3.- AUTOMATIZACIÓN DE TAREAS: CONSTRUCCIÓN DE GUIONES DE ADMINISTRACIÓN.

3.1.- Procedimientos y funciones almacenadas:

Para optimizar su trabajo al administrador de bases de datos debe contar con herramientas para automatizar sus tareas dada la gran y diversa cantidad de trabajos que debe realizar. Hoy en día casi todos los sistemas gestores, comerciales y libres, cuentan con herramientas para la automatización que en la mayoría de los casos consisten en el uso de rutinas (procedimientos y funciones) almacenados, disparadores o triggers y eventos además de la posibilidad de usar APIs de distintos lenguajes de programación como perl, php o python.

Las rutinas (procedimientos almacenados y funciones) son un conjunto de comandos SQL que pueden almacenarse en el servidor. Una vez que se hace, los clientes no necesitan relanzar los comandos individuales pero pueden en su lugar referirse al procedimiento almacenado como un único comando.

Algunas situaciones en que los procedimientos almacenados pueden ser particularmente útiles:

- Cuando múltiples aplicaciones cliente se escriben en distintos lenguajes o funcionan en distintas plataformas, pero necesitan realizar la misma operación en la base de datos.
- Cuando la seguridad es muy importante. Los bancos, por ejemplo, usan procedimientos almacenados para todas las operaciones comunes. Esto proporciona un entorno seguro y consistente, y los procedimientos pueden asegurar que cada operación se loguea apropiadamente. En tal entorno, las aplicaciones y los usuarios no obtendrían ningún acceso directo a las tablas de la BD, sólo pueden ejecutar algunos procedimientos almacenados.

Los procedimientos almacenados pueden mejorar el rendimiento ya que se necesita enviar menos información entre el servidor y el cliente. El intercambio que hay es que aumenta la carga del servidor de la base de datos ya que la mayoría del trabajo se realiza en la parte del servidor y no en el cliente.

3.2.- Procedimientos almacenados y las tablas de permisos:

Los procedimientos almacenados requieren la tabla proc en la base de datos mysql. Desde MySQL 5.0.3, el sistema de permisos se ha modificado para tener en cuenta los procedimientos almacenados como sigue:

- El permiso CREATE ROUTINE se necesita para crear procedimientos almacenados.
- El permiso ALTER ROUTINE se necesita para alterar o borrar procedimientos almacenados. Este permiso se da automáticamente al creador de una rutina.
- El permiso EXECUTE se requiere para ejecutar procedimientos almacenados. Sin embargo, este permiso se da automáticamente al creador de la rutina. También, la característica SQL SECURITY por defecto para una rutina es DEFINER, lo que permite a los usuarios que tienen acceso a la base de datos ejecutar la rutina asociada.

3.3. Sintaxis de procedimientos almacenados:

Los procedimientos almacenados y rutinas se crean con comandos CREATE PROCEDURE y CREATE FUNCTION. Una rutina es un procedimiento o una función. Un procedimiento se invoca usando un comando CALL , y sólo puede pasar valores usando variables de salida. Una función puede llamarse desde dentro de un comando como cualquier otra función (esto es, invocando el nombre de la función), y puede retornar un valor escalar. Las rutinas almacenadas pueden llamar otras rutinas almacenadas.

Desde MySQL 5.0.1, los procedimientos almacenados o funciones se asocian con una base de datos. Esto tiene varias implicaciones:

- Cuando se invoca la rutina, se realiza implícitamente USE *db_name* (y se deshace cuando acaba la rutina). Los comandos USE dentro de procedimientos almacenados no se permiten.
- Puede calificar los nombres de rutina con el nombre de la base de datos. Esto puede usarse para referirse a una rutina que no esté en la base de datos actual. Por ejemplo, para invocar procedimientos almacenados **p** o funciones **f** esto se asocia con la base de datos test , puede decir CALL test.p() o test.f().
- Cuando se borra una base de datos, todos los procedimientos almacenados asociados con ella también se borran.

CREATE PROCEDURE y CREATE FUNCTION:

```
CREATE PROCEDURE sp_name ([parameter[,...]])
    [characteristic ...] routine_body

CREATE FUNCTION sp_name ([parameter[,...]])
    RETURNS type
    [characteristic ...] routine_body
```

Donde:
sp_name: 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, salida o entrada/salida:
[IN | OUT | INOUT] *param_name type*

type: cualquier tipo de datos válido de *MySQL*.

routine_body: es el cuerpo de la rutina formado generalmente por sentencias SQL. En caso de haber más de una debe ir dentro de un bloque delimitado por sentencias BEGIN y END.

```
characteristic:
    LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'
```

Deterministic: indica si es determinista o no, es decir si siempre produce el mismo resultado.

Contains SQL/noSQL: especifica si contiene sentencias SQL o no.

```
Modifies SQL data/Reads SQL data: indica si las sentencias modifican o no los
datos.

SQL security: determina si deje ejecutarse con permisos del creador (definer) o
del que lo invoca (invoker).
```

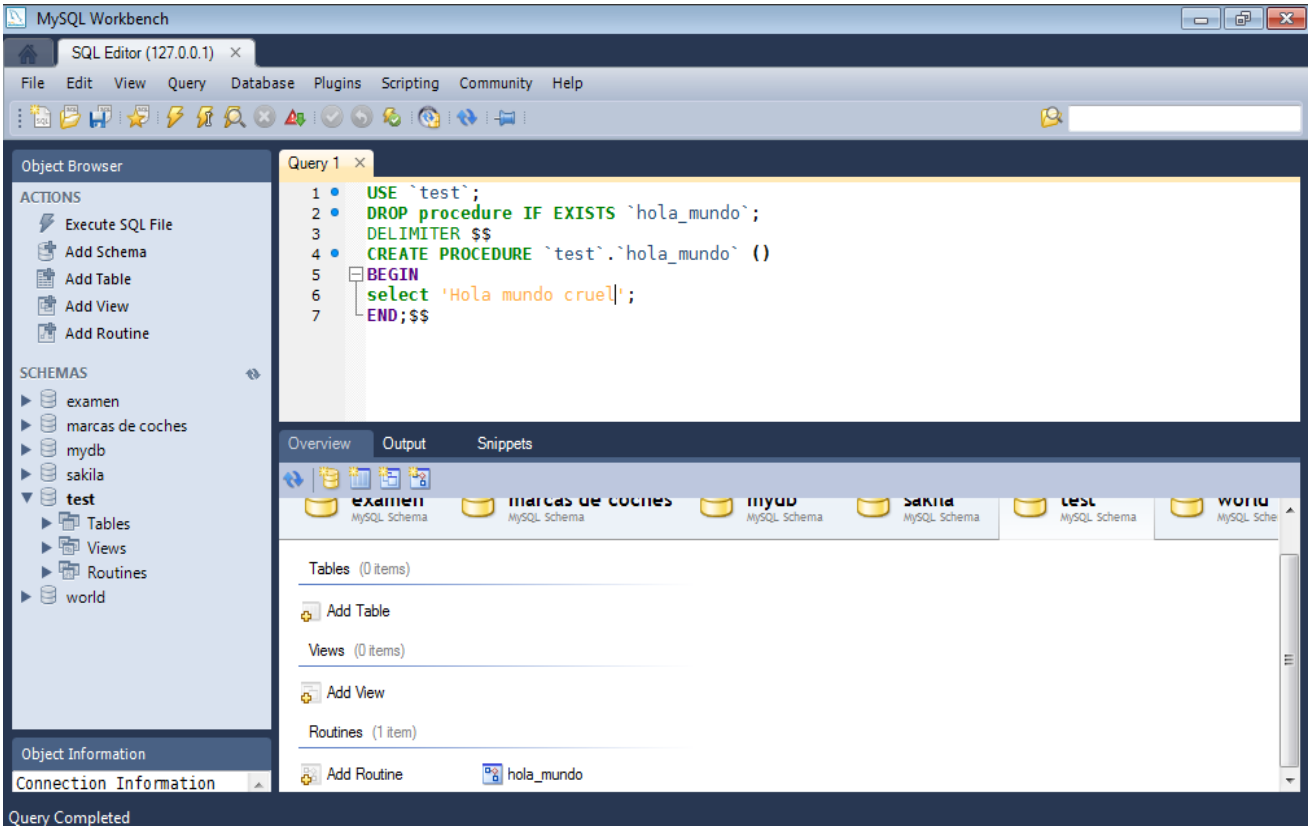
Por defecto, la rutina se asocia con la base de datos actual. Para asociar la rutina explícitamente con una base de datos, especifique el nombre como db_name.sp_name al crearlo.

De forma resumida una rutina obedece al siguiente esquema:

```
nombre (parámetros) + modificadores
begin
declaración (DECLARE) y establecimiento de variables (SET)
proceso de datos (instrucciones sql / instrucciones de control)
end
```

Si el nombre de rutina es el mismo que el nombre de una función de SQL, necesita usar un espacio entre el nombre y el siguiente paréntesis al definir la rutina, o hay un error de sintaxis. Esto también es cierto cuando invoca la rutina posteriormente.

