AUTOMATIZACIÓN DE TAREAS: CONSTRUCCIÓN DE GUIONES

Procedimientos y funciones almacenados

Las rutinas (procedimientos y funciones) almacenados son un conjunto de comandos SQL que pueden almacenarse en el servidor. Una vez que se hace, los clientes no necesitan lanzar cada comando individual sino que pueden llamar al procedimiento almacenado como un único comando.

Una función almacenada es un programa almacenado que devuelve un valor. Si bien los procedimientos almacenados pueden devolver valores a través de parámetros OUT o INOUT en las funciones solo se devuelven a través de cero o un único valor de retorno. Las funciones almacenadas se pueden usar en expresiones y pueden incluirse en otras funciones o procedimientos así como en el interior de sentencias SQL como SELECT, UPDATE, DELETE e INSERT.

El esquema general de una rutina almacenada se resume en el siguiente esquema:

1. Nombre rutina +parámetros (entrada, salida o E/S)
2. Declaración e inicialización de variables
3. Procesamiento datos (Bloques BEGIN/END con instrucciones de control condicionales y repetitivas)
4. Fin (con la instrucción RETURN para devolver un valor en caso de funciones almacenadas

La diferencia entre una función y un procedimiento es que la función devuelve valores. Estos valores pueden ser utilizados como argumentos para instrucciones SQL, tal como lo hacemos normalmente con otras funciones como son, por ejemplo, MAX() o COUNT().

El formato de las funciones y procedimiento es básicamente el mismo, pero las funciones devuelven un valor en algún punto de su código y deben declarar el tipo de valor devuelto en la cabecera.

Utilizar la cláusula RETURNS es obligatorio al momento de definir una función y sirve para especificar el tipo de dato que será devuelto (sólo el tipo de dato, no el dato).

Los procedimientos se llaman con CALL y las funciones con SELECT.

Sintaxis y ejemplos de rutinas almacenadas

La sintaxis general para la creación de un procedimiento o función es:

```
CREATE PROCEDURE nombre ([parameter[,…]])
[characteristic…] routine_body
CREATE FUNCTION nombre ([parameter[,…]])
RETURNS type
[characteristic…] routine_body
Parameter:
[IN/OUT/INOUT] param_name type
Type:
Any valid MYSQL data type
Characteristic:
LANGUAGE SQL:
| [NOT] DETERMINISTIC
| {CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL
DATA}
| SQL SECURITY {DEFINER |INVOKER}
```

- **Nombre**: es el nombre de la rutina almacenada
- **Parameter**: Son los parámetros que en general se caracterizan por un tipo y un nombre. El tipo puede ser de entrada (IN), salida (OUT) o entrada/salida (INOUT)
- **Routinebody**: es el cuerpo de la rutina formado generalmente por sentencias SQL. En caso de haber más de una deben ir dentro de un bloque delimitado por sentencias BEGIN-END.
- **Deterministic**: indica si es determinista o no, es decir, si siempre produce el mismo resultado.
- **Contains SQL/no SQL**: especifica si contiene sentencias SQL o no.
- **Modifies SQL DATA/Reads SQL data**: indica si las sentencias modifican o no los datos.l
- **SQL security**: Determina si debe ejecutarse con permisos del creador (DEFINER) o del que lo invoca (INVOKER).

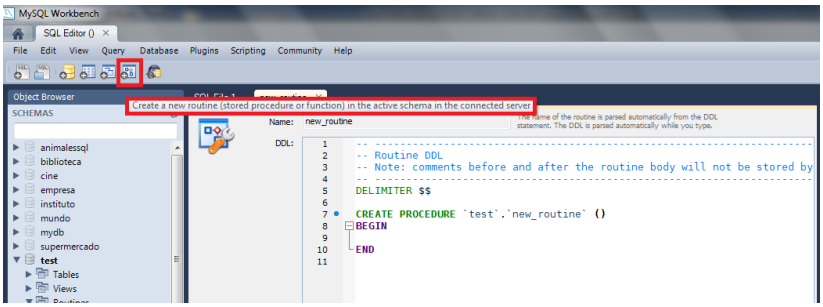
Los corchetes indican parámetros opcionales y las palabras en mayúsculas son reservadas de SQL. Todos los procedimientos o funciones se crean asociados a una base de datos que será la activa en ese momento o la que pongamos como prefijo en el nombre del mismo.

