Funciones almacenadas en MySQL

Funciones almacenadas en MySQL

MySQL tiene muchas funciones predefinidas para realizar diferentes tareas : funciones de tratamiento de cadenas, funciones de tratamientos de fechas, funciones matemáticas, etc. Muchas de ellas ya las hemos utilizado en temas anteriores, pero puede que necesitemos realizar una operación para la cual no exista ninguna función predefinida.

MySQL cuenta con una extensión procedimental de SQL que nos permite definir las funciones que necesitemos y guardarlas en una base de datos para su uso posterior.

El concepto de función es el mismo que en cualquier lenguaje de programación : un bloque de instrucciones al que le asignamos un nombre y cuyo objetivo es obtener un valor de salida a partir de unos valores de entrada.

Las funciones almacenadas se utilizan como cualquier otra función predefinida de MySQL.

Una función almacenada de MySQL se define de esta manera general :

```
create function nombre_funcion ( [parametros] ) returns tipo_retorno
  cuerpo_de_la_función
```

En una misma base de datos no puede haber dos funciones que se llamen igual, pero sí que puede haber dos funciones que se llamen igual en diferentes bases de datos.

Cuerpo de la función

Si la función es sencilla el cuerpo puede constar únicamente de la instrucción **return**.

Ejemplo 1.

Función que calcula el número de días hasta final del mes actual.

```
create function dias_fin_mes () returns int
  return datediff( last_day(now()) , now() );
```

Si el cuerpo de una función consta de más de una instrucción es necesario definir un bloque de instrucciones **begin** ... **end.** Necesitamos que MySQL interprete todo lo escrito entre el create function inicial hasta el end final como un solo comando. Para ello es necesario cambiar el delimitador por otro que no sea el punto y coma habitual. Se suele utilizar $$. Después de esto es necesario volver a restablecer el delimitador antiguo.

Ejemplo 2.

Función que devuelve la cadena proporcionada sin acentos.

```
delimiter $$
create function quita_acentos ( _cadena varchar(100) ) returns varchar(100)
begin
   set _cadena = replace(_cadena,'á','a');
   set _cadena = replace(_cadena,'é','e');
   set _cadena = replace(_cadena,'í','i');
   set _cadena = replace(_cadena,'ó','o');
   set _cadena = replace(_cadena,'ú','u');
   return _cadena;
end$$
delimiter ;
```

Parámetros

Una función puede tener 1 o más parámetros que se escriben en forma de lista separada por comas y encerrada entre paréntesis. Los paréntesis son obligatorios aunque no haya parámetros.

De cada parámetro hay que indicar el nombre y el tipo de datos SQL.

Es recomendable seguir algún tipo de convenio al nombrar los parámetros para así distinguirlos de los nombres de tablas y columnas que también pueden aparecer en el código de las funciones.

El convenio que adoptaremos consistirá en poner un guión bajo como prefijo al nombrar variables locales y parámetros.

Ejemplo 3.

Función con u parámetro que calcula si una letra es una vocal.

```
create function es_vocal ( _letra char(1) ) returns boolean
  return _letra in ('a','e','i','o','u');
```

Ejemplo 4.

Función con dos parámetros que calcula el área de un triángulo.

```
create function area_triangulo ( _base double, _alt double ) returns double
  return _base * _alt / 2;
```

Variables locales

Las funciones almacenadas pueden utilizar variables locales. Todas la variables que se vayan a utilizar en la función se deben declarar al principio del cuerpo de la función, antes que cualquier otra instrucción

Cualquier cambio en el valor de una variable local o en un parámetro no afecta a ninguna variable fuera de la función.

En la declaración se debe indicar el nombre y el tipo SQL de la variable.

La declaración puede incluir un valor inicial para la variable mediante la palabra **default**. Si no se hace así la variable tendrá un valor NULL hasta que se le asigne uno. Recuerda que con los valores nulos no se pueden realizar operaciones como sumar, concatenar, etc.

```
declare _nombre varchar(50);
declare _suma   int default 0;
declare _tipo   char(1) default 'A';
```

Ejemplo 5.