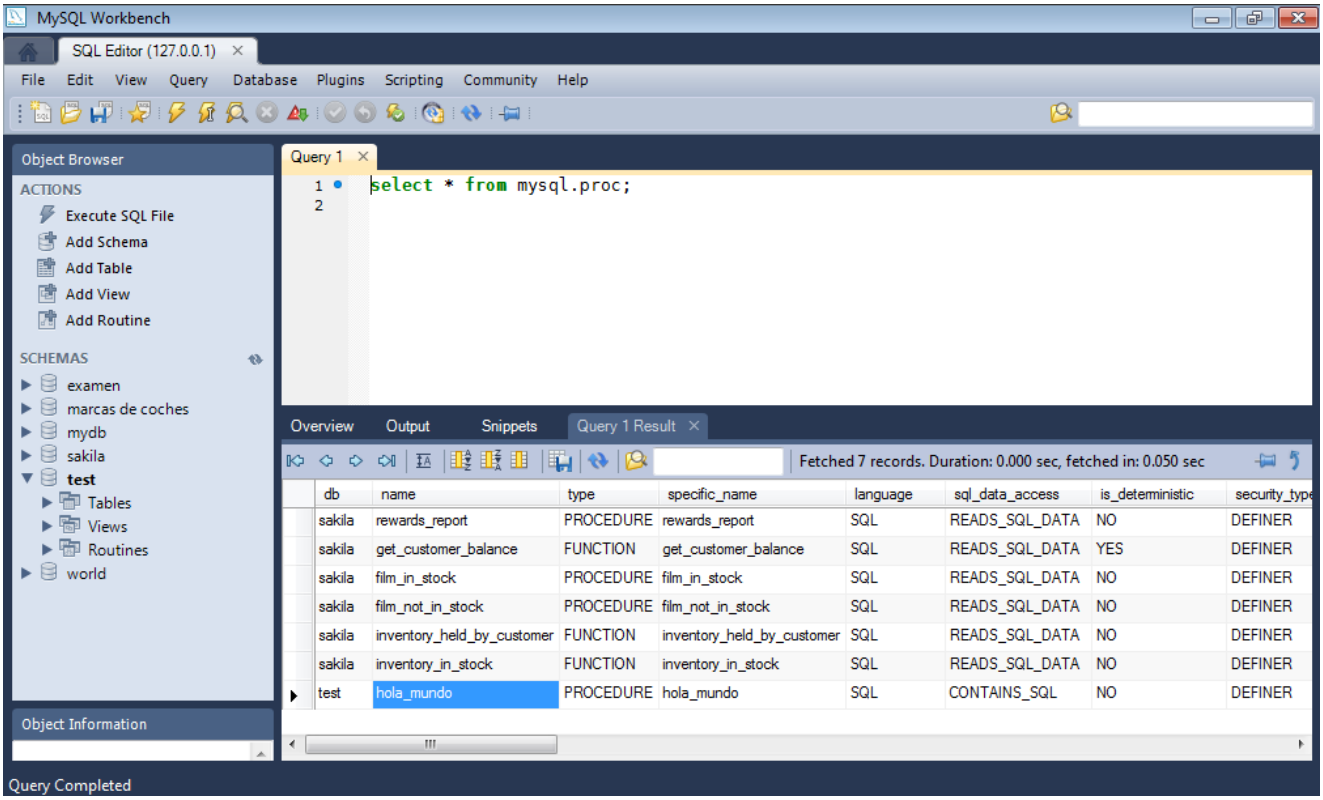
Ejemplo:



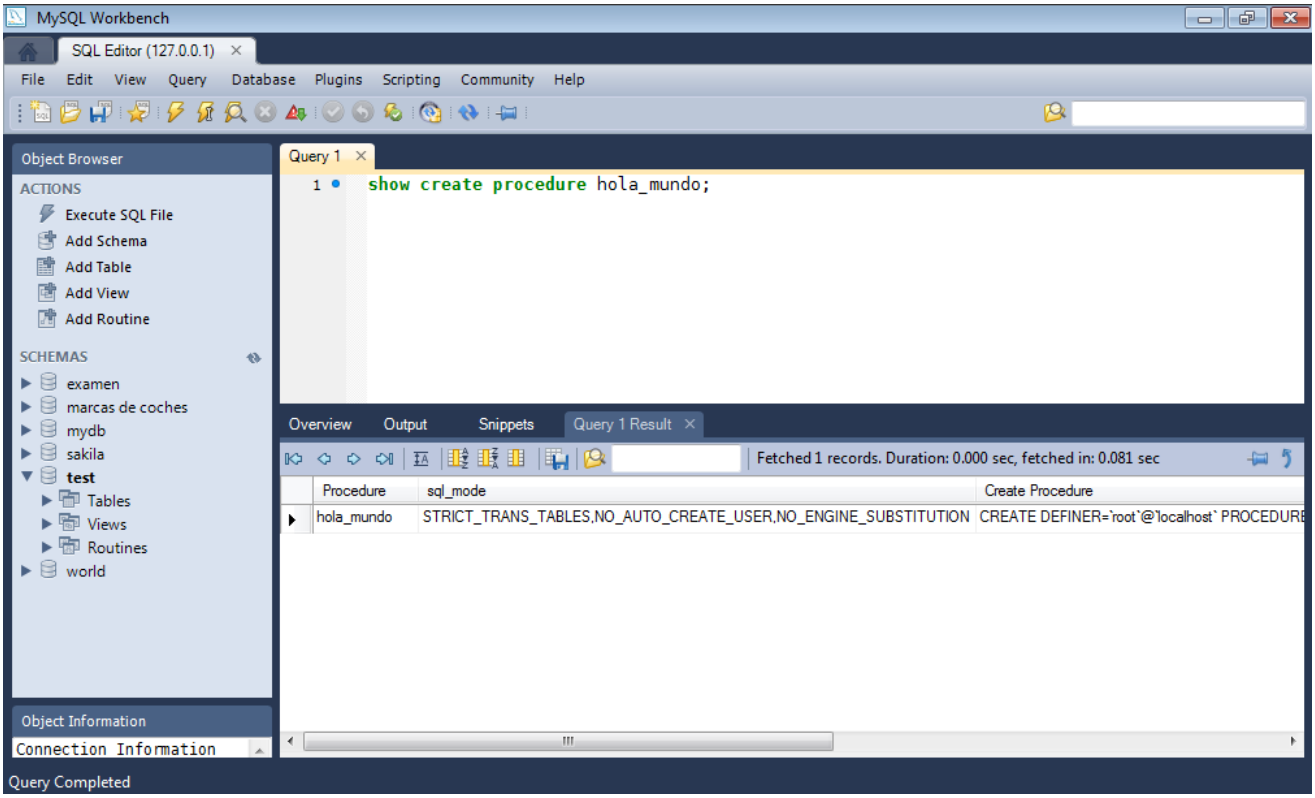
La palabra clave DELIMITER indica el carácter de comienzo y fin del procedimiento. Lo normal sería terminarla con un ; pero dado que los ; los usaremos en las sentencias SQL dentro del procedimiento es conveniente usar otro carácter (normalmente $$ o //). En la última línea se produce el fin del procedimiento (END) seguido del doble carácter con el que comenzamos el procedimiento, en este caso el &&, indicando que ya hemos terminado.

Ahora comprobamos que se ha creado:

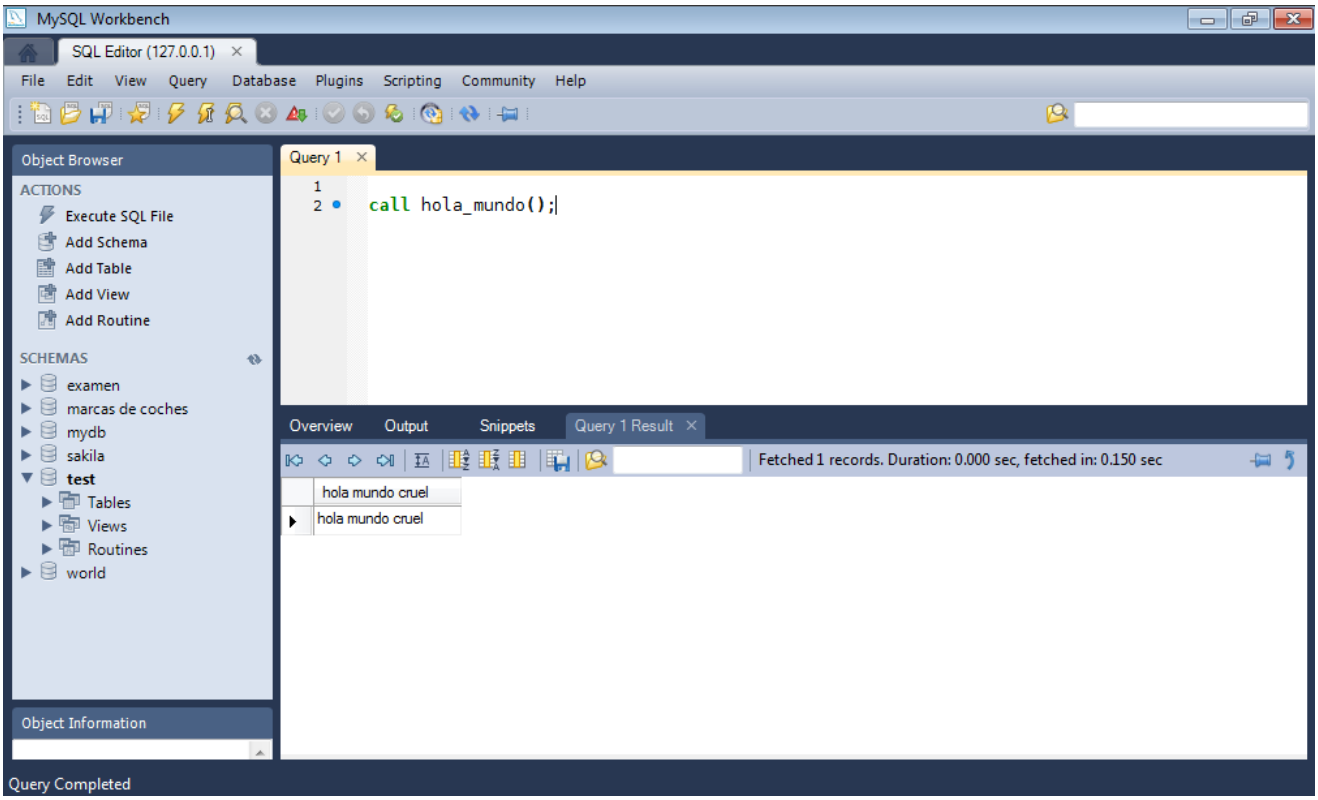
- En la tabla proc de la base de datos mysql.



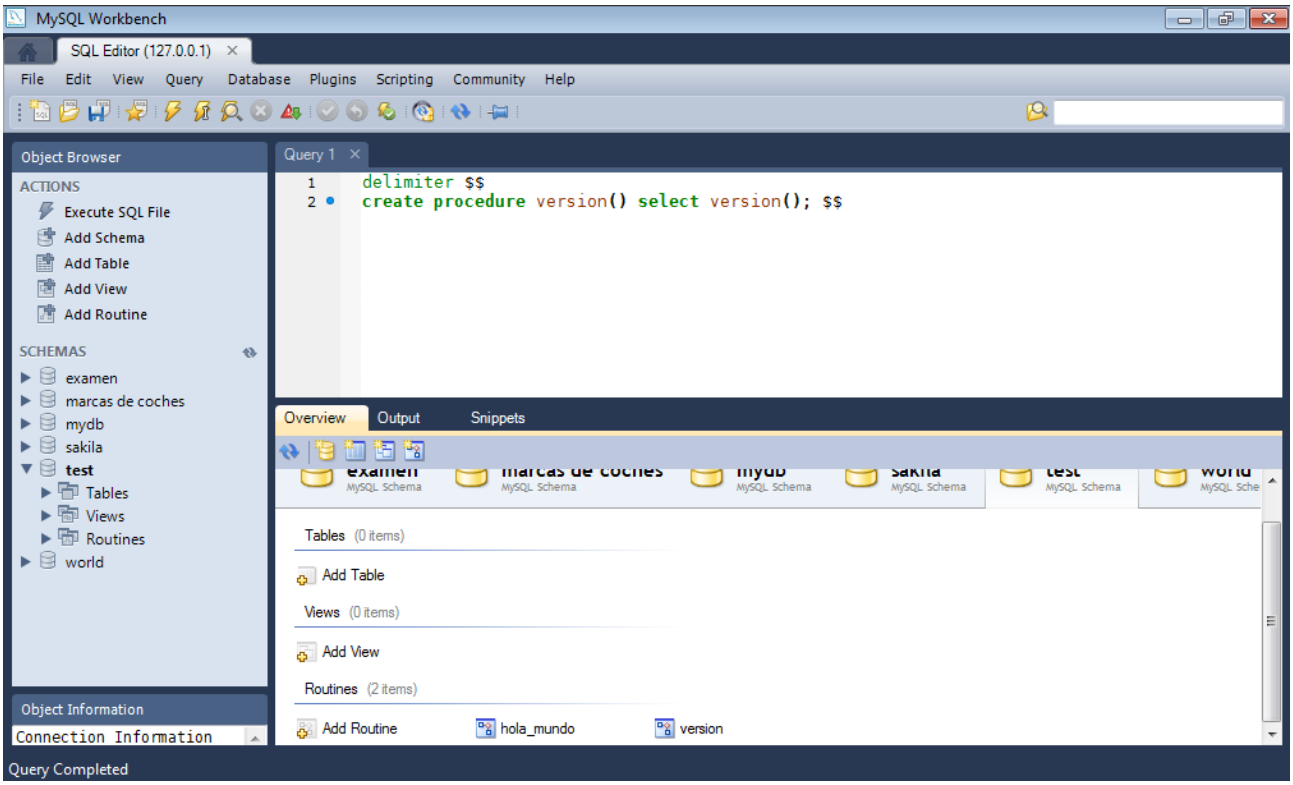
- O utilizando la sentencia show.



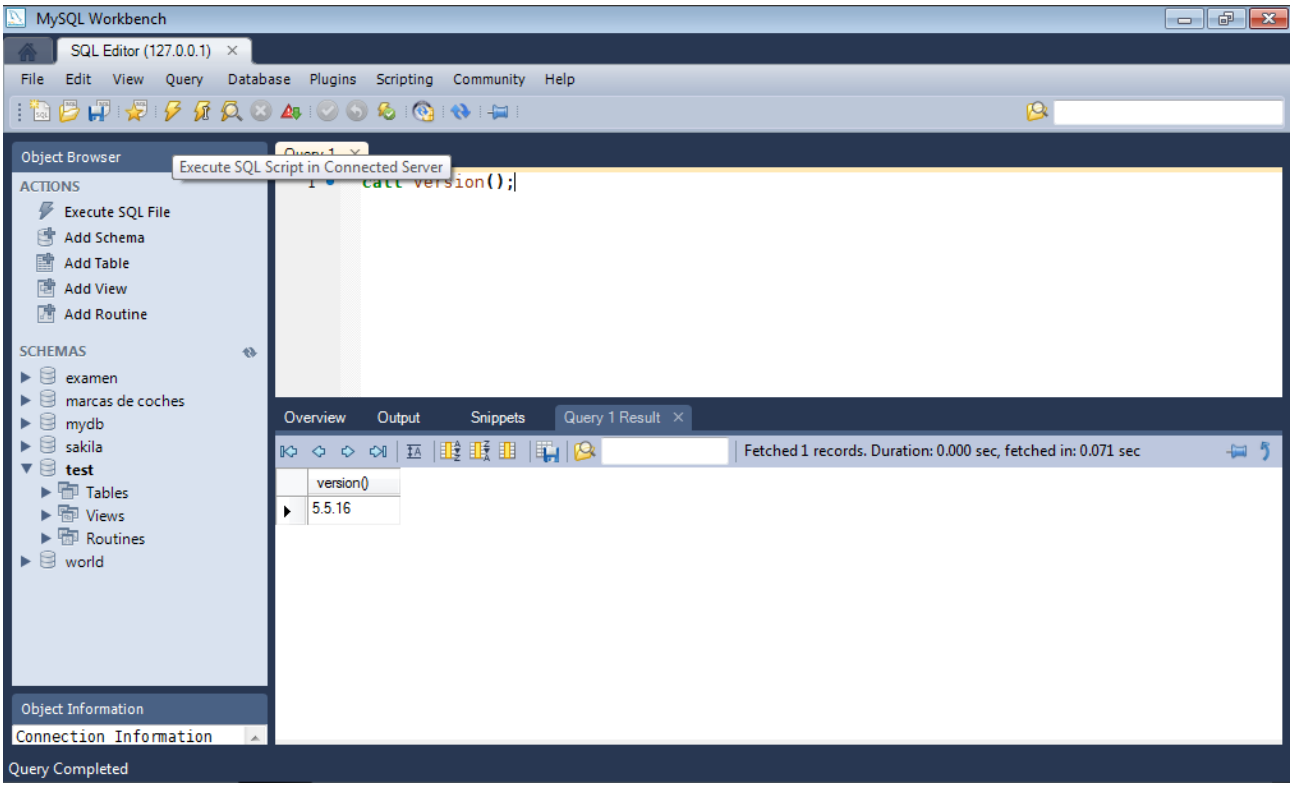
Para ejecutar el procemiento utilizamos la sentencia CALL:



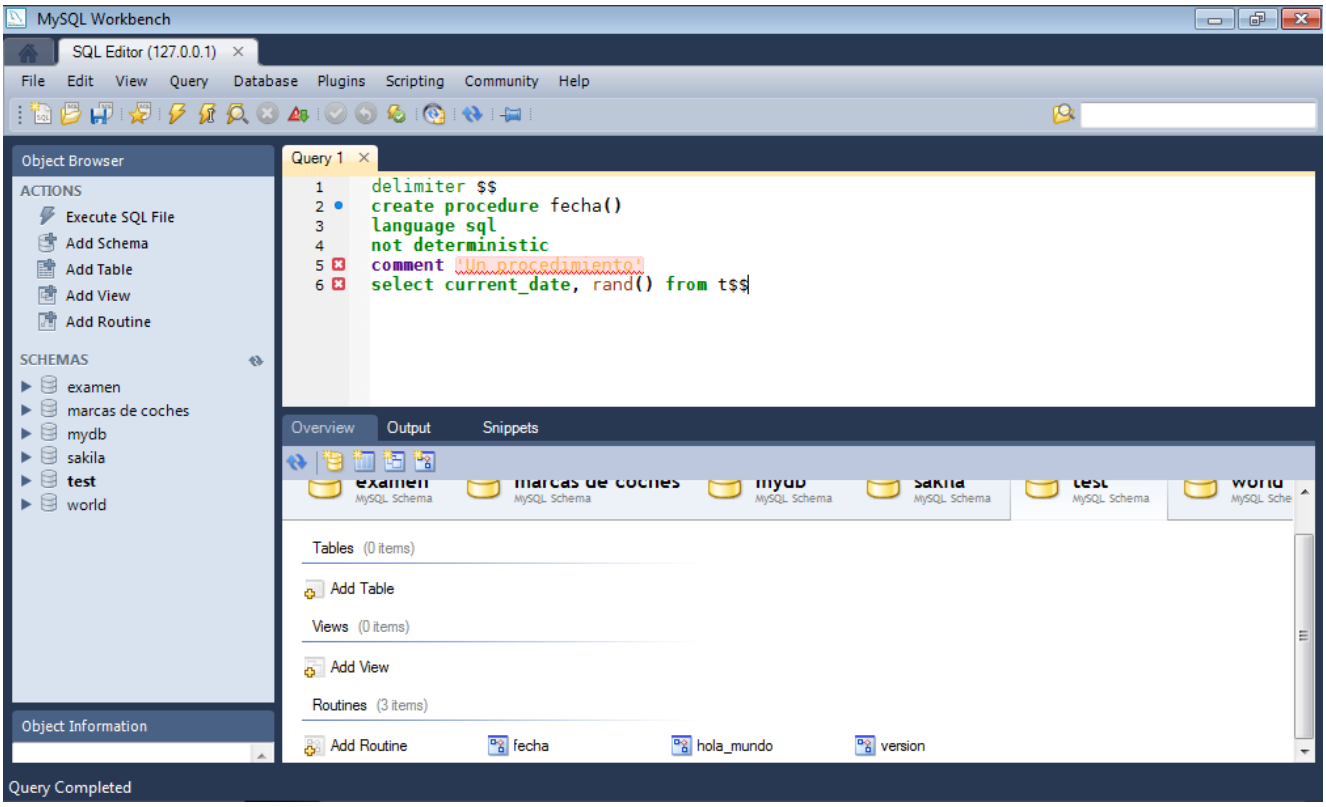
Otro ejemplo, donde podemos ver que si sólo usamos una sentencia sql no es necesario utilizar BEGIN y END.



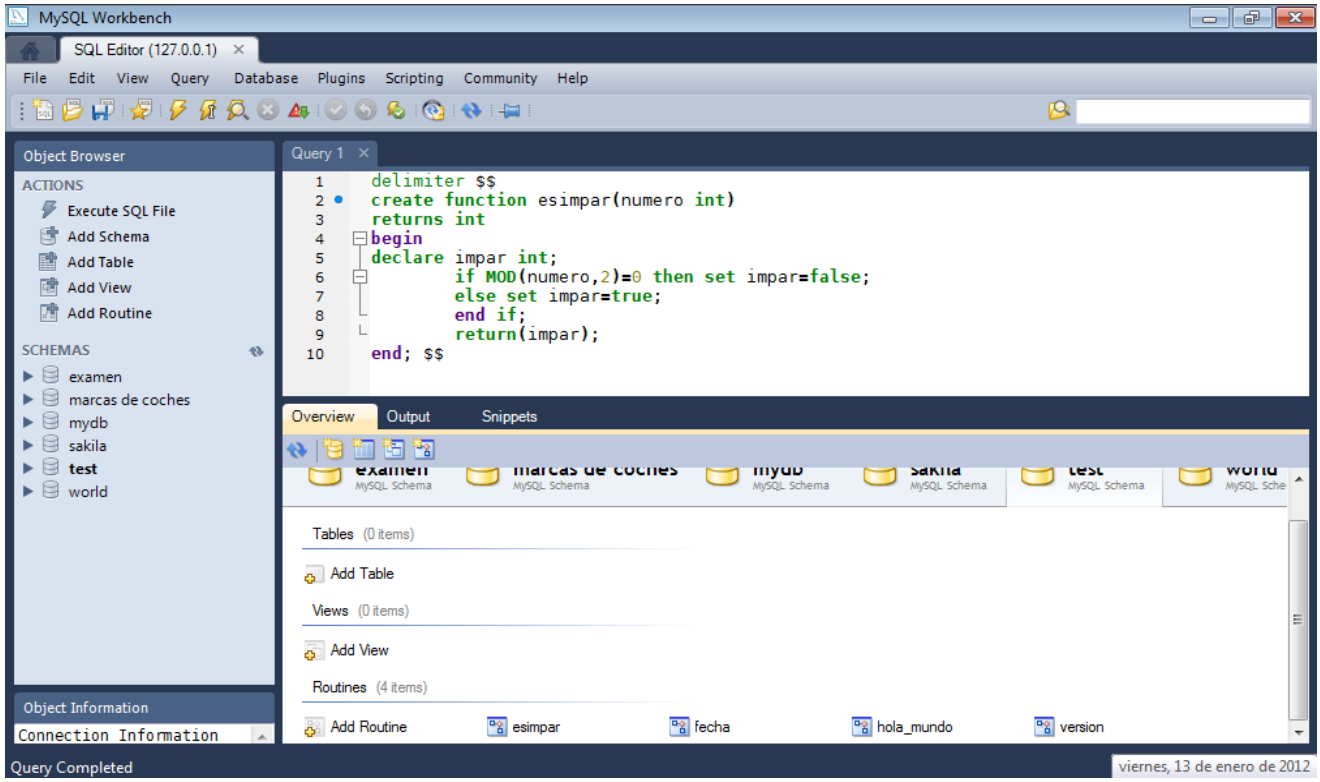
La llamamos y vemos el resultado:



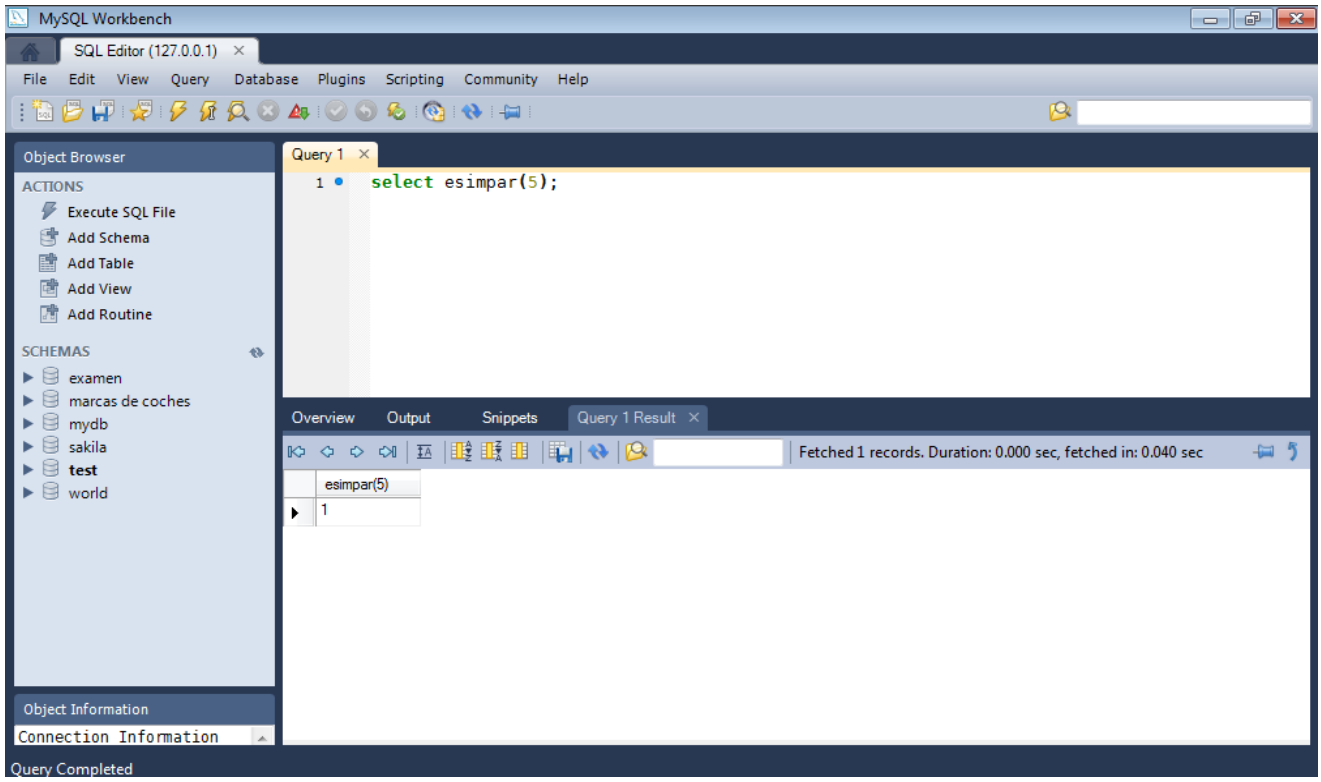
Ahora veremos un ejemplo, donde se usa nuevas líneas como LANGUAGE para indicar el lenguaje, NOT DETERMINISTIC que indica que el algoritmo no siempre produce el mismo resultado cada vez que se llama y COMMENT para documentar el procedimiento con comentarios.