De forma resumida, una rutina obedece al siguiente esquema simplificado:

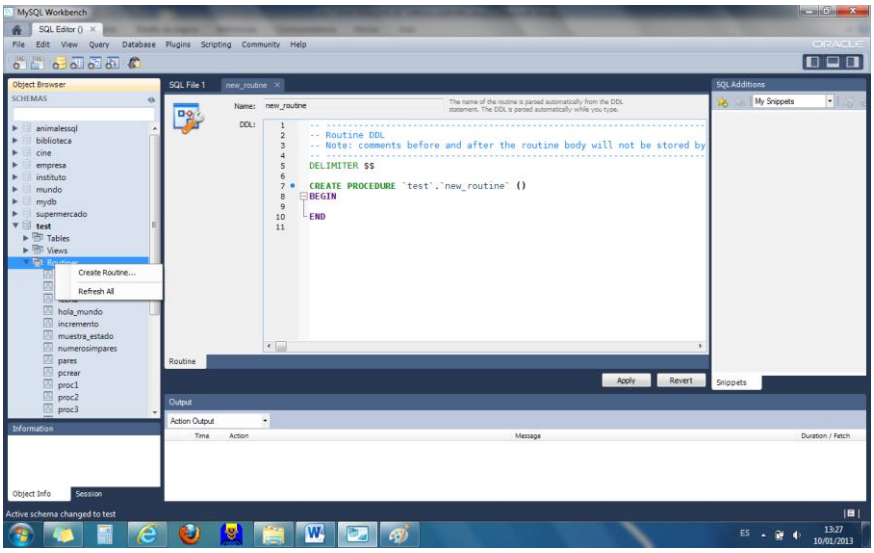
```
Nombre (parámetros) + modificadores
Begin
Declaración (DECLARE) y establecimiento de variables (SET)
Proceso de datos (instrucciones sql/ instrucciones de
control)
End
```

Ejemplo 1

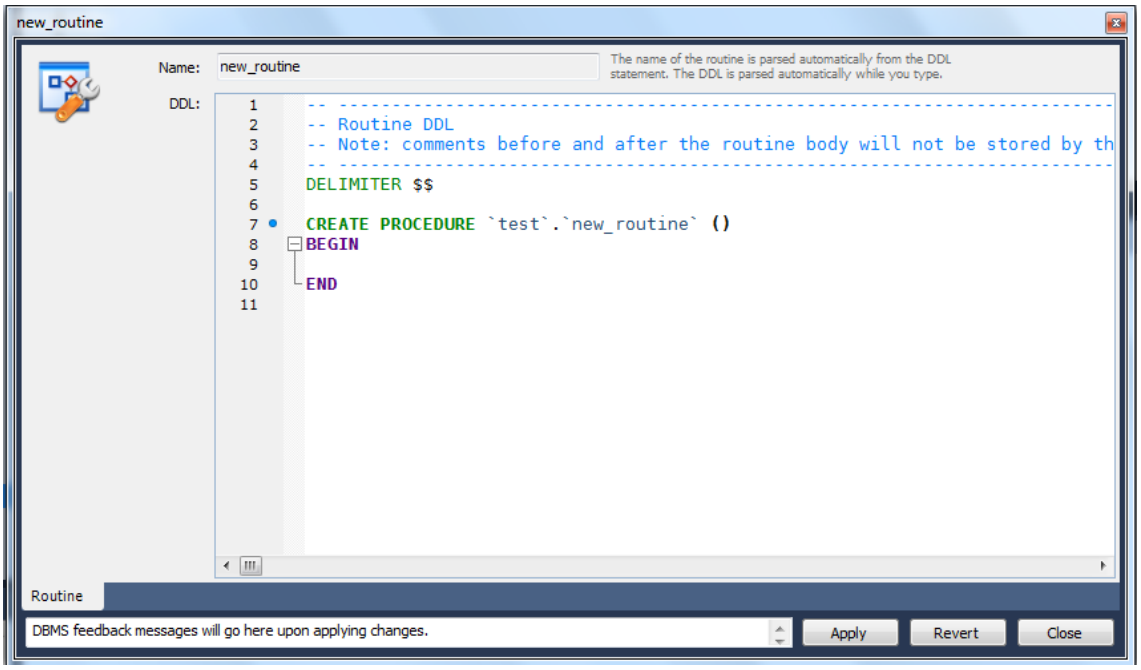
Vamos a seleccionar el esquema test como esquema por defecto. A continuación, pulso sobre la opción **Create a new routine** tal y como se muestra en la pantalla que aparece a continuación:



También puedo hacerlo pulsando sobre Routines en el esquema test con el botón derecho y seleccionando CreateRoutine en el menú que aparece. Se muestra a continuación.



