**INTRODUCCIÓN FUNCIONES Y PROCEDIMIENTOS**

**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 a la rutina almacenada 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 o de sentencias SELECT, en las funciones solo se devuelven a través de 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.

**PROCEDIMIENTOS**

La sintaxis general para la creación de un procedimiento es:

**CREATE PROCEDURE *nombreprocedimiento* ([parámetros de entrada[,….]])**

**BEGIN**

***Cuerpo del procedimiento***

**END**

Los parámetros de entrada se caracterizan por un tipo y un nombre. Habrá que indicar el nombre y el tipo de parámetro (si es INT, VARCHAR…). Si hay más de un parámetro de entrada, se separarán por comas.

El cuerpo del procedimiento está compuesto por sentencias SQL. En caso de haber más de una deben ir dentro de un bloque delimitado por sentencias BEGIN-END.

El cuerpo del procedimiento podrá contener:

Declaración (DECLARE) y establecimiento de variables (SET)

Proceso de datos (instrucciones sql/ instrucciones de control)

A la hora de ejecutarlos se les llama con **CALL nombreprocedimiento.** Si tiene parámetros de entrada, se incluirán a continuación entre paréntesis.

**FUNCIONES**

La sintaxis general para la creación de una función es:

**CREATE FUNCTION *nombrefuncion* ([parámetros de entrada[,….]])**

**RETURNS tipodato**

**BEGIN**

***Cuerpo de la función***

**RETURN dato**

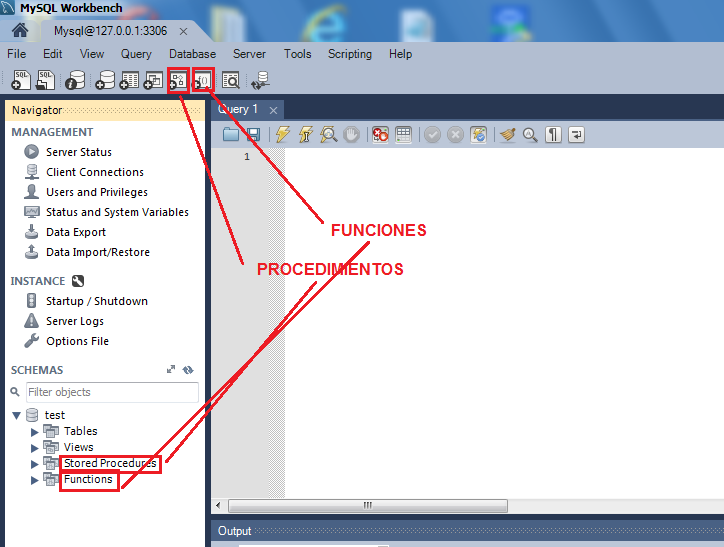
**END**

Las funciones siempre devuelven un valor y deben declarar el tipo de valor que va a devolverse en la cabecera. Para ello se usa la cláusula RETURNS y sirve para especificar el tipo de dato que será devuelto (sólo el tipo de dato, no el dato).

Además, dentro del cuerpo de la función tiene que aparecer la clausula RETURN que devolverá un dato que ha de coincidir con el tipo de dato definido en RETURNS.

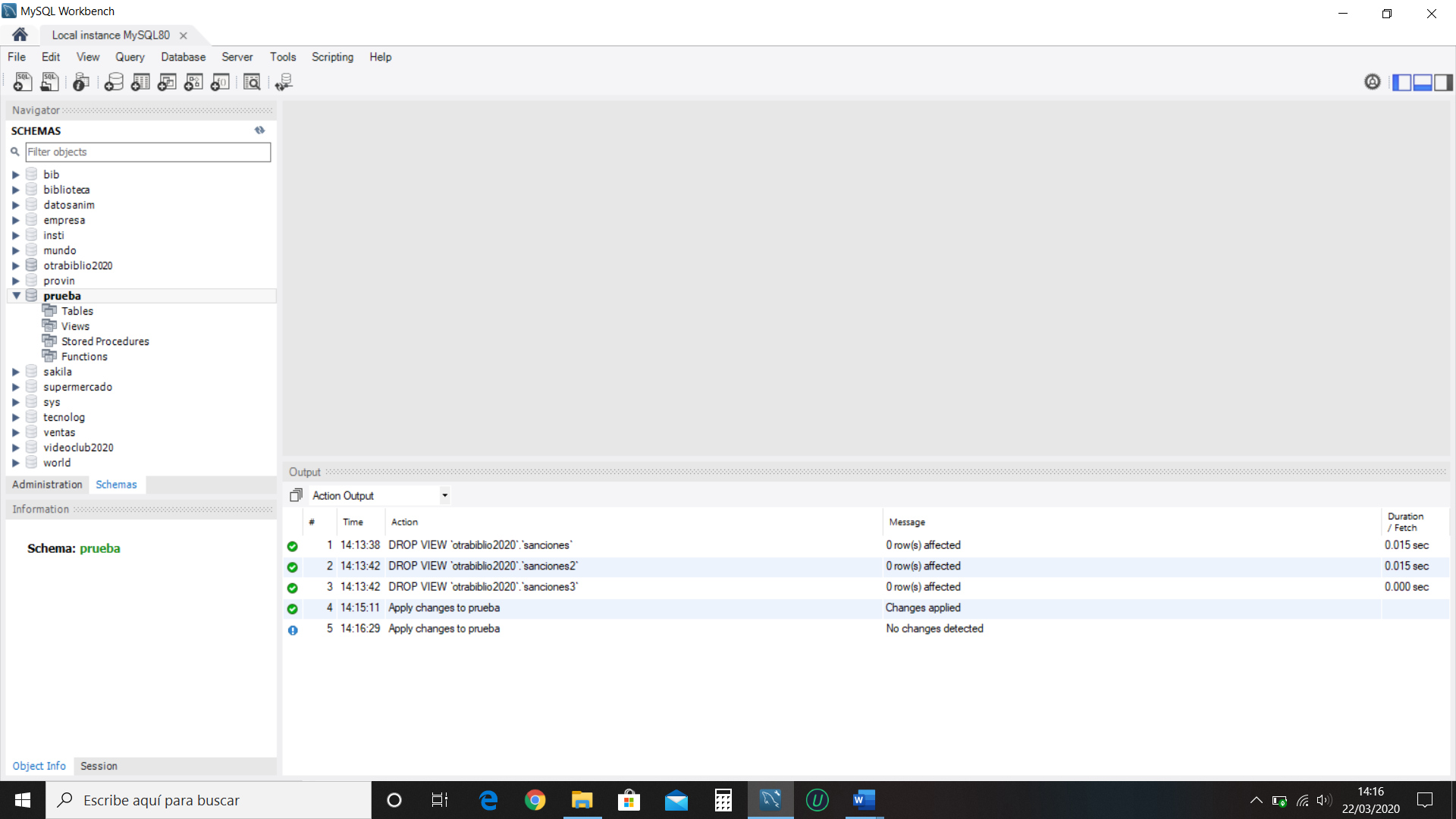
Las funciones se ejecutan con **SELECT nombrefuncion.** Si tiene parámetros de entrada, se incluirán a continuación entre paréntesis. Recuerda que has usado ya muchas funciones de las que tiene MySQL predefinidas: CURDATE(), MAX(salario), SUM(unidades)…. Y todas iban incluidas dentro del SELECT.