La cláusula RETURNS puede especificarse sólo con **FUNCTION**, donde es obligatorio. Se usa para indicar el tipo de retorno de la función, y el cuerpo de la función debe contener un comando RETURN value. La lista de parámetros entre paréntesis debe estar siempre presente. Si no hay parámetros, se debe usar una lista de parámetros vacía (). Ejemplo de función:



DECLARE es para crear variables con su nombre y tipo y SET permite asignar valores a las variables usando el operador de igualdad.En este ejemplo la función impar devuelve un 0 o un 1 si la variable pasada como parámetro es o no par. Dicho valor se asigna a una variable de sesión que para mostrar necesitamos un SELECT.

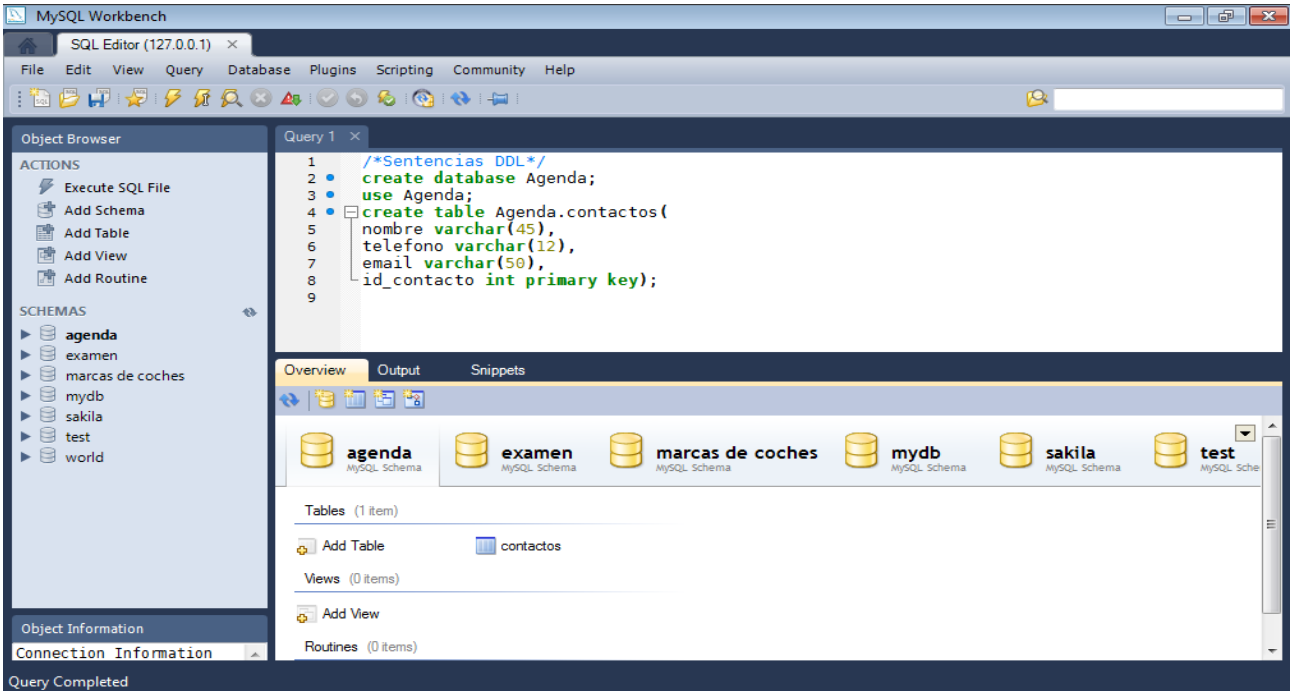


Ejercicio:

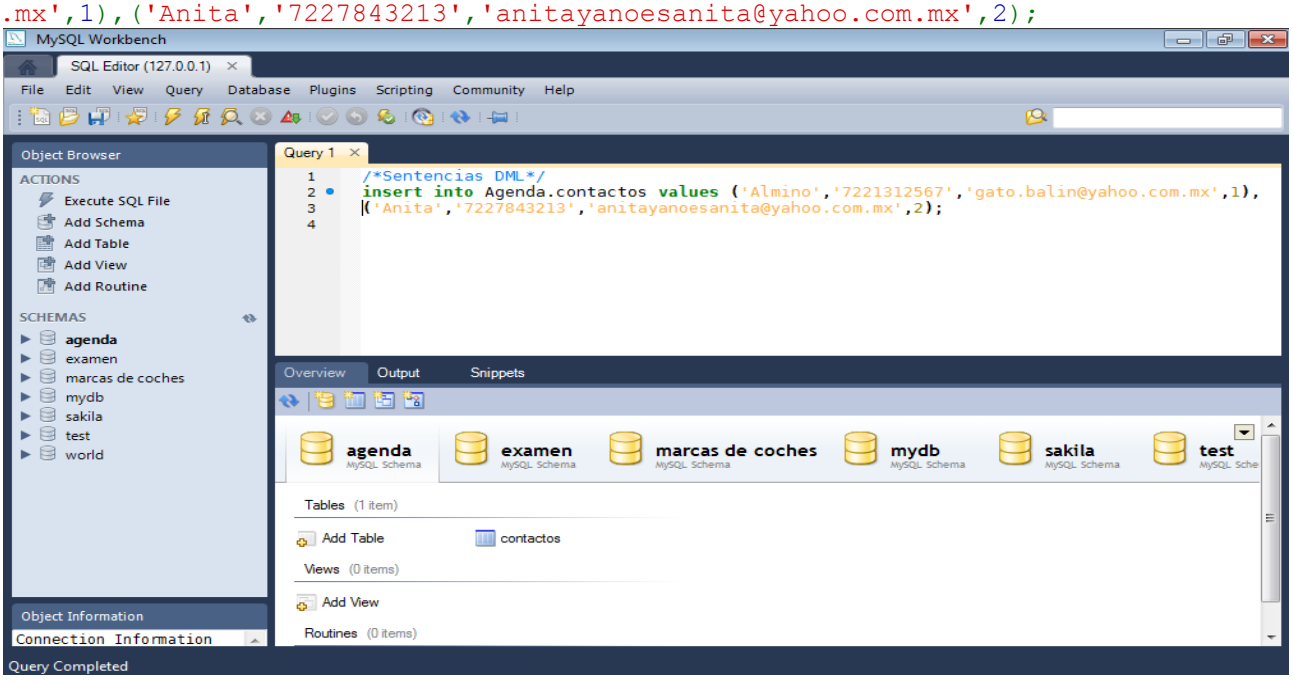
En el Workbench hacer lo siguiente:

1.- Crear la siguiente base de datos llamada ***agenda***, dentro de ésta crear la tabla ***contactos*** , insertar dos registros , luego se actualiza uno, con el siguiente código:

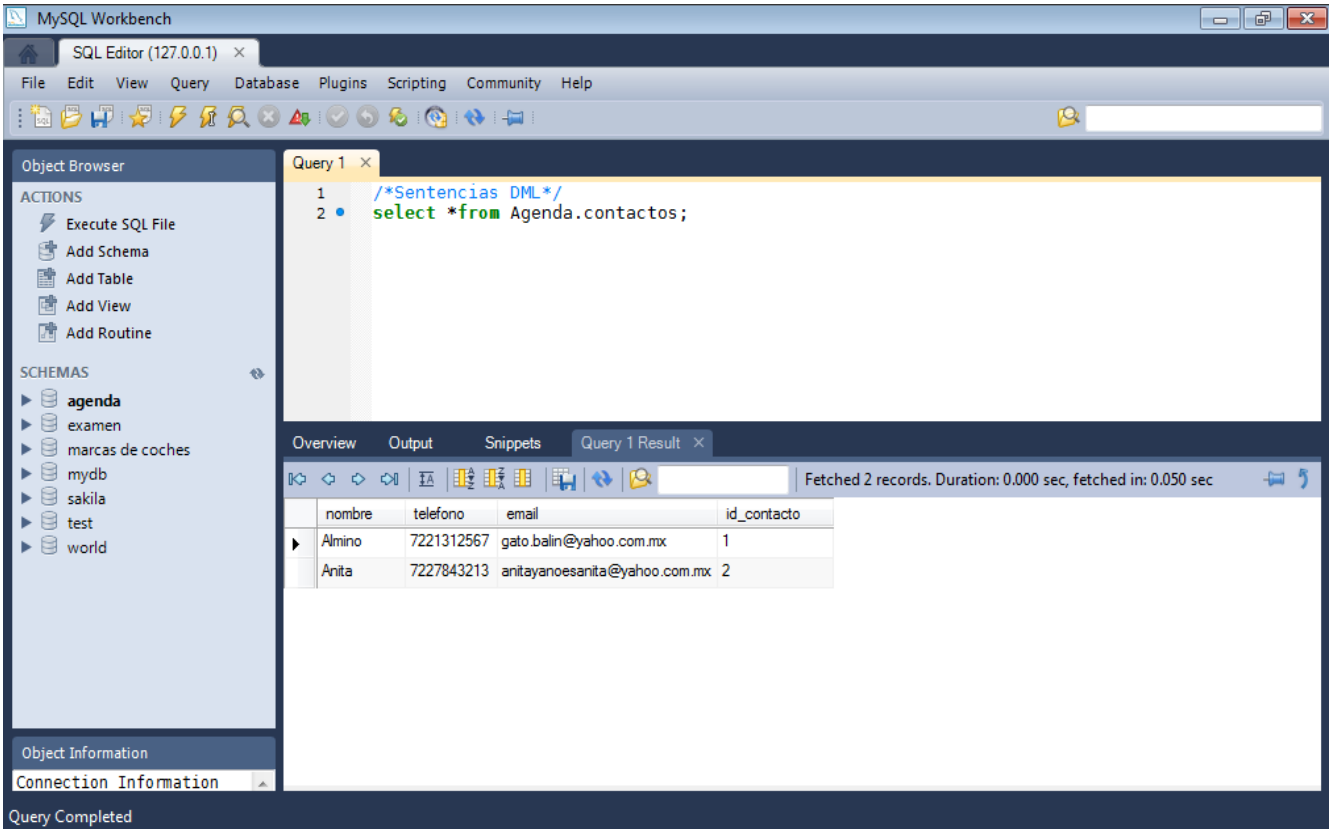
```
/*Sentencias DDL*/
create database Agenda;
use Agenda;
create table Agenda.contactos(
      nombre varchar(45),
      telefono varchar(12),
      email varchar(50),
      id_contacto int primary key);
```



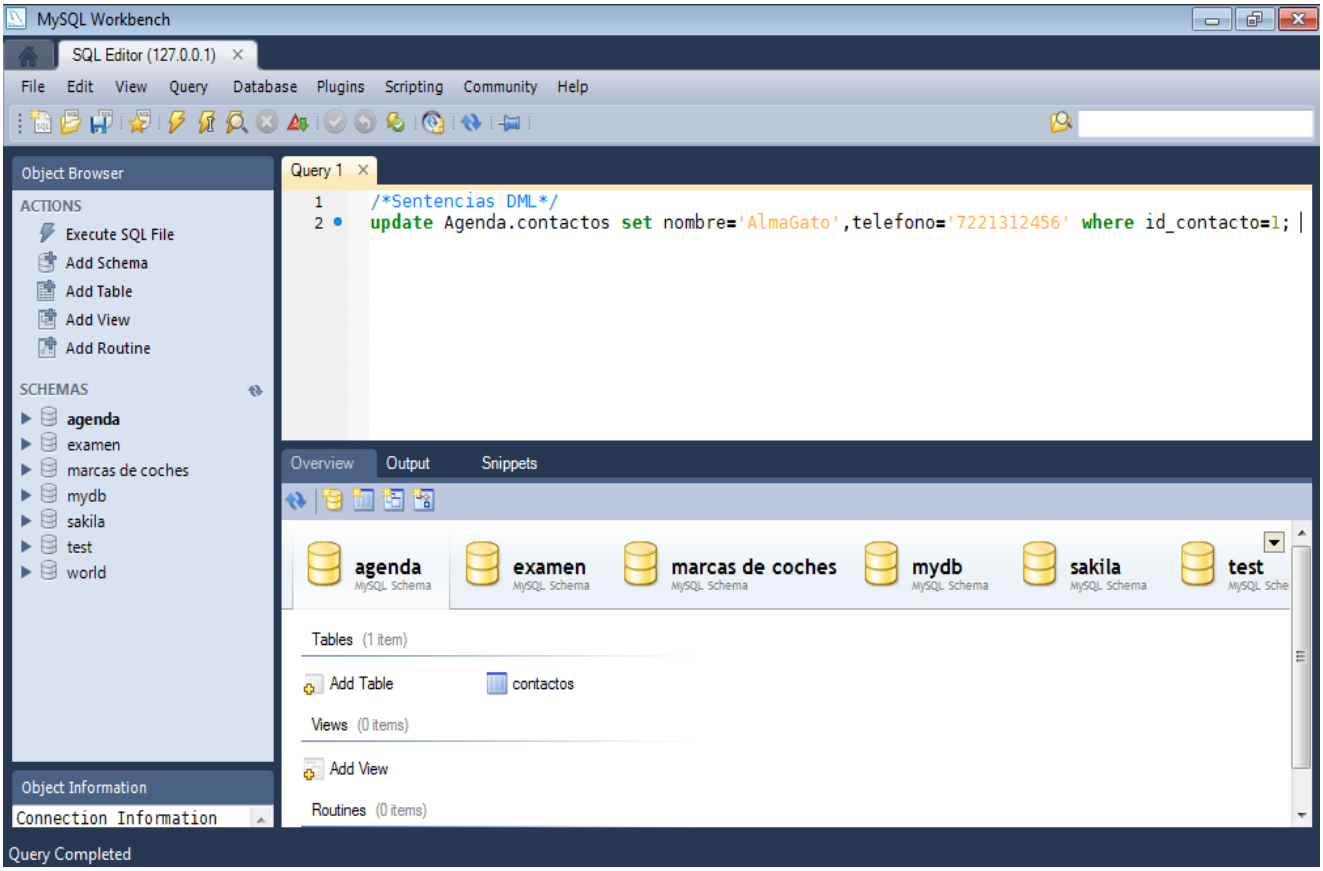

```
/*Sentencias DML*/
insert into Agenda.contactos values ('Almino','7221312567','gato.balin@yahoo.com
.mx',1),('Anita','7227843213','anitayanoesanita@yahoo.com.mx',2);
```



```
/*Sentencias DML*/
select *from Agenda.contactos;
```

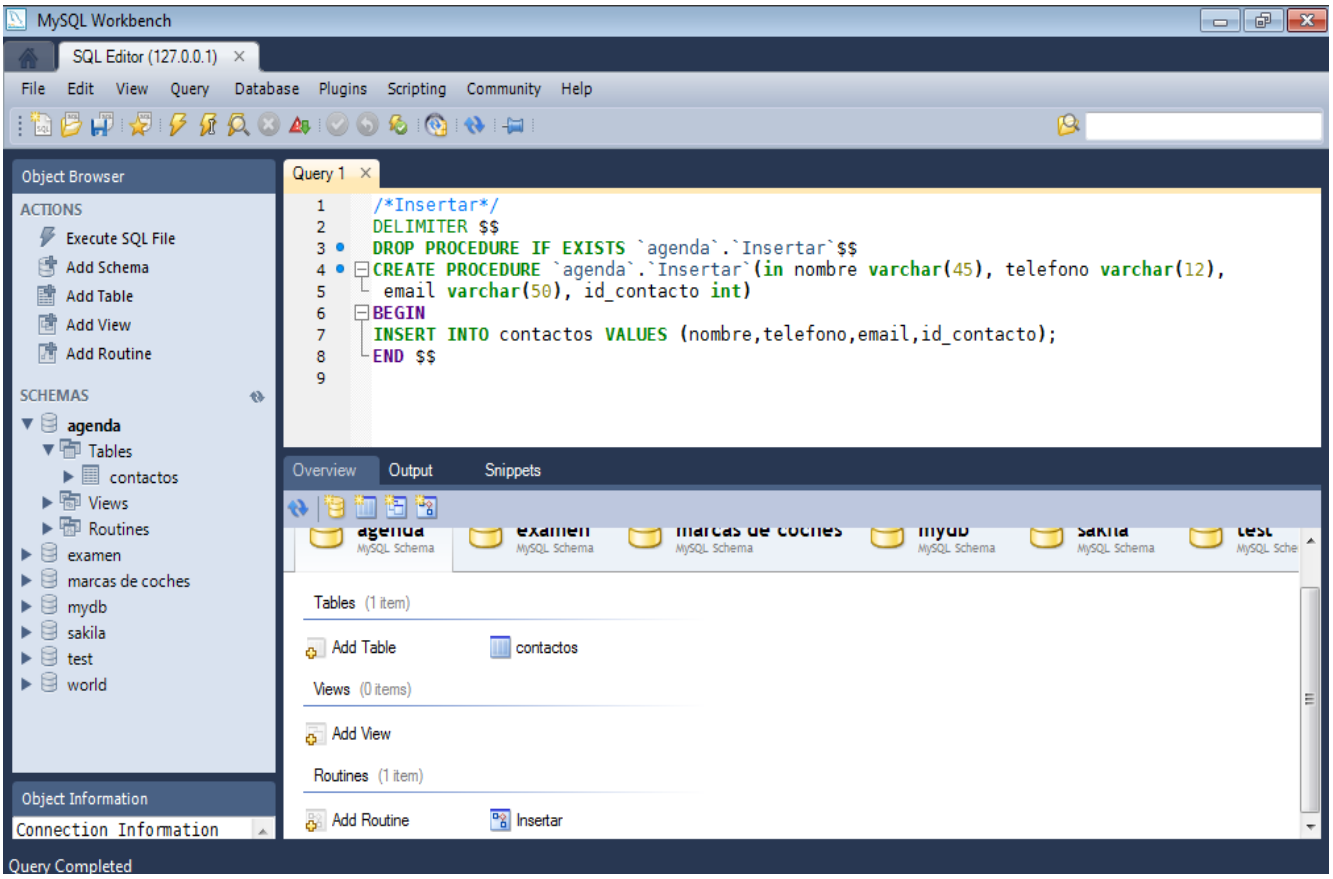



```
/*Sentencias DML*/
update Agenda.contactos set nombre='AlmaGato',telefono='7221312456' where id_con
tacto=1;
```