Me aparecerá la siguiente pantalla:



En ella escribiré lo siguiente:

```
DELIMITER $$
DROP PROCEDURE IF EXISTS hola_mundo$$
CREATE PROCEDURE test.hola_mundo()
BEGIN
SELECT 'hola mundo';
END$$
```

Esta rutina lo que va a hacer es que aparezca en pantalla la cadena "hola mundo". La palabra DELIMITER indica el carácter de comienzo y fin del procedimiento. Típicamente sería ; pero como ese carácter se usa para cada sentencia SQL vamos a utilizar otro carácter (normalmente $$ o //).

A continuación, elimino el procedimiento por si existe. Esto evita errores cuando queremos modificar un procedimiento que ya existe.

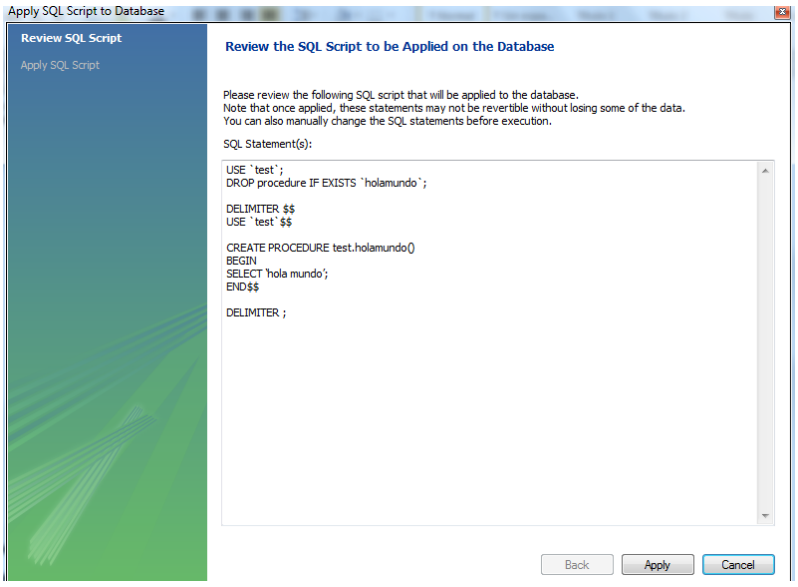
La siguiente línea define el procedimiento en el que aparece el nombre seguido por paréntesis entre los que pondremos los parámetros en caso de haberlos. En este caso, precedo el nombre con el esquema test al que pertenece el procedimiento.

BEGIN indica el comienzo de una serie de bloques de sentencias SQL que componen el cuerpo del procedimiento cuando hay más de una.

Después aparece el conjunto de sentencias SQL, es este caso es un SELECT que muestra la cadena por pantalla.

END indica el final del procedimiento. Detrás se pone $$ indicando que ya hemos terminado el procedimiento.

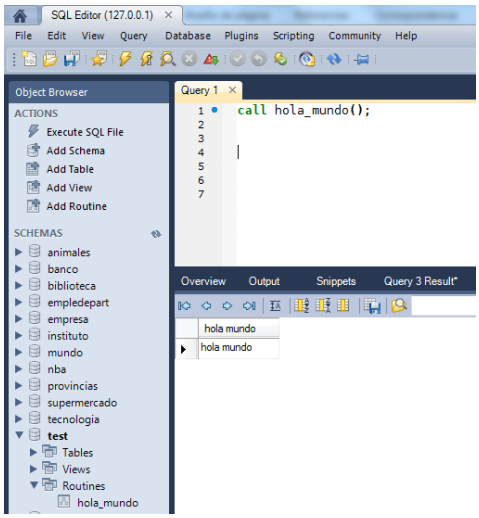
Pulso Apply y aparece la siguiente pantalla



Vuelvo a pulsar Apply y doy a Finish.

Para ejecutar la rutina, escribo CALL hola_mundo();

Y obtengo la siguiente pantalla:



Las sentencias BEGIN Y END sólo son necesarias en caso de tener más de una sentencia.

