Procedimientos Almacenados (PA) en MySql

Lo primero que explicaré será el comando **DROP PROCEDURE IF EXISTS**, este comando sirve para eliminar el PA si previamente existe (NOTA: este debe ser agregado antes del **DELIMETER**), esto es útil por ejemplo cuando estamos modificando nuestro PA y necesitamos estar actualizándolo constantemente, la sintaxis sería así:

```
DROP PROCEDURE IF EXISTS myProcedure;
```

Bien, una vez visto eso, el siguiente comando que veremos será el **DELIMETER**, se refiere a escribir un delimitador para nuestras consultas SQL, este delimitador se debe especificar cuando vamos a tener varias consultas dentro de nuestro PA para decirle a MySQL que todo lo que este dentro de ese delimitador formará parte de ese PA, tu puedes elegir cualquier delimitador, pero entre los más comunes están:

```
DELIMETER //
     ....
//
```

O bien:

```
DELIMETER $$
    ....
$$
```

Después de escribir nuestro delimitador, vamos a crear nuestro PA con el comando **CREATE PROCEDURE**, cuya sintaxis debe ser así:

```
CREATE PROCEDURE myProcedure({[PARAMS]})
BEGIN
    ....
END
```

Sin duda muchos verán que esto es similar a cuando programabas en Pascal o un lenguaje prehistórico dónde tenías que especificar los bloques de INICIO y FIN, y los parámetros aquí son opcionales.

Parámetros

Cuando creamos un nuevo PA los parámetros son opcionales y sin duda estos son de gran ayuda cuando necesitamos pasar algunos valores, la forma de especificar estos parámetros es la siguiente:

```
CREATE PROCEDURE myProcedure(
  IN _language VARCHAR(2),
```

```
  IN _page INT,
  IN _max INT)
BEGIN
   ....
END
```

Cuando son muchos parámetros una buena técnica de separación y para poder ordenar y ver mejor los parámetros es dar un “enter” a cada uno y separarlos por comas (,). La sintaxis para estos parámetros es simplemente escribir el comando IN seguido del nombre del parámetro (recomiendo poner guiones bajos al principio de cada parámetro, **leer la explicación en el siguiente párrafo**) y después especificar su tipo (INT, VARCHAR, TINYINT, etc.).

Declaración de variables

Cuando creamos un PA en MySQL esté nos permite crear variables para poder asignarles algún valor e inclusive hacer ciertas validaciones, para declarar una variable se debe utilizar el comando DECLARE, **una recomendación personal** es que todas las variables que vayas a crear en tu PA les antepongas un guión bajo (_) y además utilices la técnica de camelCase, esto se debe a que muchas veces utilizamos variables con nombres genéricos que pudieran ser palabras reservadas de MySQL y esto nos puede causar problemas sin que nos demos cuenta en un principio y es muy complicado encontrarlos, por ejemplo si declaramos la variable "status", nos puede causar conflicto puesto que STATUS es una palabra reservada que utiliza MySQL como comando, en este caso para ver los PAs existentes con **SHOW PROCEDURE STATUS**, este caso lo mejor sería declararla como "_status" (pero al final ustedes deciden como llaman sus variables esto solo es una recomendación de muchos conocedores de BD) y nos libramos de problemas, unos ejemplos de variables serían:

```
DECLARE _start INT DEFAULT 0;
DECLARE _limit1 INT DEFAULT 0;
DECLARE _limit2 INT DEFAULT 0;
DECLARE _language VARCHAR(2) DEFAULT 'en';
```

Una vez que hemos declarado nuestras variables si necesitamos asignarle algún valor diferente en algún punto de nuestro código podemos utilizar el comando SET, de la siguiente manera:

```
IF _page > 0 THEN
    SET _limit1 = _page * _max - _max;
END IF;
```

Comparaciones IF... THEN ... ELSE ... END IF

Cómo en todo lenguaje hay veces que necesitamos hacer bifurcaciones a nuestros códigos y tomar caminos distintos dependiendo de la situación, pues en MySQL tenemos la posibilidad de utilizar los IFs al crear nuestros PAs, aunque sin duda se

utilizan un poco de manera un poco "rudimentaria" y me recuerda a Pascal sin duda alguna, pero bueno aquí te dejo algunos bloques de IFs que puedes utilizar.

```
IF _variable1 > 0 AND _variable2 == 1 THEN
    ....
END IF;

IF _variable1 > 0 OR _variable2 == 1 THEN
    ....
ELSE
    ....
END IF;

IF _variable1 == 0 OR _variable2 == 1 THEN
    ....
ELSE
    IF _variable1 <> 5 THEN
        ....
    END IF;
END IF;
```