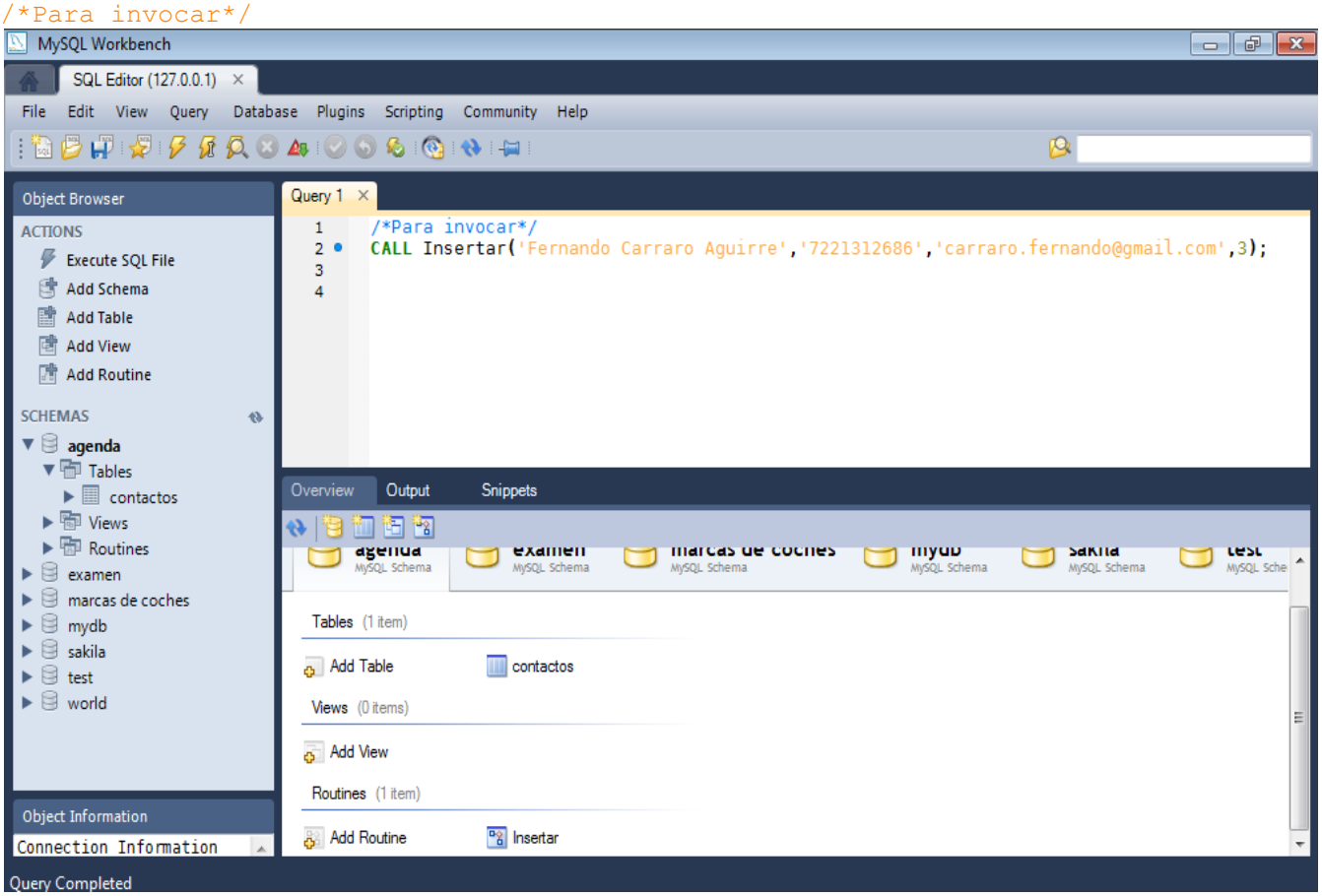


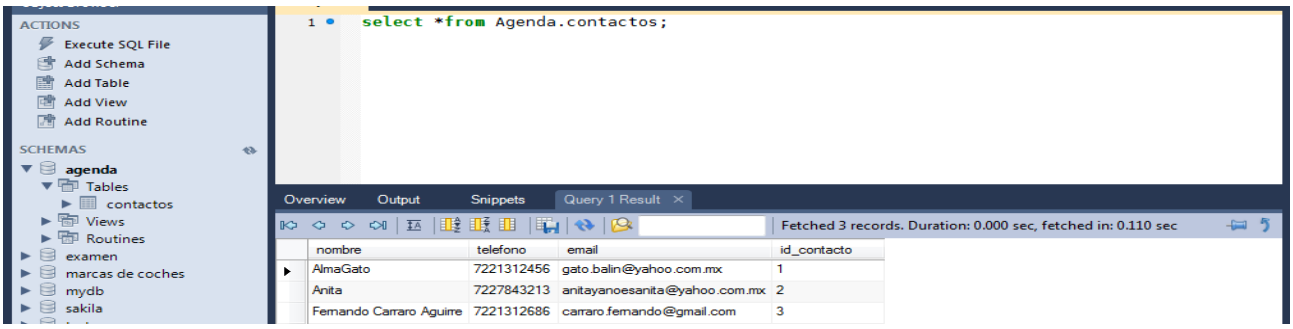
2.- Hacer varios procedimientos para insertar, borrar y actualizar registros.

```
/*Insertar*/
```