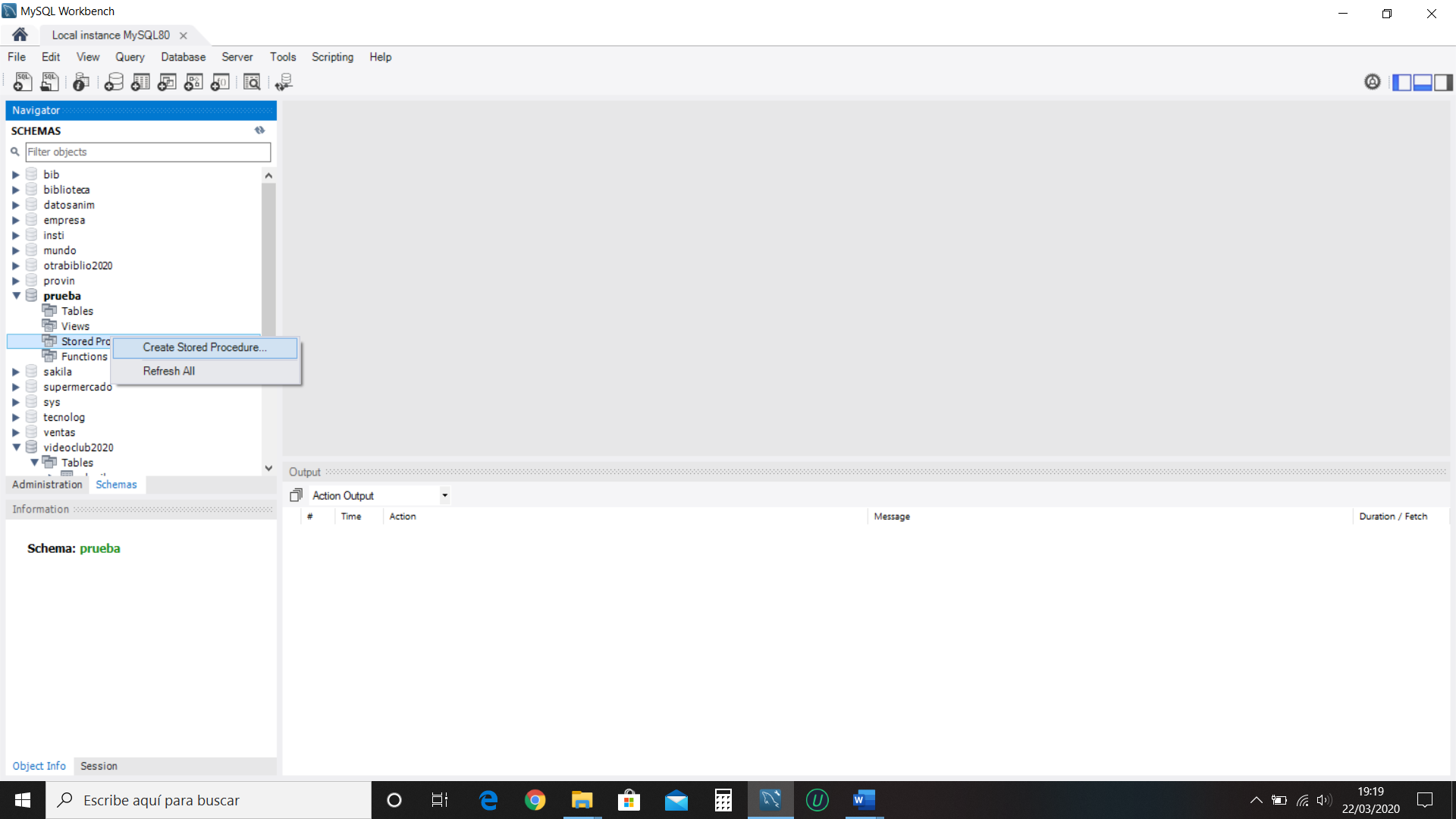
Para crear funciones o procedimientos en Workbench:



Para crear una función o procedimiento, si elijo las opciones que hay debajo de los esquemas, tendré que pulsar con el botón derecho sobre cualquiera de las dos opciones.