Selects

Los SELECT en MySQL normalmente los utilizamos para "seleccionar" ciertos campos y hacer ciertas condiciones para que nos regresen ciertos registros dentro de X tabla. Aunque en un PA sirven para lo mismo también tienen otra funcionalidad y es la de "regresar" los valores seleccionados como resultados (es decir, como que si fueran un RETURN de una función, cabe destacar que con PA no se puede utilizar RETURN).

No tan sólo sirve para "Seleccionar" registros de una db, también sirve para "Seleccionar" una variable o ciertos valores que queramos enviar "de regreso cómo resultado".

Un claro ejemplo es cuando tenemos la necesidad de hacer un "debug" a nuestros PA, la manera más fácil de hacerlo es crear un PA llamado debug, de la siguiente manera:

```
DELIMITER $$
CREATE PROCEDURE debug(msg VARCHAR(255))
BEGIN
   SELECT CONCAT("*** ", msg) AS '*** DEBUG:';
END $$
DELIMITER
```

En el PA debug, recibimos cómo parámetro un "mensaje" o texto el cual enviaremos cuando queramos "debuggear" algún valor o variable de otro PA. La función CONCAT cómo su nombre lo indica sirve para concatenar una cadena con otra, en este caso concatenar los *** con el mensaje, y esto para que podamos identificar fácilemente el

mensaje de debug que nos regresa, en otro PA, se puede utilizar simplemente mandandolo a llamar de la siguiente manera:

```
DROP PROCEDURE IF EXISTS getPosts;

DELIMITER $$

CREATE PROCEDURE getPosts(
    IN _language VARCHAR(2),
    IN _page INT,
    IN _max INT)
BEGIN
    DECLARE _start INT DEFAULT 0;
    DECLARE _limit1 INT DEFAULT 0;
    DECLARE _limit2 INT DEFAULT 0;

    SET _limit1 = 0;
    SET _limit2 = _max;

    CALL debug(_limit2);
END $$
DELIMITER ;
```

De esa manera podremos saber que valor tiene la variable "_limit2", y ese valor lo podremos ver en nuestro resultset cuando ejecutemos nuestro PA.

Regresando un poco al tema de los SELECT, cómo decía, en un PA podemos tener tantos SELECT cómo queramos y cada uno nos regresará un resultset, quiere decir, supongamos que usamos PHP para llamar ese PA, normalmente cuando hacemos una consulta "sencilla" en PHP nos regresa los valores en una matriz algo tipo:

```
[0] => Array([0] => ["Campo"] => valor);
```

Pero cuando mandamos 2 ResultSets, en este caso el del Debug y el de los valores, recibiremos algo así (espero se entienda la idea):

```
[0] => Array([0] => ["**Debug:"] => 'EL MENSAJE DEL DEBUG'); //
Debug resultset
[1] => Array([0] => ["Campo"] => valor); // Valores resultset
```