Función que devuelve el nombre de usuario a partir de una dirección de correo electrónico.

```
delimiter $$
create function usuario_email ( _email varchar(100) ) returns varchar(100)
begin
   declare _pos int;
   set _pos = instr(_email, '@');
   return left(_email,_pos-1);
end$$
delimiter ;
```

Eliminar una función

Para eliminar una función se utiliza drop function :

```
drop function nombre_funcion;
drop function if exists nombre_funcion;  -- evita el error si no existe
```

Modificar una función

Para modificar una función es necesario eliminarla y volverla a crear.

MariaDB permite utilizar CREATE OR REPLACE sin tener que eliminar la función previamente, pero MySQL no tiene esa característica.

Uso de funciones

Imagínate que creamos una función llamada cuenta_comas () para obtener el número de comas que aparece en una cadena.

Si lo único que queremos es pasarle unos valores y ver el resultado haremos:

```
select cuenta_comas('es,cat,en');  -- devolverá 2
```

Pero su uso principal será incluir la función en otros comandos SQL como estos:

```
select nombre, idiomas, cuenta_comas(idiomas) from empleados;
select nombre,idiomas from empleados where cuenta_comas(idiomas)>2;
update empleados set num_idiomas = cuenta_comas(idiomas) + 1;
```

También se puede llamar a una función desde otra función. Por ejemplo una función podría contener esta asignación :

```
set _num = cuenta_comas( _idiomas );
set _num = select cuenta_comas(_idiomas); -- ESTO SERIA INCORRECTO
```

Tipos de instrucciones

Asignaciones

Se realizan con la instrucción SET.

```
set <variable> = <expresión>
```

La expresión puede contener valores literales, operadores, funciones o consultas. También se pueden asignar valores a variables mediante **SELECT...INTO...** Esto es muy útil para asignar valores a varias variables mediante una única consulta.

Ejemplos

```
set _numero = 0;
set _pos = instr(_cadena,'-') + 1;
set _nombre = (select nombre from alumno where alumno_id = 1);

select dni, nombre from alumno where alumno_id = 1 into _dni, _nombre;
```

Instrucciones condicionales.

Instrucción IF

Permite evaluar una condición y ejecutar un bloque de instrucciones u otro dependiendo de si la condición se cumple o no.

Ejemplo 1.

```
if _total >= 1000 then
   set _total = _total * 0.95;
end if;
```

Ejemplo 2.

```
if _email like '_%@_%._%' then
   set _valido = true;
else
   set _valido = false;
end if;
```

Ejemplo 3.

```
if _unidades > 300 then
    set _descuento = 0.15;
elseif _unidades > 200 then
    set _descuento = 0.10;
elseif _unidades > 100 then
    set _descuento = 0.05;
else
    set _descuento = 0;
end if;
```

Instrucción CASE

Esta instrucción permite tomar decisiones en base a condiciones múltiples.

Puede utilizarse de dos maneras.

Opción 1.

Puede usarse para evaluar diferentes expresiones una detrás de otra y al llegar a la primera que se cumpla ejecutar las instrucciones indicadas. Si no se cumple ninguna puede usarse ELSE para indicar las instrucciones que deben ejecutarse en ese caso.

Es equivalente a IF ... ELSEIF ... . ELSEIF ... ELSE ... END IF.

```
case
    when _unidades > 300 then
        set _descuento = 0.15;
    when _unidades > 200 then
        set _descuento = 0.10;
    when _unidades > 100 then
        set _descuento = 0.05;
    else
        set _descuento = 0;
end case;
```

Opción 2.

Puede usarse como un operador que permite elegir un valor que luego es utilizado en una expresión.

En el siguiente ejemplo CASE permite elegir un valor que es asignado a una variable.

```
set _descuento = case _categoria
    when 'A' then 0.15
    when 'B' then 0.10
    when 'C' then 0.05
    else 0
end;
```

Instrucciones de repetición.

MySQL dispone de 3 instrucciones para realizar bucles : **WHILE**, **REPEAT** y **LOOP.**

Instrucción WHILE

Funciona igual que la instrucción WHILE de otros lenguajes de programación.

```
set _fact = 1;
set _i = 1;
while _i <= _n do
    set _fact = _fact * _i;
    set _i = _i + 1;
end while;
```

Instrucción REPEAT

Funciona igual que la instrucción DO ... WHILE de otros lenguajes de programación.

```
set _n = 5;
set _fact = 1;
repeat
    set _fact = _fact * _n;
    set _n = _n - 1;
until _n = 0 end repeat;
```

Instrucción LOOP

La condición para salir del bucle puede comprobarse en cualquier parte y salir de él mediante la instrucción LEAVE. Es necesario asignar una etiqueta al bucle. En el siguiente ejemplo la etiqueta elegida ha sido 'bucle'.

```
bucle:loop
    set _fact = _fact * _n;
    if _n=1 then
        leave bucle;
    end if;
    set _n = _n - 1;
end loop bucle;
```

Generación de errores con SIGNAL

Al realizar cualquier operación con SQL podemos obtener mensajes de error por diversos motivos. Cuando se produce un error obtenemos una notificación con un código y un mensaje.