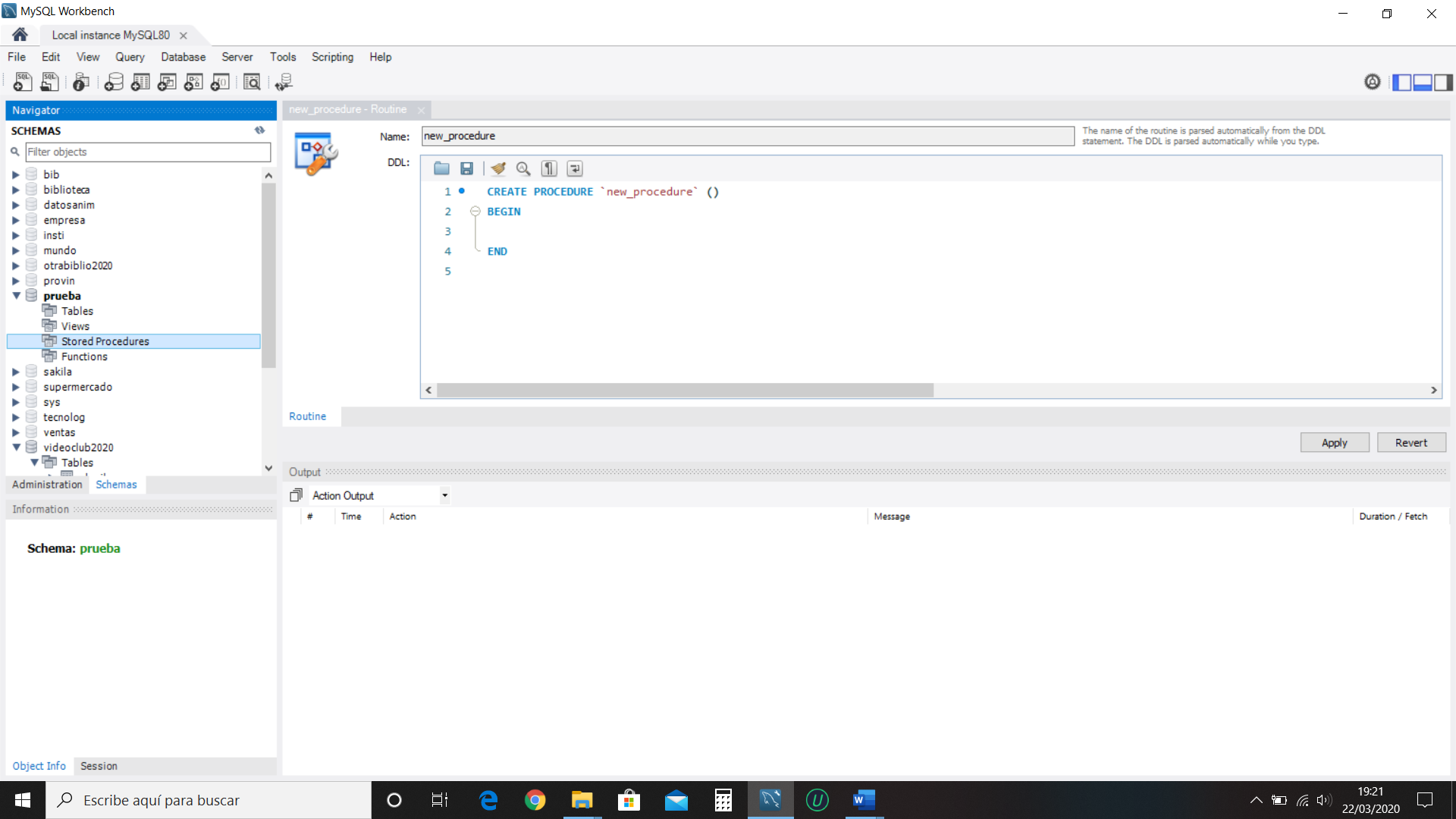
Voy a crear un esquema que se llame prueba, sin tablas de momento en el que os voy a crear algún procedimiento y función sencillos.





Pulso con el botón derecho sobre Stored Procedures y selecciono **Create Stored Procedure.**

Aparecerá la siguiente pantalla:



**EJEMPLO 1**

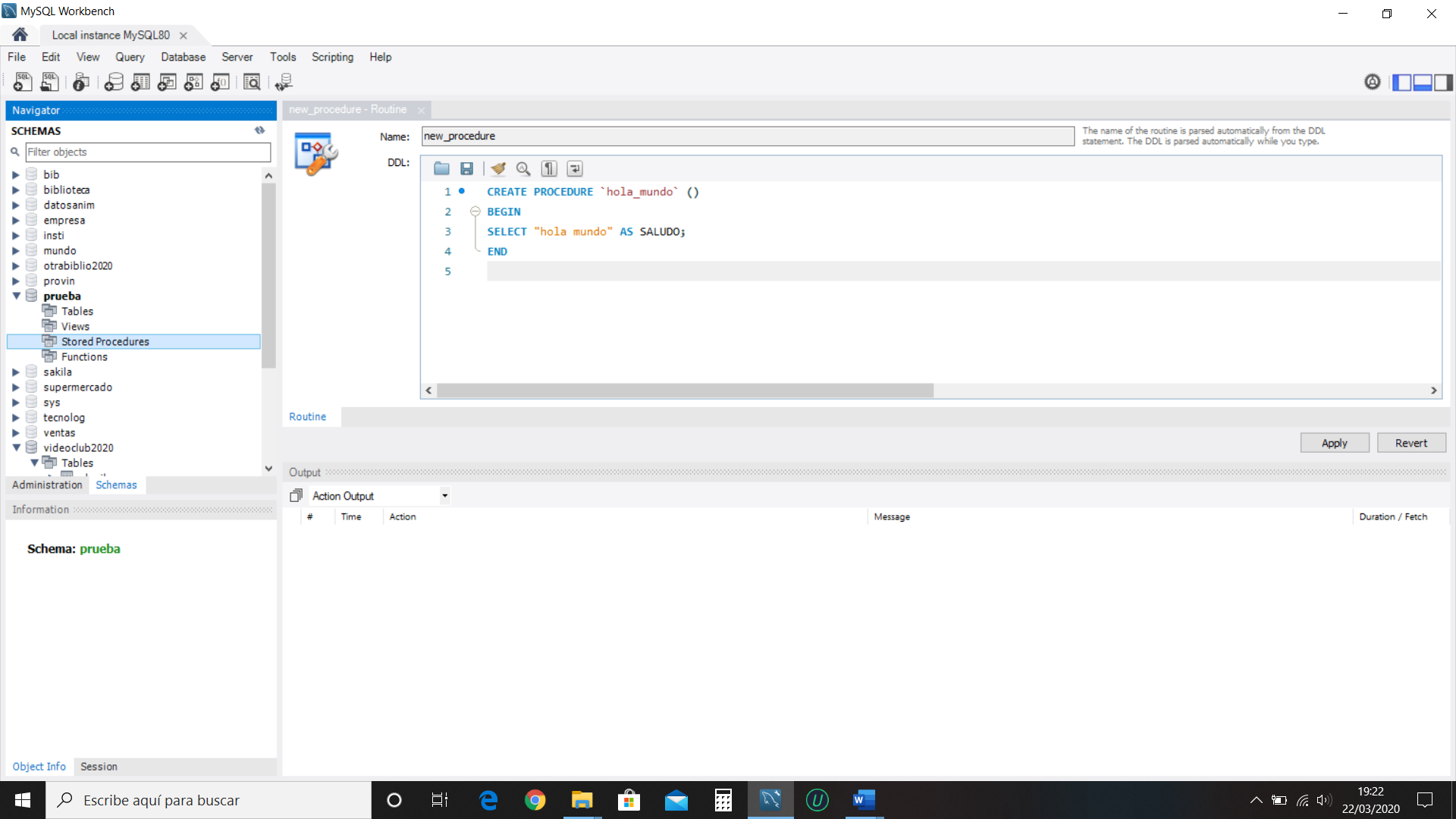
Vamos a crear el siguiente procedimiento:

**CREATE PROCEDURE `hola\_mundo` ()**

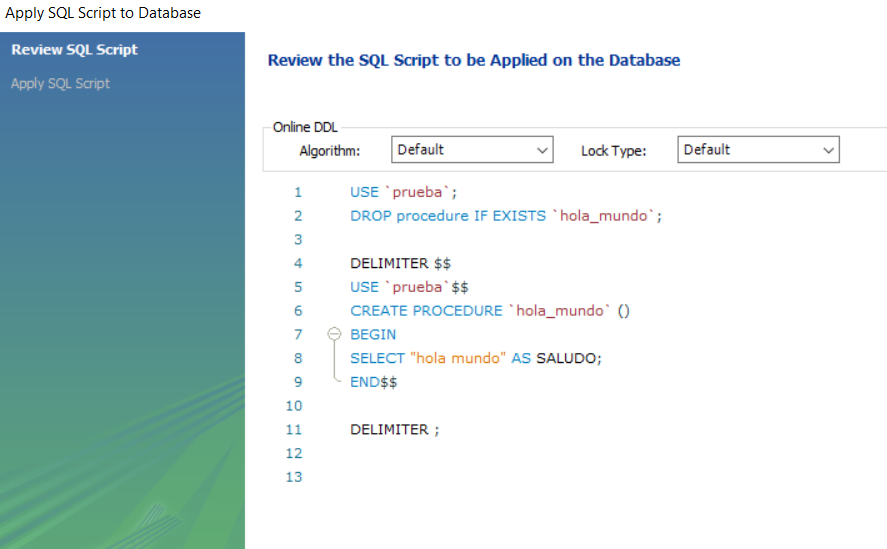
**BEGIN**

**SELECT "hola mundo" AS SALUDO;**

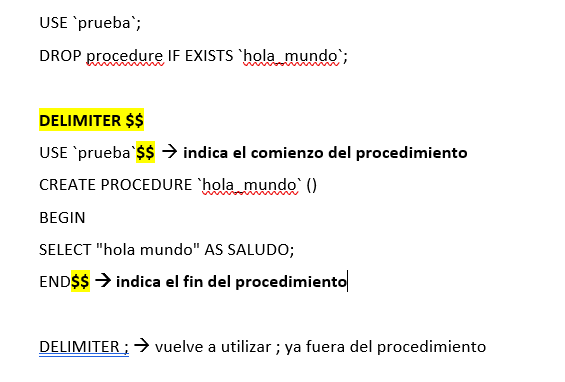
**END**



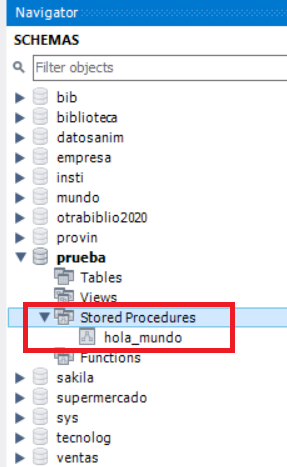
Lo escribo y pulso Apply.



Como sabes, el punto y coma MySQL lo utiliza para terminar una sentencia. Como en los procedimientos y en las funciones hay más de una sentencia que va a terminar con punto y coma, los procedimientos y funciones usar la palabra DELIMITER y otro carácter para indicar comienzo y fin del procedimiento o función. Como ves, el más utilizado es $$ y es el carácter que delimita el procedimiento (a veces también se usa //, pero el más utilizado es $$). Se ocupa MySQL de añadirlo. Simplemente, te lo explico, para que sepas a qué se refiere cuando te aparezca al pulsar Apply.



Y otra vez Apply.

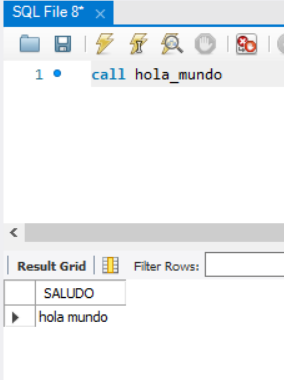


Como podéis ver este procedimiento que hemos creado, no lleva parámetros, puesto que detrás del nombre (hola\_mundo) aparecen los dos paréntesis () pero no hay parámetro.

Para ejecutar este procedimiento, simplemente tenemos que escribir:

**CALL hola\_mundo** y ejecutar con el rayo.

Nos saldría lo siguiente:



**EJEMPLO 2**

Vamos a escribir ahora un procedimiento que devolverá fecha actual y un número aleatorio por pantalla.

Para ello escribe dentro de un nuevo procedimiento lo siguiente:

**CREATE PROCEDURE `fecha`()**

**BEGIN**

**SELECT CURDATE(), RAND();**

**END**

Como puedes ver, podemos incluir dentro de los procedimientos llamadas a funciones propias de MySQL o, como ya veremos más adelante, también funciones creadas por nosotros. La función RAND() genera números aleatorios entre 0 y 1.

Prueba a crearlo y ejecutalo. Incluye a continuación el resultado.

**INVESTIGA (opcional):** Si quisieras obtener números aleatorios >=5 y <10, ¿cómo lo harías?

**EJEMPLO 3**

Voy a poneros ahora un ejemplo de función para que veáis que son muy parecidas a los procedimientos.

Aunque aún no hemos visto las sentencias condicionales, voy a adelantarme un poco porque son muy sencillas. Además, voy a aprovechar para hablaros del tema de la declaración y establecimiento de valores en variables.

Voy a crear una función a la que llamaré SUELDO, que va a recibir un sueldo mensual como parámetro de entrada y va a devolver “ALTO” si sobrepasa los 1500 euros y “BAJO” si es menor.

Para ello voy a escribir lo siguiente:

**CREATE FUNCTION SUELDO (x decimal(6,2))**

**RETURNS varchar(4)**

**BEGIN**

**DECLARE SAL VARCHAR(4);**

**IF X>1500 THEN SET SAL="ALTO";**

**ELSE SET SAL="BAJO";**

**END IF;**

**RETURN SAL;**

**END**

Os explico:

Hemos puesto un parámetro x de tipo decimal(6,2), para poder meter decimales en el sueldo que introduzcamos. Como mucho 4 de parte entera y 2 de decimal.