Ejemplo 2

```
DELIMITER $$
CREATE PROCEDURE fecha ()
LANGUAGE SQL
NOT DETERMINISTIC
COMMENT "A PROCEDURE"
SELECT CURDATE(), RAND() $$
```

En LANGUAGE se indica el lenguaje , NOT DETERMINISTIC indica que la rutina no producirá siempre el mismo resultado cada vez que se ejecuta y COMMENT indica comentarios. La rutina obtendrá la fecha actual y un número aleatorio por pantalla.

Lo ejecutaré poniendo CALL fecha();

Ejemplo 3

```
DELIMITER $$
DROP FUNCTION IF EXISTS colores$$
CREATE FUNCTION colores(a CHAR)
RETURNS VARCHAR(20)
DETERMINISTIC NO SQL
BEGIN
    DECLARE color VARCHAR(20);
    IF a="A" THEN
        SET color="azul";
    ELSEIF a="V" THEN
        SET color="verde";
    ELSEIF a="R" THEN
```

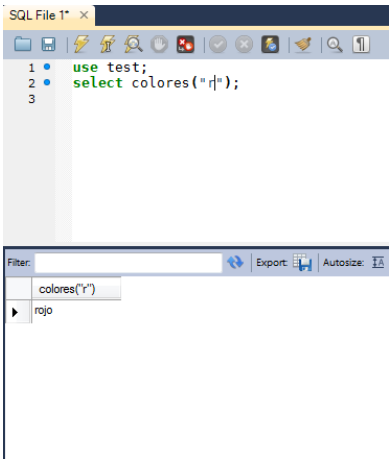
```
            SET color="rojo";
      ELSE
            SET color = "Nada";
      END IF;
      RETURN color;
END$$
```

```
SELECT colores('A'), colores('R'),colores('V'), colores('J');
```

Esta función recibe un valor de estado (A-V-R) y nos va a devolver azul, verde o rojo, dependiendo de que sea un valor u otro. Para ejecutar la función pondré

```
SELECT colores(valor);
```

Siendo valor uno de los tres posibles valores A-V-R. Dicho valor, al ser de tipo texto, irá entre comillas.

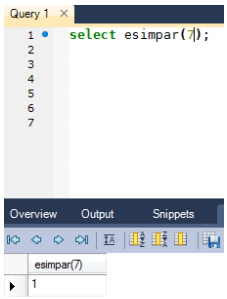


Ejemplo 4

```
DELIMITER $$

CREATE FUNCTION esimpar (numero int)
RETURNS int
DETERMINISTIC NO SQL
BEGIN
DECLARE impar int;
    IF MOD(numero,2)=0 THEN SET impar=0;
else SET impar=1;
    END IF;
RETURN impar;
END;$$
```

En este caso recibimos un número como entrada devolviendo 0 si es par y 1 si es impar.



Ejercicio 1

Modifica la función anterior y llámala esImparV2 para que si el número es impar devuelva la cadena “impar” y si es par devuelva la cadena “par”.

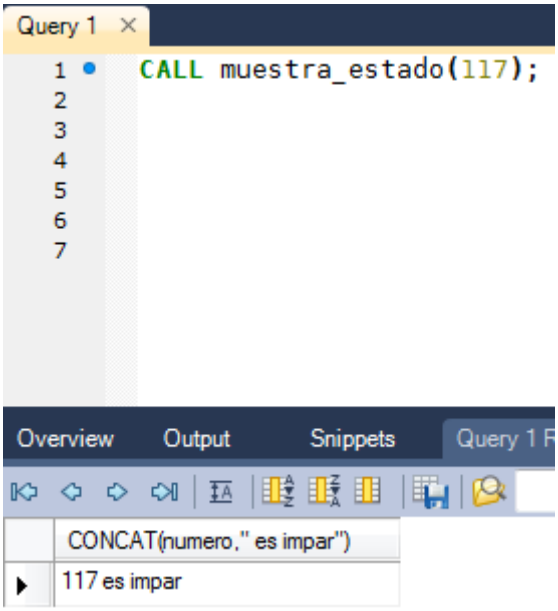
Modifica la función anterior y llámala esImparV3 para que si el número es impar devuelva verdadero y si es par devuelva falso.