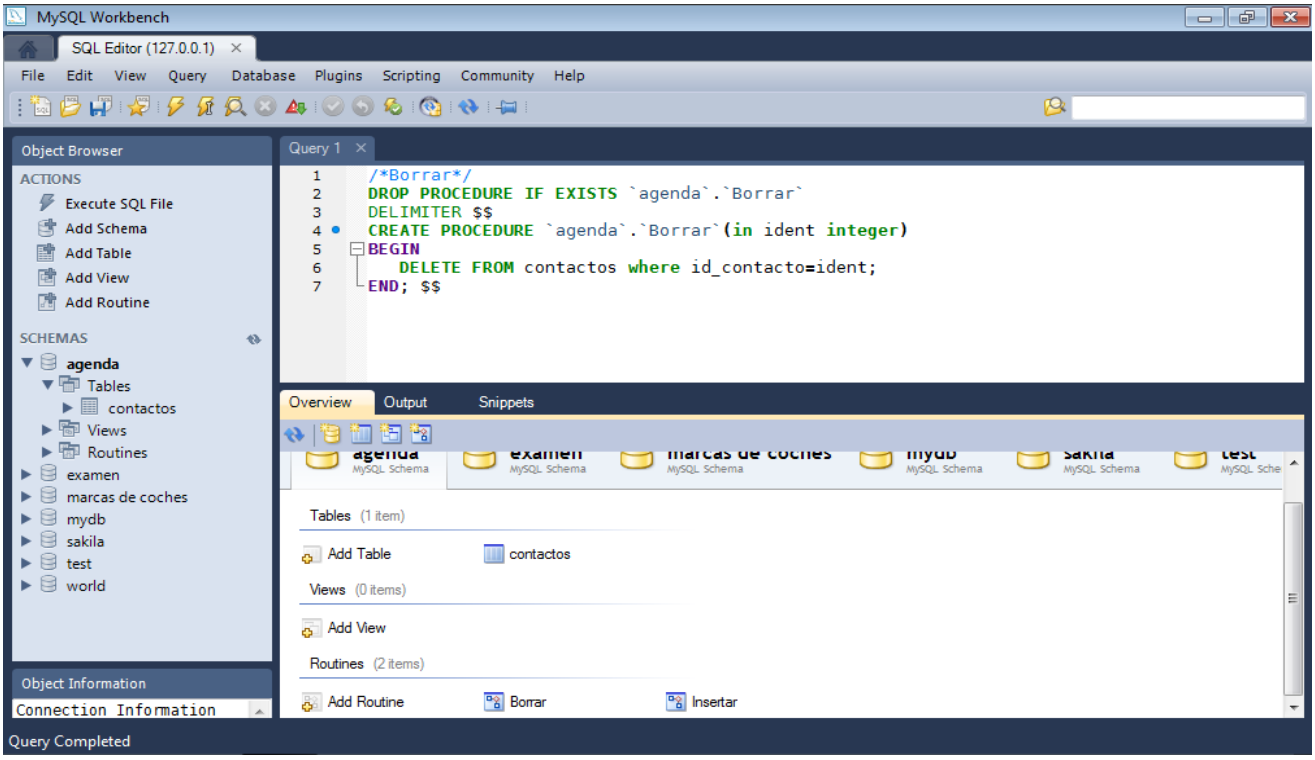



```
/*Para invocar*/
```

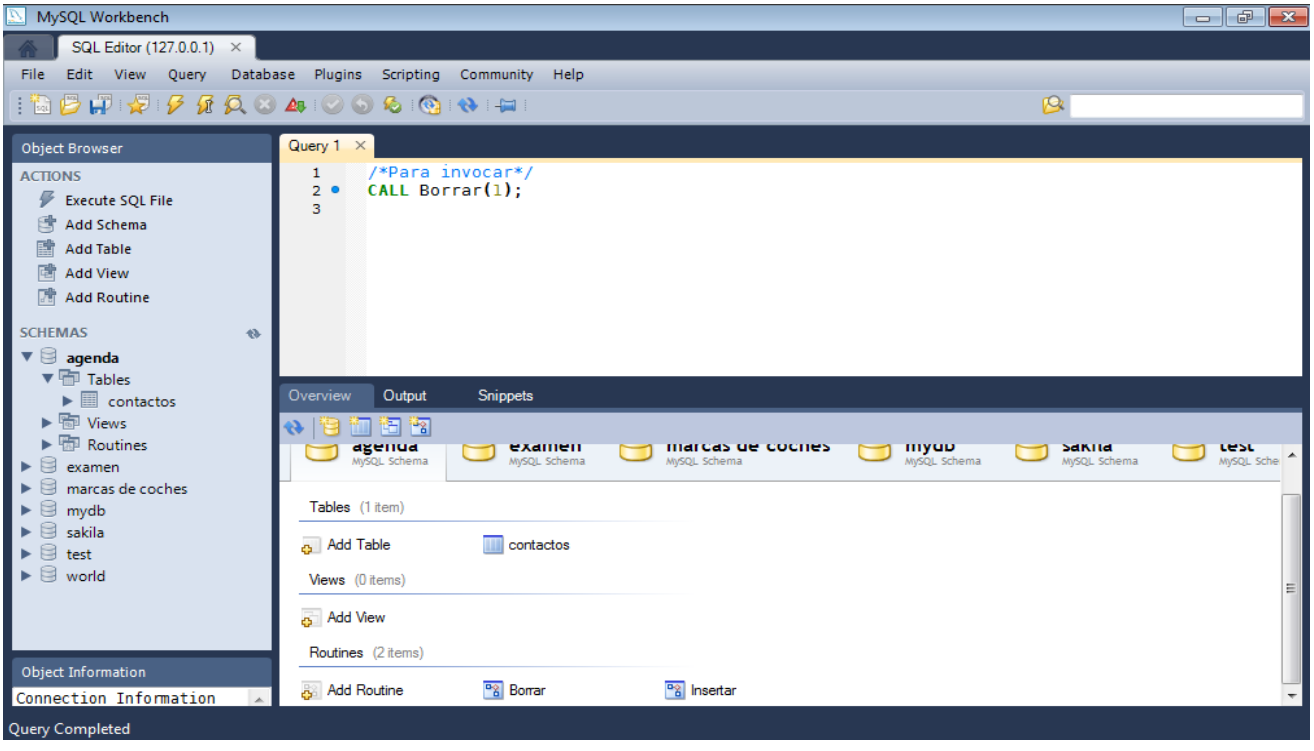


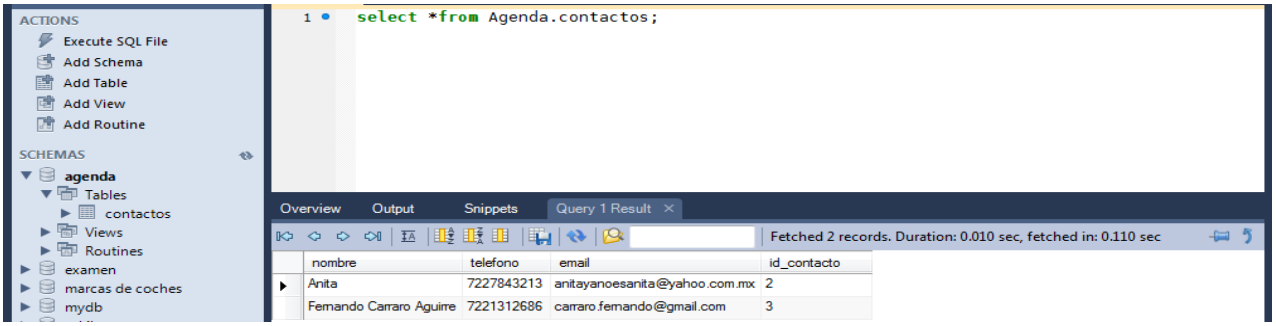



```
/*Borrar*/
```

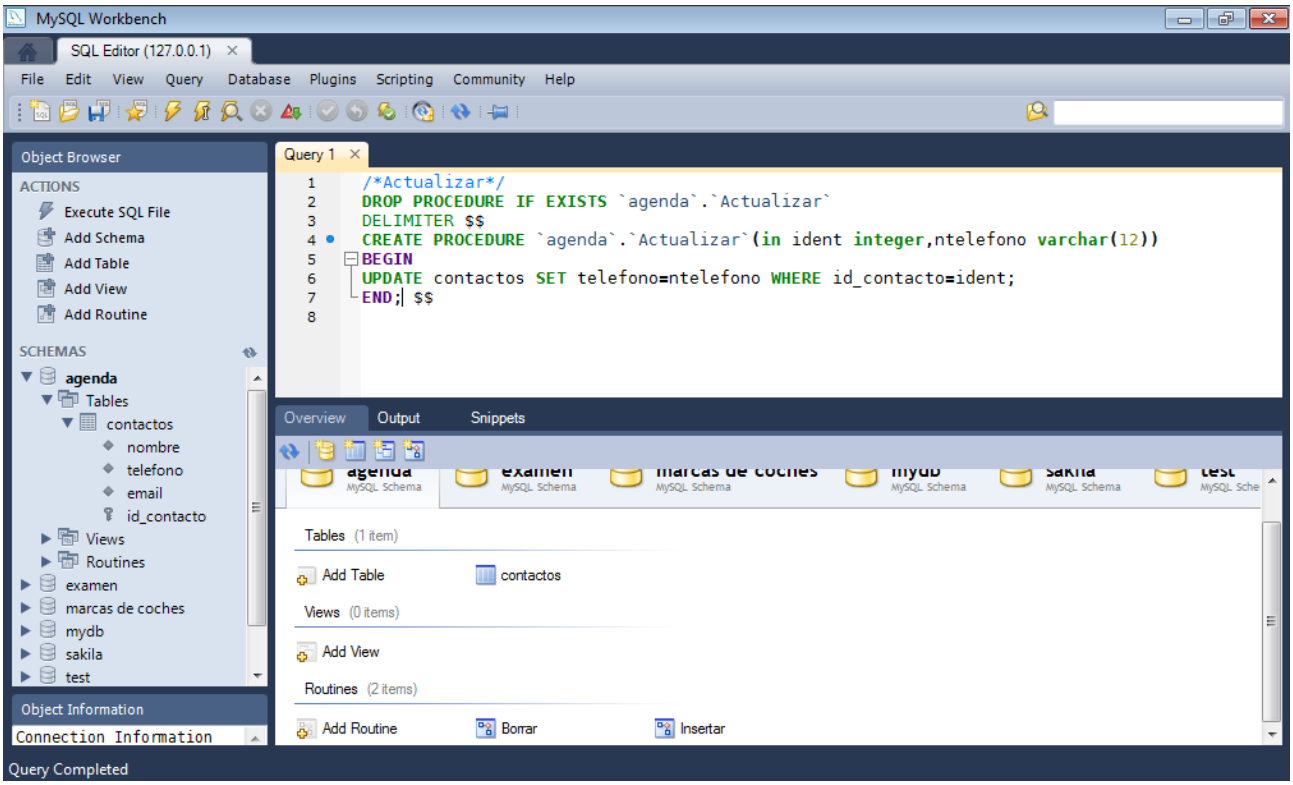



```
//Para invocar
```






```
//Actualizar
```




```
//Para invocar
```