En returns indicamos el tipo de dato que va a devolver la función. En este caso, como voy a devolver una cadena que contendrá “ALTO” o “BAJO” pues la declaro como VARCHAR(4)

A continuación del BEGIN aparece una **SENTENCIA DE DECLARACIÓN DE VARIABLES.**

**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.

En nuestro caso hemos declarado la variable SAL en la que vamos a guardar las palabras ALTO o BAJO, luego es VARCHAR(4). Fíjate que coincide con el tipo de RETURNS. **Lo que se devuelva en una función tiene que ser del tipo de dato que se indique en el RETURNS.**

Ahora ves que usamos la sentencia IF, que es uno de los tipos de sentencia condicional. Seguro que la has usado en más lenguajes de programación.

IF X>1500 THEN SET SAL="ALTO";

Si el sueldo es mayor de 1500, asignamos a la variable SAL “ALTO”.

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

Después ponemos ELSE SET SAL="BAJO"; 🡪 sino es alto, es bajo, luego es lo que le asigno.

Fíjate que después viene un END IF;

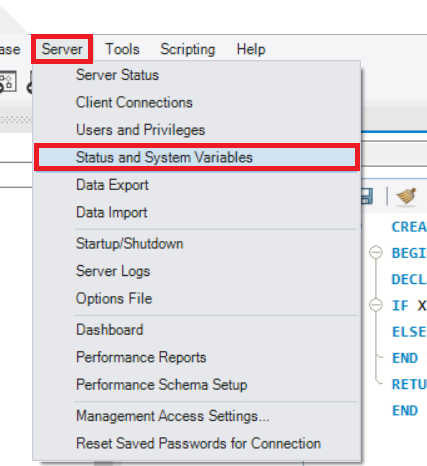
Todos los IF que se abran, hay que cerrarlos con END IF.

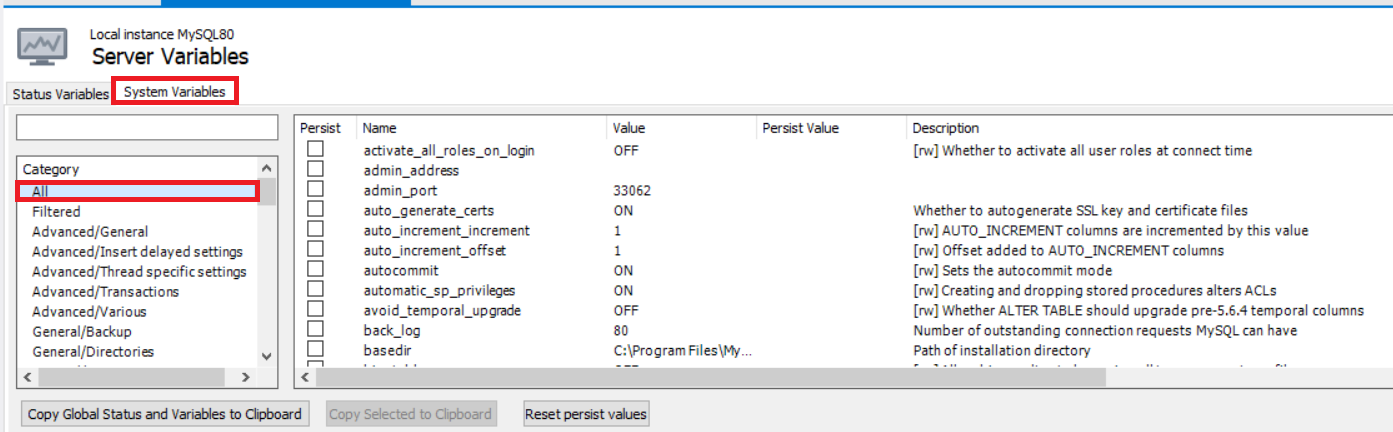
Y, por último, aparece RETURN SAL. Esta sentencia devuelve el resultado de la función.

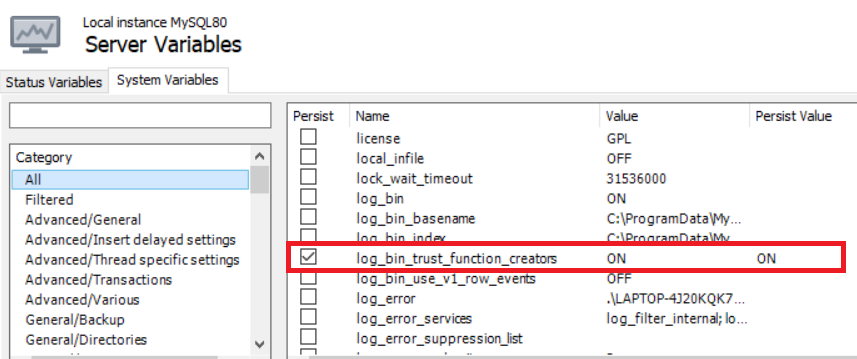
Antes de ejecutarla tenemos que modificar una cosa para que MYSQL nos permita ejecutar las funciones correctamente.