Ejemplo 5

Es usual llamar a las funciones desde otras funciones o procedimientos como en el siguiente ejemplo:

```
DELIMITER $$
DROP PROCEDURE IF EXISTS muestra_estado$$
CREATE PROCEDURE muestra_estado(in numero int)
BEGIN
IF (esimpar(numero)) THEN
    SELECT CONCAT(numero," esimpar");
ELSE
SELECT CONCAT(numero," es par");
END IF;
END;$$
```

Este procedimiento nos muestra un mensaje indicando si el número que ponemos en el procedimiento es par o impar.



De este modo, las funciones permiten reducir la complejidad del código encapsulándolo y simplificando su complejidad.

Parámetros y variables

Ejemplo 6

Crea dentro del esquema test una tabla llamada t con dos campos de tipo INT que se llamen a y b. A continuación, crea la siguiente rutina:

```
DELIMITER $$
DROP PROCEDURE IF EXISTS proc1 $$
CREATE PROCEDURE proc1
(IN parametro1 INT)
BEGIN
DECLARE variable1 INT;
DECLARE variable2 INT;
IF parametro1=17 THEN
SET variable1=parametro1;
SET variable2=10;
ELSE
SET variable1=10;
SET variable2=30;
END IF;
INSERT INTO t (a,b) VALUES
(variable1,variable2);
END$$
```

Este nuevo ejemplo lo que va a hacer es que, dependiendo del valor que meta como parámetro asignará unos valores a los campos de la tabla t u otros.

DECLARE →Crea una nueva variable con su nombre y tipo. Los tipos son los usados en MySQL. Esta cláusula puede incluir una opción para indicar valores por defecto. Si no se indican, dichos valores serán NULL.

Por ejemplo:

```
DECLARE a,b INT DEFAULT 5;
```

Crea dos variables enteras con valor 5 por defecto.

SET →Permite asignar valores a las variables usando el operador de igualdad.

En el ejemplo se recibe una variable entera de entrada llamada parametro1. A continuación, se declaran dos variables variable1 y variable2 de tipo entero y se testea el valor del parámetro. Si es 17 se asigna su valor a variable1 y 10 a la variable 2 y si no se asigna 10 a la variable1 y 30 a la variable2. Por último, se guardan esos valores en los campos de la tabla t.

El ejemplo no tiene sentido, sólo valor didáctico.

Tipos de parámetros

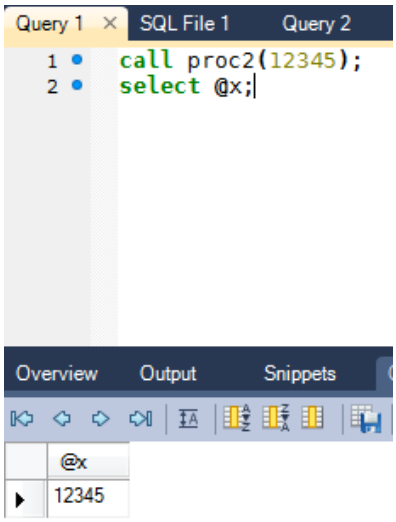
También observamos la posibilidad de incluir un parámetro. Existen tres tipos de parámetros:

- ***IN:*** Es el tipo por defecto y sirve para incluir parámetros de entrada que usará el procedimiento.
- ***OUT:*** Parámetros de salida. El procedimiento puede asignar valores a dichos parámetros que son devueltos en la llamada
- ***INOUT:*** Permite pasar valores al proceso que serán modificados y devueltos en la llamada

Ejemplo IN

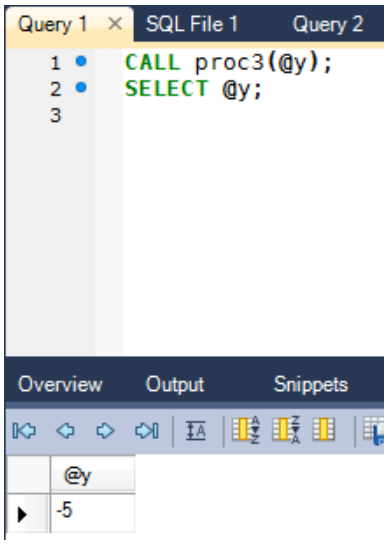
```
DELIMITER $$
CREATE PROCEDURE proc2 (IN P int)
SET @x=p
$$
```

En este ejemplo de procedimiento se establece el valor de una variable de sesión (precedida por @) al valor de entrada p.



