Ejercicio 5

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `ejer5b`()
BEGIN
/* declaramos y definimos */
DECLARE i INTEGER;
DECLARE j INTEGER;
DECLARE suma INTEGER;
SET i = 0;
SET j = 10;
SET suma = 0;
WHILE i <= j DO
   SET suma = suma + i;
   SET i = i + 1;
END WHILE;
/* mostramos el resultado separado por comas */
SELECT suma;
END
```

Ejercicio 5 (utilizando loop)

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `ejer5c`()
BEGIN
/* declaramos y definimos */
DECLARE i INTEGER;
DECLARE j INTEGER;
DECLARE suma INTEGER;
SET i = 0;
SET j = 10;
SET suma = 0;
loop1: WHILE i <= j DO
   SET suma = suma + i;
   SET i = i + 1;
END WHILE loop1;
SELECT suma;
END
```

Ejercicio 8 .- No se puede

Ejercicio 9.- Si se puede

Ejercicios 10.- pagina 20 de la teoría

Ejercicio 11

Crea un trigger en la tabla productos de la base de datos Q3ERP para que cuando se inserte un artículo verifique si el precio es positivo. En caso contrario pondrá el precio a cero.

```
DELIMITER //
CREATE TRIGGER productos_BI_trigger BEFORE INSERT ON productos FOR EACH ROW
BEGIN
 IF (NEW.precio < 0) THEN
    SET NEW.precio := 0;
 END IF;
END//
DELIMITER ;

INSERT INTO productos VALUES(DEFAULT,4,2,100, "Impresora3D", -50, 1, '***');
```

Ejercicio 13.

Crea un trigger en todas las tablas de la base de datos Q3ERP que cada vez que se modifique algo se guarde la modificación en la tabla log.

```
3   -- TRIGGER SOBRE LA TABLA PRODUCTOS
4   DROP TRIGGER IF EXISTS productos_AU_trigger;
5   DELIMITER //
6   CREATE TRIGGER productos_AU_trigger AFTER UPDATE ON productos FOR EACH ROW
7   BEGIN
8       DECLARE descripcion TEXT;
9       -- NOW() es una función que nos permite obtener la fecha y hora actual
10      -- CURRENT_USER es una variable que contiene el usuario actual
11      SET descripcion := CONCAT_WS(' ','UPDATE ON productos. OLD: (', OLD.producto, OLD.precio,')',
12                                                          'NEW: (', NEW.producto, NEW.precio,')');
13      INSERT INTO log VALUES (DEFAULT, NOW(), CURRENT_USER, descripcion);
14  END//
15  DELIMITER ;
16  -- Comprobamos que funciona
17  SELECT * FROM productos;
18  UPDATE productos SET precio = 200 WHERE producto_id = 1;
19  SELECT * FROM log;
```

Ejercicio 15.

Crea un procedimiento que tenga como parámetro de entrada un número real y como parámetro de salida una

-- cadena de caracteres. El parámetro de salida indicará si el número es positivo, negativo o cero.

-- Llamamos el procedimiento pasándole un 5

CALL ejercicio15(5,@signo); -- Como la llamada está fuera de un procedimiento la variable mi_var debe ir con @ delante

SELECT @signo; -- Mostramos el valor de signo, que será "positivo"

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `ejercicio15`(IN num REAL)
BEGIN
DECLARE signo varchar(20);
    IF (num > 0) THEN
        SET signo := 'Positivo';
    ELSEIF (num < 0) THEN
        SET signo := 'Negativo';
    ELSE
        SET signo := 'Cero';
    END IF;
    select signo;
END
```

Ejercicio 16.

Escribe un procedimiento que reciba como entrada la nota de un alumno (numérico real) y un parámetro de salida

-- (cadena de caracteres) con las siguientes condiciones: [0, 5): Insuficiente / [5, 6): Aprobado / [6, 7): Bien / [7, 9): Notable

-- [9, 10]: Sobresaliente / En cualquier otro caso la nota no será válida.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `ejercicio16`(IN nota INT, OUT descripcion VARCHAR(20))
BEGIN
    IF (nota < 0 OR nota > 10) THEN
        SET descripcion := 'Nota inválida';
    ELSEIF (nota < 5) THEN
        SET descripcion := 'Insuficiente';
    ELSEIF (nota < 6) THEN
        SET descripcion := 'Aprobado';
    ELSEIF (nota < 7) THEN
        SET descripcion := 'Bien';
    ELSEIF (nota < 9) THEN
        SET descripcion := 'Notable';
    ELSEIF (nota <= 10) THEN
        SET descripcion := 'Sobresaliente';
    END IF;

END
```

Ejercicio 18 .- Hacer

Ejercicio 19. Crea una tabla llamada Fibonacci. Inserta los 50 primeros términos de la serie mediante un procedimiento almacenado. Utiliza la instrucción REPEAT.

-- Para quien no conozca la secuencia de Fibonacci:

-- Es una sucesión que comienza con los números 0 y 1. A partir de estos, «cada término es la suma de los dos anteriores»

-- Creamos la tabla, se ha elegido el tipo BIGINT porque los números que tendremos que almacenar superan el límite de INT que es 2147483647

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `ejercicio19`()
BEGIN
DECLARE contador INT;
    DECLARE valor_penultimo BIGINT; -- Almacena el penúltimo valor calculado de la secuencia
    DECLARE valor_ultimo BIGINT; -- Almacena el último valor calculado de la secuencia
    DECLARE valor_actual BIGINT; -- Almacena el valor actual de la secuencia

    SET contador := 0; -- Cuántos números hemos contado

    SET valor_penultimo := 0;
    SET valor_ultimo := 0;
    SET valor_actual := 0;

    REPEAT
        SET valor_actual := valor_ultimo + valor_penultimo; -- En la secuencia de Fibonacci «cada término es la suma de los dos anteriores», por eso el valor que calcu
        INSERT INTO fibonacci VALUES(valor_actual); -- Lo almacenamos

        IF valor_actual = 0 THEN
            -- Si el resultado de obtener el valor actual es 0 significa que es el primer valor que hemos calculado, el siguiente debería ser un 1
            -- Para conseguirlo guardamos un 1 en el valor penúltimo, de esa forma, en la próxima iteración 0 (valor último) + 1 (valor penúltimo) resultará ser 1.
            SET valor_penultimo := 1;
        ELSE
            -- En caso contrario "desplazamos" los números, de forma que el último pasa a ser el penúltimo y el actual pasa a ser el último
            SET valor_penultimo := valor_ultimo;
            SET valor_ultimo := valor_actual;
        END IF;

        SET contador := contador + 1;
    UNTIL contador = 50
    END REPEAT;
END
```

Hacer 18,20,21,22,26,27,28

Ejercicio 23

```sql
CREATE FUNCTION raiz (num REAL) RETURNS REAL DETERMINISTIC
BEGIN
    DECLARE res REAL;
    SET res := SQRT(num);
    IF res IS NULL THEN
        SET res := -1;
    END IF;
    RETURN res;
END
```

Ejercicio 26

```sql
CREATE DEFINER=`root`@`localhost` FUNCTION `ejercicio26`() RETURNS int
    DETERMINISTIC
BEGIN
    DECLARE num_productos INT;
    -- En una SELECT podemos hacer uso de la cláusula INTO para almacenar el valor obtenido en una variable
    SELECT COUNT(*) INTO num_productos FROM productos;
    RETURN num_productos;
END
```