**OBLIGATORIO QUE LO HAGAS, SINO NO TE VAN A FUNCIONAR LAS FUNCIONES CUANDO LAS EJECUTES.**

Tienes que entrar en

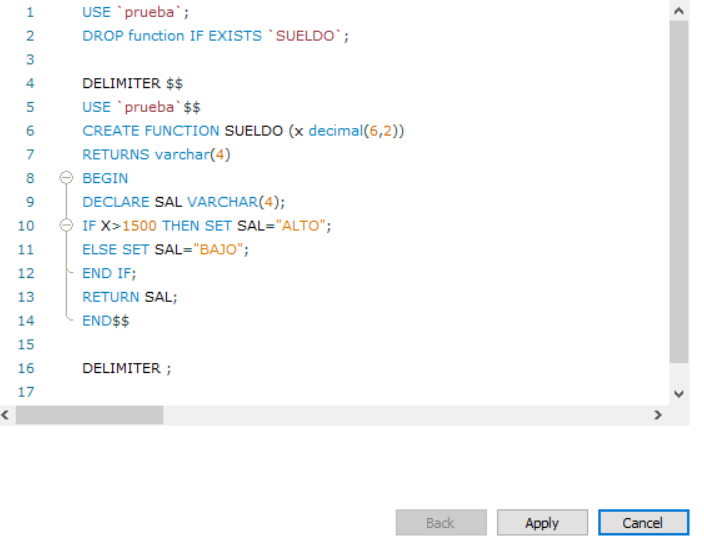




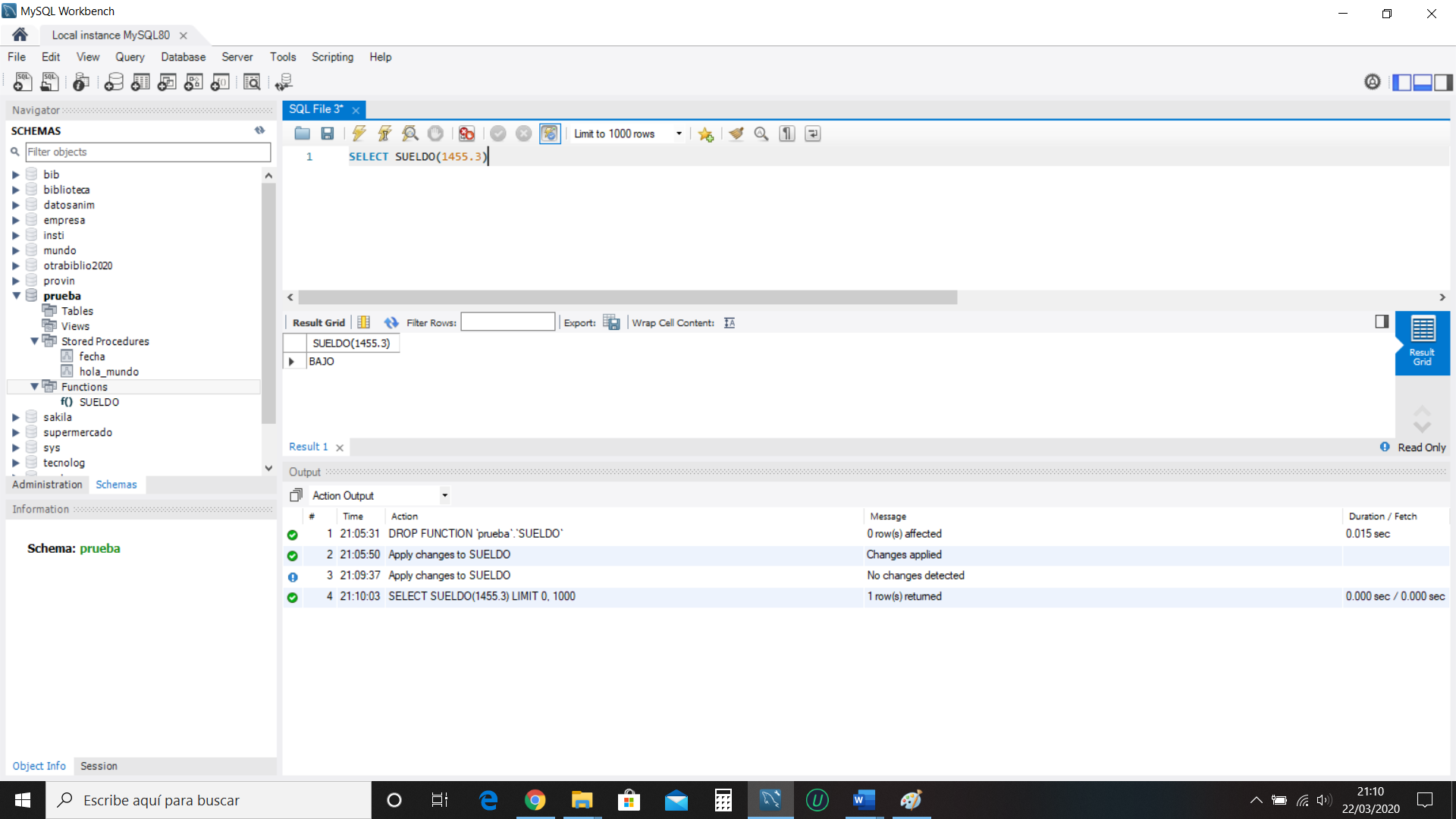


Busco esa línea y la marco, para que quede ON SIEMPRE.

Creo la función:



Por último, ejecuta la función para ver lo que obtienes. **Recuerda que las funciones se llaman con SELECT.**



**EJEMPLO 4**

Vamos a crear una función que recibirá como parámetro de entrada una letra y me devolverá azul si meto A, rojo si meto R y verde si meto V. En el resto de los casos, podrá NADA. Ya sé que este ejercicio no tiene mucho sentido para hacer nada específico, pero es para que os vayáis familiarizando con la estructura de las funciones y procedimientos.

**CREATE FUNCTION `colores` (a CHAR(1))**

**RETURNS VARCHAR(10)**

**BEGIN**

**DECLARE color VARCHAR(10);**

**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";**

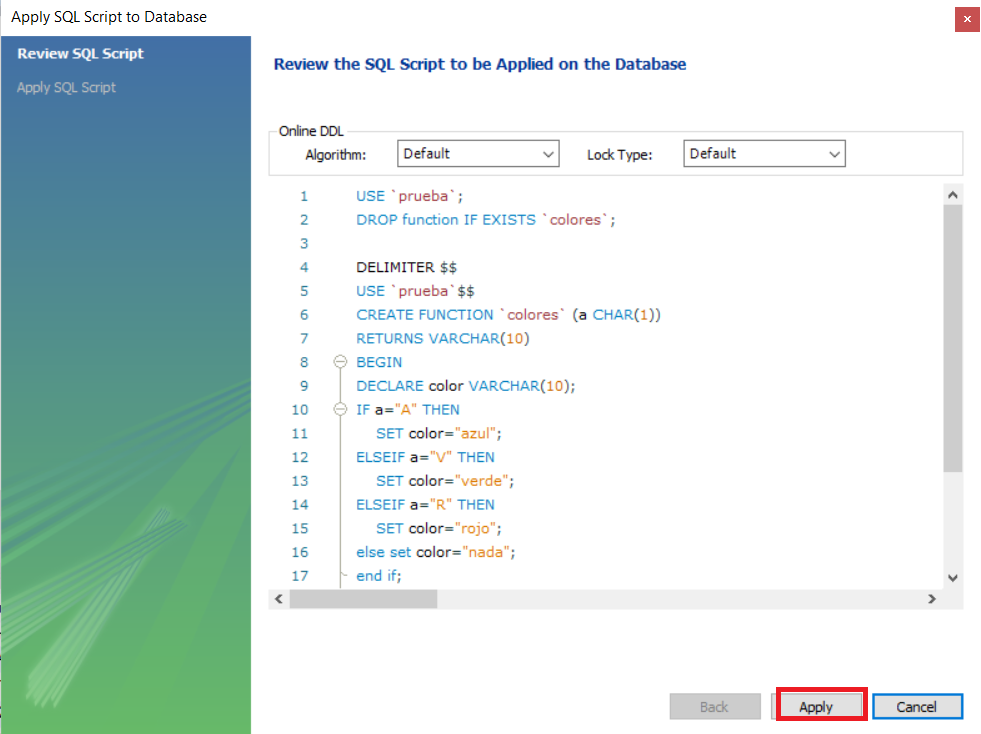
**end if;**

**RETURN color;**

**END**

En este caso, como el resultado que vamos a devolver va a ser una cadena de caracteres, tanto RETURNS como la variable color en la que vamos a devolver dicho resultado vamos a definirlos como VARCHAR.