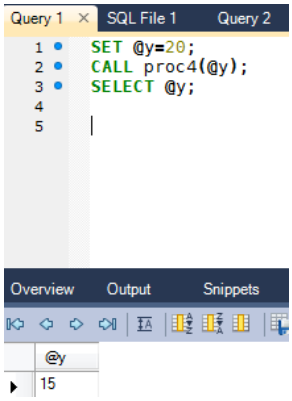
Ejemplo OUT

```
DELIMITER $$
CREATE PROCEDURE proc3 (OUT p INT) SET P=-5 $$
```



Ejemplo INOUT

```
DELIMITER $$
CREATE PROCEDURE proc4(INOUT p INT) SET p=p-5 $$
```



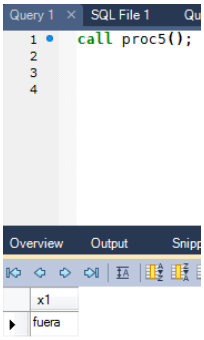
En este caso usamos el mismo valor del parámetro de entrada para decrementar su valor previamente asignado con SET.

Alcance de las variables

Las variables tienen un alcance que está determinado por el bloque BEGIN/END en el que se encuentran. Es decir, no podemos ver una variable que se encuentra fuera de un procedimiento salvo que la asignemos a un parámetro OUT o a una variable de sesión (usando la @). Lo vemos en el siguiente ejemplo:

```
DELIMITER $$
CREATE PROCEDURE proc5 ()
BEGIN
DECLARE x1 char(6) DEFAULT "fuera";
BEGIN
DECLARE x1 CHAR(6) DEFAULT "dentro";
SELECT x1;
END;
SELECT x1;
END;$$
```

Las variables x1 del primer y segundo bloque BEGIN/END son distintas, sólo tienen validez dentro del bloque como se muestra en la llamada al procedimiento. Al final me escribe "fuera" pues es el valor que se corresponde con el último BEGIN/END, aunque la última asignación de x1 sea "dentro".



Ejercicios

2. Sobre el esquema de pruebas test crea un procedimiento para mostrar el año actual.
3. Crea un procedimiento que muestre las tres primeras letras de una cadena pasada como parámetro en mayúsculas. Usa las funciones UPPER para pasar a mayúsculas (LOWER pasa a minúsculas) y LEFT(cadena,n) para quedarse con los n primeros caracteres de la izquierda.
4. Crea un procedimiento que muestre dos cadenas pasadas como parámetros concatenadas y en mayúsculas. Quiero que si una de las cadenas es nula, aparezca la otra. Que cuando las dos son nulas, que aparezca cadena vacía.
5. Crea una función que devuelva el valor de la hipotenusa de un triángulo a partir de los valores de sus lados. Busca las funciones que hacen referencia a la raíz cuadrada y a la elevación al cuadrado en el manual de MySQL (pow, sqrt).

Instrucciones condicionales

En muchos casos, el valor de una o más variables o parámetros determinará el proceso de las mismas. Cuando esto ocurre debemos usar instrucciones condicionales de tipo simple o IF cuando sólo hay una condición, alternativas (IF-THEN-ELSE) cuando hay dos posibilidades o CASE cuando hay un conjunto de condiciones distintas.

IF-THEN-ELSE

La sintaxis general es:

```
IF exp1 THEN
...
ELSEIF exp2 THEN
...
ELSE
```

```
…
END IF
```

Cuando hay muchas condiciones es más apropiado usar la instrucción CASE. La sintaxis general de CASE es:

```
CASE expresión
      WHEN value THEN
            Acción 1
      [WHEN value THEN
            Acción 2…….]
      [ELSE
            Acción 3]
END CASE;
```

Donde expresión puede tomar varios valores, dependiendo de los cuales se realiza una acción u otra.

Ejemplo 7

Vamos a crear una función que devuelva MAYOR o MENOR dependiendo de que el número que metamos sea mayor o menor de 5.

```
DELIMITER $$
CREATE function nummayor(num int)
returnsvarchar(5)
DETERMINISTIC NO SQL
BEGIN
DECLARE textomayor varchar(5);
If num>5 then
Set textomayor="MAYOR";
    ELSE
Set textomayor="MENOR";
    END IF;
RETURN textomayor;
END$$
```

Ejercicios

6. Crea una función que pida dos números y devuelva 1 si el primer número es divisible por el segundo y 0 en caso contrario.
7. Usa las estructuras condicionales para mostrar el día de la semana según un valor de entrada numérico, 1 para lunes, 2 martes…. Dentro de una función.

