;
      SET contador = contador + 1;
   END WHILE;
   SET resto = dividendo - aux ;
RETURN contador;
end//
DELIMITER ;
```

Para usarlo, simplemente llamaríamos a la función así:

```
SELECT divide(20,2);
```

CURSORES:

Cuando desde dentro de un procedimiento o función queremos **recuperar más de una fila(tupla) no nos sirve únicamente la sentencia**

select con su propiedad into variable

En ese caso necesitamos el uso de cursores.

Un cursor se asocia a un conjunto de filas (tuplas) o a una consulta sobre una/unas tablas de una base de datos.

Debemos distinguir estos momentos en el código fuente:

A) CREACIÓN DE UN CURSOR :

```
DECLARE nombre_cursor CURSOR FOR
        sentencia SELECT;
```

Esta declaración debe hacerse dentro del cuerpo de la rutina después de las declaraciones de las variables necesarias en el proceso.Antes de las sentencias de proceso o ejecutables.

LISTA DE COMANDOS RELACIONADOS CON CURSORES :

OPEN inicializa el conjunto de resultados asociados con el cursor.

OPEN nombre_cursor;

Con esta orden se ejecuta el cursor, se obtiene el resultado de la consulta asociada al cursor y se almacena en memoria, disponemos de un buffer en memoria con la tabla resultante de la ejecución del cursor.

FETCH extrae la siguiente fila de valores del conjunto de filas del resultado del cursor, avanzando su puntero interno una posición.

FETCH nombre_cursor INTO lista de variables;

CLOSE cierra el cursor liberando la memoria que ocupa y haciendo imposible el acceso a los datos.

CLOSE nombre_cursor;

DISPARADORES O TRIGGERS

Un trigger o disparador es una rutina almacenada que se activa o se ejecuta cuando en una tabla ocurre un **evento** de tipo INSERT, DELETE O UPDATE.
Es decir, un disparador está ***asociado a una tabla concreta y a un evento concreto*** y se "dispara", se ejecuta, cuando sucede el evento en la tabla.

Gestionamos disparadores con las siguientes sentencias:

CREATE TRIGGER, SHOW TRIGGER, DROP TRIGGER

Para crear un disparador:

CREATE TRIGGER nombre_disparador momento_disparador evento_disparador ON tabla
FOR EACH ROW sentencia_disparador

momento_disparador:
Es el momento en que el disparador se ejecuta. Los valores posibles para momento son **BEFORE, AFTER**, el disparador se ejecutará antes o después de la sentencia que lo activa. (INSSERT/DELETE/UPDATE en la tabla)

evento_disparador:
Indica la sentencia (INSERT, UPDATE, DELETE) que está asociada al disparador y por la tanto es la sentencia que lo activa.

No puede haber dos disparadores para una misma tabla que se correspondan al mismo momento y sentencia.

FOR EACH ROW:
Acciones a ejecutar sobre cada fila de la tabla.

sentencia_disparador:
Se ejecuta al saltar el disparador, si es una sentencia múltiple o compuesta las sentencias deben ir encerradas entre BEGIN.... END

Las columnas de la tabla asociada al disparador se pueden referenciar con los alias:
El disparador dispone al ejecutarse o lanzarse de las dos referencias siguientes
OLD y NEW.
OLD.nombre_columna valor de columna en una fila antes de ser actualizada
NEW.nombre_columna valor de columna en una fila después de ser actualizada

Claro está, que en un disparador para evento INSERT, sólo se dispone de la referencia NEW, la referencia OLD está con valor null, porque no hay valores de fila anteriores.

En un disparador para evento DELETE, sólo se dispone de la referencia OLD, porque no hay nuevos valores de fila, la referncia NEW está con valor null.

Por último, en un disparador para un evento UPDATE, usaremos tanto las referencias OLD como NEW para referirnos a los valores de columna antes y después de la modificación.

Para eliminar un disparador:

DROP TRIGGER [IF EXISTS] nombreBBDD.nombre_disparador;

Para obtener información de un disparador:

SHOW TRIGGERS [{FROM | IN }] nombre_DB