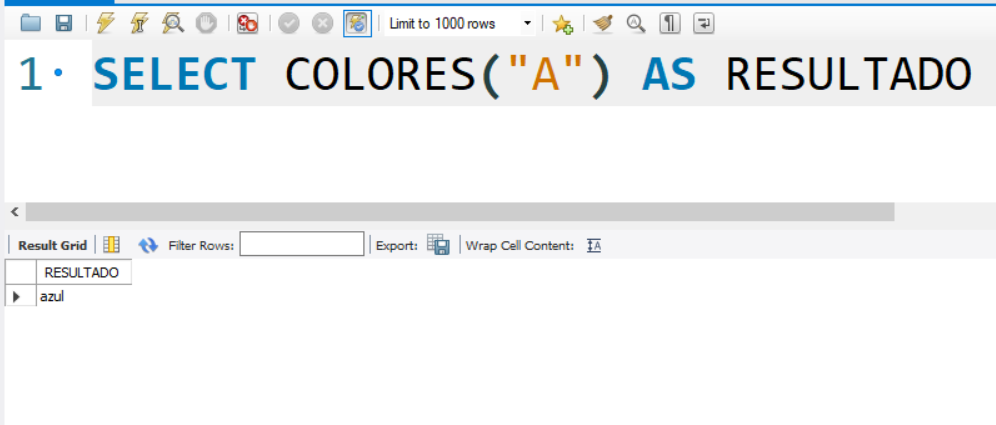
Como podéis observar, Podemos ir anidando los IF para poner varias condiciones en la estructura. Se puede hacer de otra manera, si son muchos. Ya os paso más adelante estas estructuras condicionales.



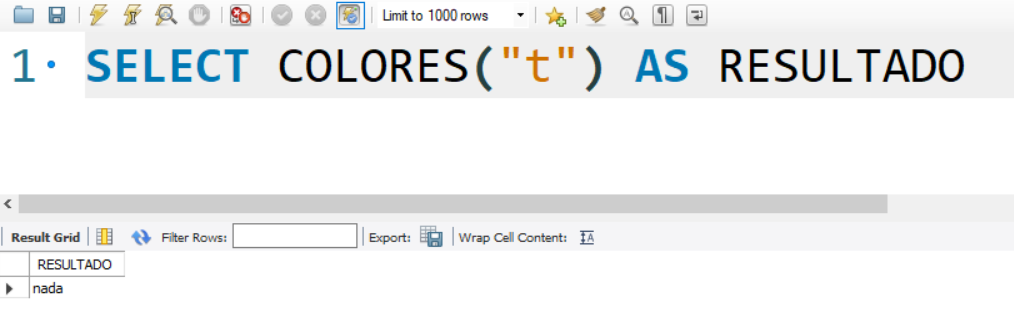
Una vez creada, vamos a ejecutarla. Recuerda que se ejecutan con SELECT. En este caso, como el parámetro es una letra, tiene que ir entre comillas. Lo de AS RESULTADO es para que salga eso como título de la columna

SELECT COLORES("A") AS RESULTADO

Obtendremos lo siguiente:



Si meto cualquier letra distinta de a, r o v vemos que la función me devuelve “nada”.



**EJEMPLO 5**

Vamos a crear ahora una función que recibirá como parámetro de entrada un número y devolverá 0 si el número es par y 1 si es impar. Para ello vamos a utilizar una función propia de MySQL que es MOD. Esta función devuelve el resto de la división entre dos números. En nuestro caso, para saber si un número es par o impar, lo divido entre 2. Si el resto es 0, el número es par y sino, es impar.

**CREATE FUNCTION `paroimpar` (num int)**

**RETURNS INT**

**BEGIN**

**declare valor int;**

**IF MOD(num,2)=0 THEN SET valor=0;**

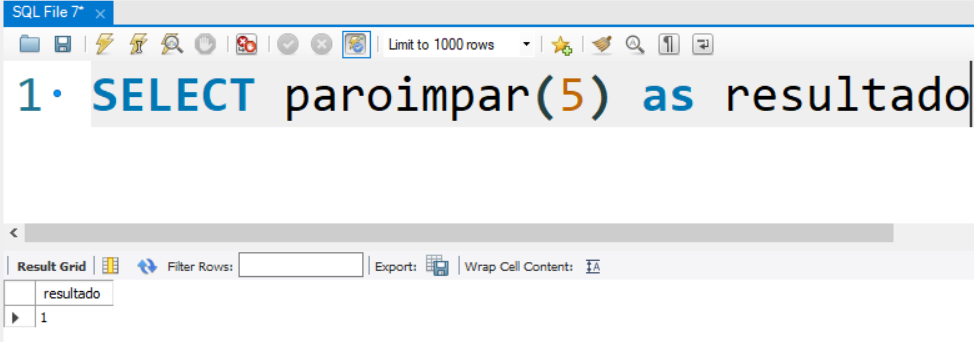
**else SET valor=1;**

**END IF;**

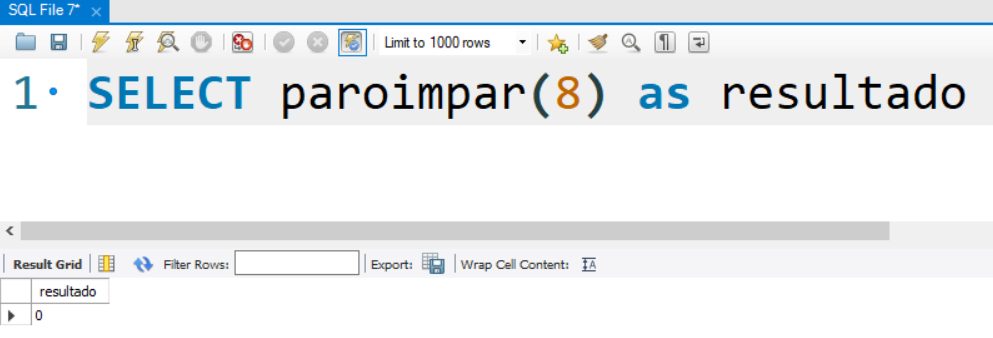
**RETURN valor;**

**END**

Una vez creada, vamos a ejecutarla.