Instrucciones repetitivas o loops

Los loops permiten iterar un conjunto de instrucciones un número determinado de veces. Para ello MySQL dispone de tres tipos de instrucciones: Simple Loop, Repeat-Until y While-Loop.

Simple loop

Su sintaxis básica es:

```
[etiqueta:] LOOP
Instrucciones
END LOOP [etiqueta];
```

Donde la palabra opcional *etiqueta* permite etiquetar el loop para podernos referir a él dentro del bloque.

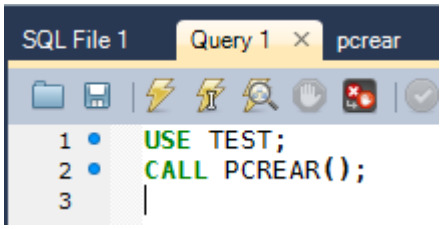
Ejemplo 8

Vamos a crear un procedimiento que va a insertar en una tabla llamada nueva (que tienes que crear en el esquema test: contendrá un único campo de tipo int(el valor de un contador) que va a comenzar con el valor 1 y va a incrementarse en 1 en cada pasada por el bucle.

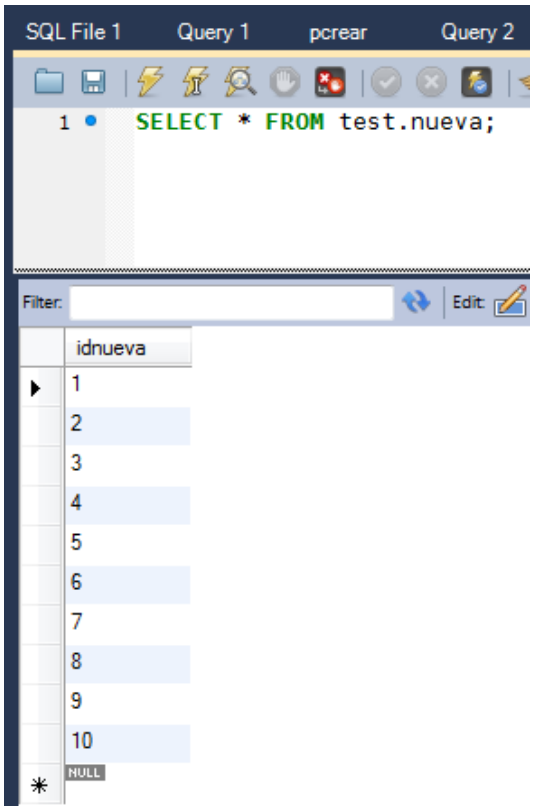
El procedimiento a escribir será el siguiente:

```
DELIMITER $$
CREATE PROCEDURE pcrear ()
BEGIN
DECLARE cont INT;
SET cont=1;
BUCLE1: LOOP
    INSERT INTO NUEVA VALUES (cont);
    SET cont=cont+1;
    IF cont>10 THEN
        LEAVE BUCLE1;
    END IF;
END LOOP BUCLE1;
END;$$
```

Para ejecutarlo tengo que poner:



El resultado que se obtendrá será el siguiente:



Repeat until loop:

Su sintaxis general es:

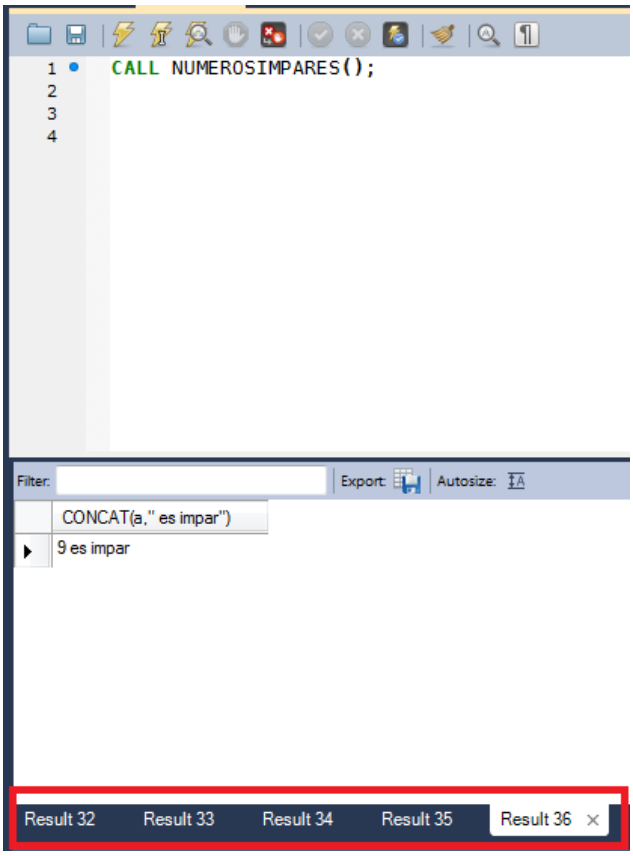
```
[etiqueta:] REPEAT
Instrucciones
UNTIL expresion
END REPEAT [etiqueta]
```