Cómo último consejo les digo que la clausula LIMIT solamente acepta valores de tipo INT divididos en 2 variables, por eso ven las variables _limit1 y _limit2, en un principio yo intentaba hacerlo concatenando los valores en un string tipo "0, 12", pero jamás resulto, es un buen tip que les puede servir, y bueno aquí les dejo el PA que hice ayer y espero les pueda servir un poco cómo ejemplo, es para obtener los posts de un blog, los veo en la siguiente publicación, saludos. No olviden escribir sus dudas en los comentarios.

```
DROP PROCEDURE IF EXISTS getPostsByCategory;

DELIMITER $$

CREATE PROCEDURE getPostsByCategory(
    IN _category VARCHAR(100),
    IN _language VARCHAR(2),
    IN _page INT,
    IN _max INT)
BEGIN
    DECLARE _start INT DEFAULT 0;
    DECLARE _limit1 INT DEFAULT 0;
    DECLARE _limit2 INT DEFAULT 0;

    SET _limit1 = 0;
    SET _limit2 = _max;

    IF _page > 0 THEN
      SET _limit1 = _page * _max - _max;
    END IF;

    SELECT COUNT(1) AS total
      FROM (
        SELECT blog_posts.id FROM blog_posts
          LEFT JOIN blog_re_categories2posts ON
(blog_re_categories2posts.postId = blog_posts.id)
          LEFT JOIN blog_categories ON (blog_categories.id =
blog_re_categories2posts.categoryId)
          WHERE blog_categories.slug = _category
          GROUP BY blog_posts.id
      ) AS Result;

    SELECT  title, blog_posts.slug, excerpt, content, author,
mainImage, createdAt, day, month, year, blog_posts.language,
GROUP_CONCAT(blog_categories.category SEPARATOR ', ') AS categories
      FROM blog_posts
        LEFT JOIN blog_re_categories2posts ON
(blog_re_categories2posts.postId = blog_posts.id)
        LEFT JOIN blog_categories ON (blog_categories.id =
blog_re_categories2posts.categoryId)
        WHERE blog_categories.slug = _category
          AND blog_posts.language = _language
          AND blog_categories.language = _language
          AND blog_posts.situation = 'Published'
      GROUP BY blog_posts.id
      ORDER BY blog_posts.id DESC
      LIMIT _limit1, _limit2;
END $$
DELIMITER ;
```

Para mandar a llamar los PA simplemente se utiliza el comando CALL en nuestra query.

```
CALL myProcedure(1, 2, 'Valor string', 3);

// En PHP sería algo tipo:

$query = mysqli_query("CALL myProcedure(1, 2, 'Valor string', 3)");
```

Parámetros de Entrada y Salida en un Procedimiento

Un procedimiento puede tener uno o más parámetros o también no tener ninguno. Puede carecer de atributos o puede poseer varios. Y como ves, el cuerpo del procedimiento es un bloque de instrucciones definido.

Un parámetro es un dato necesario para el funcionamiento del procedimiento, ya que contribuyen al correcto desarrollo de las instrucciones del bloque de instrucciones.

Los parámetros pueden ser de entrada (IN), salida (OUT) o entrada/salida (INOUT) y deben tener definido un tipo. Un parámetro de entrada en un dato que debe ser introducido en la llamada del procedimiento para definir alguna acción del bloque de instrucciones.

Un parámetro de salida es un espacio de memoria en el cual el procedimiento devolverá almacenado su resultado. Y un parámetro de **entrada/salida** contribuye tanto como a ingresar información útil como para almacenar los resultados del procedimiento. Por defecto, si no indicas el tipo de parámetro MySQL asigna IN.

Para especificar el tipo de parámetro seguimos la siguiente sintaxis:

```
[{IN|OUT|INOUT} ] nombre TipoDeDato
```

Atributos de un Procedimiento en MySQL

Son características adicionales para establecer la naturaleza del procedimiento. Veamos la utilidad de algunas:

LANGUAGE SQL: Indica que el procedimiento se escribirá en lenguaje estándar **SQL/PSM**. Pero su utilidad se basa en la suposición de que en el futuro los procedimientos podrían ser escritos en otros lenguajes como **Php**, **Java**, etc. Ya que aun los escribimos en SQL entonces no es necesario ponerlo.

SQL SECURITY {DEFINER|INVOKER}: Establece el nivel de seguridad de invocación de un procedimiento. Si usas DEFINER el procedimiento sera ejecutado con los permisos del usuario que lo creó, y si usas INVOKER será ejecutado con los permisos del usuario que lo esta invocando.

[NOT] DETERMINISTIC: Especifica si el procedimiento devolverá siempre el mismo resultado al ingresar los mismo parámetros. O si devolverá distintos resultados al ingresar los mismo resultados. Un ejemplo sería ingresar la suma 1+2, se sabe que siempre el resultado será 3, así que usamos DETERMINISTIC. Pero si el parámetro es un valor de un retiro de cuenta bancaria, el resultado del saldo que queda será diferente sin importar la cantidad retirada.

NO SQL|CONTAINS SQL|READS SQL DATA|MODIFIES SQL DATA: Estas características determinan la estructura del procedimiento. NO SQL indica que el procedimiento no contiene sentencias del lenguaje SQL. READS SQL DATA especifica que el procedimiento lee información de la base de datos mas no escribe datos. MODIFIES SQL DATA indica que el procedimiento escribe datos en la base de datos. CONTAINS SQL es el tipo por defecto de un procedimiento e indica que el procedimiento contiene sentencias SQL

COMMENT cadena: Con este atributo podemos añadir una **descripción** al procedimiento con respecto a las instrucciones que ejecuta. Por ejemplo, *"Este procedimiento da de baja a todos los clientes que hace 3 meses no compran a la compañía"*.