****

Como el 5 es impar, me devuelve 1.



En este caso, devuelve 0 porque el 8 es par.

**EJERCICIO1:**

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

**EJEMPLO 6**

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

**CREATE PROCEDURE `ESPAR` (num int)**

**BEGIN**

**IF paroimpar(num)=1 THEN**

**SELECT CONCAT(num," es impar") AS RESULTADO;**

**ELSE**

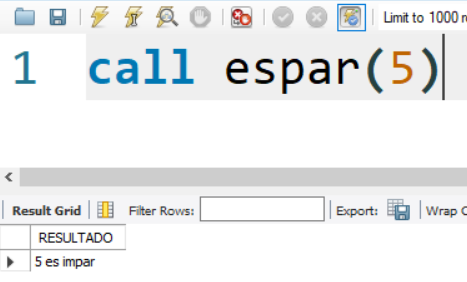
**SELECT CONCAT(num," es par") AS RESULTADO;**

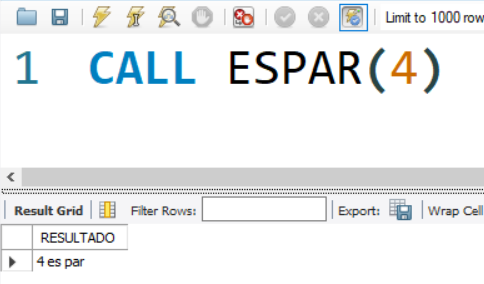
**END IF;**

**END**

En este caso, estamos creando un procedimiento que está utilizando la función creada en el ejemplo anterior para ver si el número introducido como parámetro del procedimiento es par o impar.

Vamos a ejecutarlo. Recuerda que los procedimientos se ejecutan con CALL.





Como puedes comprobar, puedo llamar a las funciones desde un procedimiento para utilizarlas cuando quiera.

Además, ya veremos más adelante que todas las funciones que nosotros creemos podremos utilizarlas con las tablas de los esquemas igual que cualquier otra función de las predefinidas de MYSQL.

**EJEMPLO 7**

Crea dentro del esquema prueba una tabla llamada t con dos campos de tipo INT que se llamen a y b. No hace falta que pongas claves. A continuación, crea la siguiente rutina:

**CREATE PROCEDURE proc1 (parametro1 INT)**

**BEGIN**

**DECLARE variable1, variable2 INT;**

**IF parametro1=17 THEN**

**SET variable1=parametro1;**

**SET variable2=10;**

**ELSE**

**SET variable1=10;**

**SET variable2=30;**

**END IF;**

**INSERT INTO t 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.

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.

El ejemplo tiene sólo valor didáctico, para que veáis que puedo usar procedimientos que me permitan insertar valores en las tablas. Pretendo que empecéis a ver cómo puede integrarse esto con los esquemas y las tablas.

Vamos ahora a ejecutar el procedimiento.

Para ello, lo llamaré con CALL y un parámetro de tipo entero.



Vamos a ver ahora el contenido de la tabla. Tiene que tener 17 y 10, puesto que el procedimiento decía:

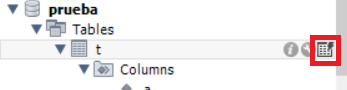
**IF parametro1=17 THEN**

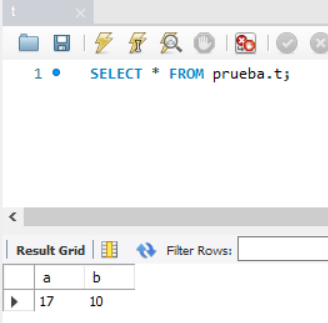
**SET variable1=parametro1;**

**SET variable2=10;**

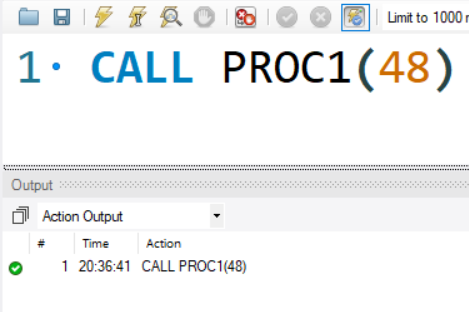
El parámetro 1 es 17 🡪 SET variable1=17

SET variable2=10





Si ahora lo ejecutamos con 48

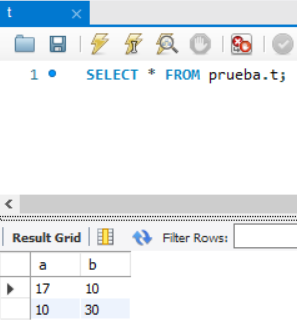


Vemos que en la tabla ahora se han añadido 10 y 30 porque el procedimiento decía que si el parámetro de entrada era distinto de 17, hacíamos lo siguiente:

**ELSE**

**SET variable1=10;**

**SET variable2=30;**



**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 @) (esto, de momento, no os lo voy a explicar). Lo vemos en el siguiente ejemplo:

**CREATE PROCEDURE proc5 ()**

**BEGIN**

**DECLARE x1 char(6) DEFAULT "fuera";**

**BEGIN**

**DECLARE x1 CHAR(6) DEFAULT "dentro";**

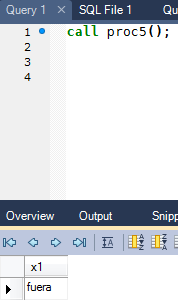
**SELECT x1;**

**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” y no se haya modificado.



**EJERCICIOS:**

1. **Sobre el esquema de pruebas test, crea un procedimiento que muestre el año actual. Este procedimiento no llevará parámetro de entrada.**
2. **Sobre el esquema test, crea una función a la que pasaremos como parámetro un dato de tipo fecha y la función me devolverá el número de días que han pasado hasta hoy.**
3. **Crea una función 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.**
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**