Ejemplo 9

En este ejemplo se muestran los números impares de 0 a 10.

```
DELIMITER $$
CREATE PROCEDURE `numerosimpares`()
BEGIN
    DECLARE a int;
    SET a=0;
REPEAT
        SET a=a+1;
      if MOD(a,2)<>0 THEN
        SELECT CONCAT(a," esimpar");
        END IF;
 UNTIL a>=10
 END REPEAT;
END;$$
```

El resultado al ejecutar es el siguiente:



¿Cómo harías para que devolviera el número de números impares detectados? ¿Cómo harías para que este número se fuera acumulando entre las distintas llamadas al procedimiento?

While Do

Su sintaxis general es:

```
[etiqueta:] WHILE expresión DO
Instrucciones
END WHILE [etiqueta]
```

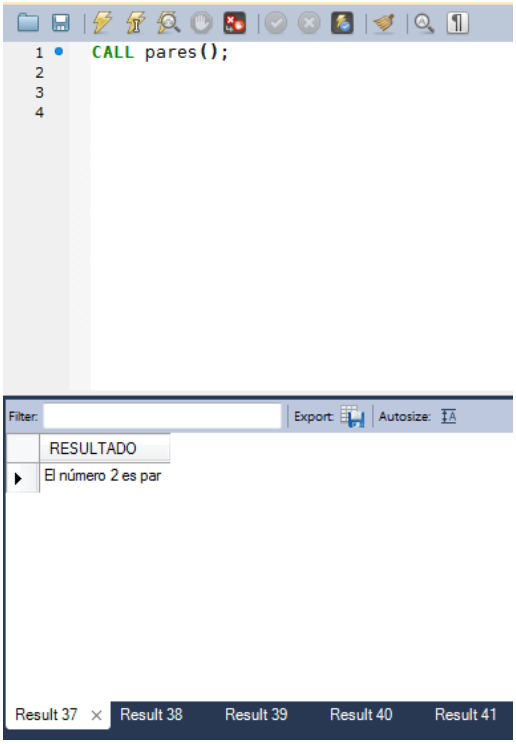
Ejemplo 10

El siguiente ejemplo es igual que el anterior pero mostrando los pares entre 0 y 10 usando WHILE:

```
DELIMITER $$

CREATE  PROCEDURE pares()
BEGIN
DECLARE I int;
SET i=1;
B3: WHILE i<=10 DO
    IF MOD(i,2)=0 THEN
        SELECT  CONCAT("El  número  ",i,"  es  par")  as
RESULTADO;
    END IF;
    SET i=i+1;
END WHILE B3;
END
```

El resultado será el siguiente:



Para poder ver todas las filas, vamos grabando cada fila en una tabla prueba con un campo varchar(30) donde concatenamos “el numero”,i,”es par”, etc.

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `pares`()
BEGIN
DECLARE i int;
SET i=1;
B3: WHILE i<=10 DO
    IF MOD(i,2)=0 THEN
        SELECT CONCAT("El número ",i," es par") as
RESULTADO;
        insert into prueba values (concat("el numero ",i,"es
par"));
    END IF;
    SET i=i+1;
END WHILE B3;
END
```

Ejercicios

8. Sobre el esquema test, crea un procedimiento que muestre la suma de los primeros n números enteros, siendo n un parámetro de entrada.
9. Haz un procedimiento que muestre la suma de los términos 1/n con n entre 1 y m, es decir, ½+1/3+…+1/m, siendo m el parámetro de entrada. Ten en cuenta que m no puede ser 0, en cuyo caso, se deberá validar y mostrar el error correspondiente.