Ejemplo de un Procedimiento con un parámetro IN

En el siguiente ejemplo desarrollemos un procedimiento para el siguiente requerimiento:

Imprima los números del 1 hasta n, donde n esta dado por el usuario.

Usaremos un procedimiento para capturar el numero n del usuario. Incorporaremos una variable contadora que comience en 1 y un WHILE para el incremento e impresión. Veamos:

```
DELIMITER //

CREATE PROCEDURE numeros_1_hasta_n (IN n INT)
BEGIN
DECLARE contador INT DEFAULT 1;
WHILE contador<=n DO
SELECT contador;
SET contador = contador + 1 ;
END WHILE;
END//

DELIMITER ;
```

*La sentencia DELIMITER cambia el carácter de terminación ';' por cualquier otro carácter, en este caso elegimos '//'. Se hace con el fin de que **MySQL** no termine el procedimiento al encontrar el primer punto y coma. Al final restablecemos el valor original del caracter de escape.*

¿Como ejecuto un procedimiento ya almacenado?

Usaremos el comando CALL enseguida del nombre del procedimiento y si tiene parámetros, entonces se ingresan sus parámetros. Ahora veamos como llamar al anterior procedimiento:

```
CALL numeros_1_hasta_n(5)
```

Veamos el resultado:

Modificar un Procedimientos en MySQL

*Para modificar un procedimiento en **MySQL** usaremos la sentencia ALTER PROCEDURE. Esta modificación permite cambiar el **nivel de seguridad** y la **descripción del procedimiento**.*

```
ALTER PROCEDURE nombre
[SQL SECURITY {DEFINER|INVOKER}]
[COMMENT descripción]
```

Ejemplo: Cambiar la descripción de un Procedimiento

A continuación mostraremos un procedimiento que inserta un cliente en la base de datos.

```
DELIMITER //
CREATE PROCEDURE insertar(id_cliente INT, nombre_cliente VARCHAR(100), apellido_cliente
VARCHAR(100))
COMMENT 'Procedimiento que inserta un cliente a la base de datos'
BEGIN
IF NOT EXISTS ( SELECT C.ID
FROM CLIENTE AS C
WHERE C.ID = id_cliente) THEN
```

```
INSERT INTO CLIENTE(ID, NOMBRE, APELLIDO)
VALUES ( id_cliente,nombre_cliente,apellido_cliente);
ELSE
SELECT 'Este cliente ya existe en la base de datos!';
END IF;
END//

DELIMITER ;
```

Ahora le cambiaremos la descripción mediante ALTER PROCEDURE:

```
ALTER PROCEDURE insertar_cliente
COMMENT 'Insertar Cliente'
```

Borrar un Procedimiento en MySQL

*La sentencia DROP permite borrar un procedimiento de **MySQL**. Obviamente el procedimiento debe estar almacenado con anterioridad para poder llevar a cabo la operación. Dado el caso de que no se encuentre creado, podemos usar el operador EXISTS junto al condicional IF para comprobar su existencia.*

DROP PROCEDURE [IF EXISTS] nombre_procedimiento

Ejemplo: Eliminar un Procedimiento antes de crearlo

La eliminación de un procedimiento se lleva a cabo con DROP PROCEDURE. Es común usa esta sentencia antes de crear un procedimiento.

*Como ejemplo crearemos un **procedimiento** llamado mostrar_clientes, el cual simplemente consulta todas las columnas de una tabla llamada CLIENTE:*

```
DROP PROCEDURE IF EXISTS mostrar_clientes;
CREATE PROCEDURE mostrar_clientes()
SELECT * FROM CLIENTE;
```

Si el procedimiento que deseamos crear ya existe, entonces lo borraremos para darle paso a la nueva definición.

Mostrar un Procedimiento en MySQL

*Para **mostrar las características** de un procedimiento en **MySQL** usaremos la sentencia SHOW CREATE PROCEDURE.*

```
SHOW CREATE PROCEDURE nombre_procedimiento
```