Si estamos utilizando MySQL Workbench esta notificación aparece en la ventana de mensajes de la parte inferior. Si estamos utilizando el cliente de comandos el mensaje se muestra en el terminal.

El uso de una función también puede producir un error.

```
select pow(10,1000000);   --  Error Code: 1690. DOUBLE value is out of  range
```

Al definir una función podemos utilizar la instrucción **SIGNAL** para hacer que la función finalice notificando un error al usuario que ejecutó la función.

Esto es muy útil, entre otras cosas, para validar los valores de los parámetros de una función y si no son correctos mostrar un mensaje indicando el motivo.

Los errores predefinidos en MySQL ya tienen un código asignado. Para los errores definidos por nosotros utilizaremos siempre el código SQLSTATE genérico '45000' (tal como indica el manual de MySQL)

Ejemplo

```
create function aleatorio ( _min int, _max int ) returns int
begin
  if _max < _min then
    signal sqlstate '45000'
    set message_text='El segundo parámetro no puede ser menor';
  end if;
  return floor( _min + rand()*( _max - _min + 1 ));
end$$
```

Si se ejecuta esta función con valores incorrectos se produce un error.

Funciones con consultas

Las funciones almacenadas pueden contener instrucciones SELECT para consultar las tablas de la base de datos siempre y cuando esté incluida en otra instrucción, por ejemplo una asignación, una instrucción IF, una instrucción RETURN, etc...

También se pueden utilizar consultas SELECT...INTO para asignar valores a variables.

Ejemplos de consultas permitidas

Consulta que obtiene un único valor que se asigna a una variable

```
set _id = (select id from usuario where dni='12345678Z');
```

Consulta que devuelve un único valor utilizado como parte de una expresión

```
if (select count(*) from usuario)> 5 then …
```

Consulta que devuelve un único valor que se utiliza como valor de retorno

```
return (select max(id) from usuario);
```

Consulta con SELECT … INTO que asigna valor a dos variable diferente.

```
select id,nombre from usuario where dni='12345678Z' into _id, _dni;
```

NOTA :

Una función también puede contener instrucciones INSERT, DELETE o UPDATE pero nos abstendremos de utilizar esta posibilidad y utilizaremos únicamente funciones que se limiten a consultar datos y devuelvan un resultado sin modificar el contenido de la base de datos.

Cuando necesitemos insertar, eliminar o modificar utilizaremos un procedimiento almacenado en lugar de una función.

Ejemplo

En el ejemplo siguiente suponemos que tenemos una tabla llamada alumno con las columna id y nota de un grupo de alumnos.

La función es_aprobado devuelve un valor booleano indicando si el alumno está aprobado. El parámetro es el id del alumno.

La función comprueba si hay algún alumno con ese id y si no lo hay finaliza notificando un error.

```
create function es_aprobado(_id int) returns boolean
begin
    declare _nota double;

    if _id not in (select id from alumno) then
        signal sqlstate '45000' set message_text='El alumno no  existe';
    end if;

    set _nota = (select nota from alumno where id=_id);

    if _nota>=5 then
         return true;
    else
         return false;
    end if;
end$$
```

La comprobación de la existencia del alumno es recomendable realizarla utilizando EXISTS o NOT EXISTS.

A partir de ahora este tipo de consultas las haremos así :

```
 if _id not in (select id from alumno) then
    signal sqlstate '45000' set message_text='El alumno no existe';
end if;

if not exists (select 1 from alumno where id=_id) then
    signal sqlstate '45000' set message_text='El alumno no existe';
end if;
```

Recordatorio: Cuando utilizamos EXISTS la lista de campos que escribimos detrás del SELECT es indiferente. Es una costumbre extendida utilizar un 1.