**UNIDAD 8. PROGRAMACIÓN DE BASES DE DATOS**

**Programación en Bases de Datos (Nivel Básico)**

**✅ 1. Introducción a la programación en bases de datos**

En una base de datos, además de almacenar datos, también podemos **programar lógica** que se ejecuta dentro del propio motor de la base de datos. Esta programación se puede usar para:

* Automatizar tareas.
* Centralizar la lógica de negocio (por ejemplo, cálculos, validaciones).
* Reutilizar código (mediante funciones o procedimientos).
* Mejorar el rendimiento al reducir viajes entre la base de datos y la aplicación.

**🔧 ¿Qué se puede programar en una base de datos?**

* **Procedimientos almacenados (Stored Procedures):** bloques de código que realizan acciones, con o sin parámetros.
* **Funciones:** devuelven un valor y pueden ser usadas en consultas.
* **Triggers (disparadores):** código que se ejecuta automáticamente al producirse un evento (INSERT, UPDATE, DELETE).
* **Cursores y control de flujo:** para manejar filas una a una y aplicar lógica condicional o bucles.

**✅ 2. Crear un Procedimiento Almacenado en MySQL Workbench**

**¿Qué es un procedimiento almacenado?**

Un **procedimiento almacenado** es un bloque de código SQL que se guarda en la base de datos y se puede ejecutar cuando sea necesario. Puede aceptar parámetros de entrada y salida, y permite automatizar tareas repetitivas.

**🧪 Crear el procedimiento desde MySQL Workbench**

1. Abre MySQL Workbench y conéctate a tu base de datos.
2. En el panel izquierdo, busca tu base de datos y expande el nodo **Stored Procedures**.
3. Haz clic derecho y selecciona **“Create Stored Procedure…“**.
4. Se abrirá una pestaña con una plantilla como esta:

CREATE PROCEDURE `ejemplo1`()

BEGIN

-- tu código aquí

END

1. Escribe tu código, haz clic en **Apply**, revisa y luego en **Finish**.

**📝 Notas importantes**

* **Siempre cambia el delimitador** si usas BEGIN...END para evitar errores de ejecución.
* Si el procedimiento ya existe y quieres cambiarlo, primero debes borrarlo con:
* DROP PROCEDURE IF EXISTS nombre\_procedimiento;
* Todos los procedimientos tienen un nombre seguido de paréntesis.
* El código que queremos ejecutar se escribe entre **BEGIN** y **END**.

**✅ 3. Variables**

Una **variable** es un espacio en memoria que se utiliza para **guardar temporalmente un valor** durante la ejecución de un bloque de código.

**📌 ¿Para qué sirven?**

* Para almacenar datos intermedios.
* Para contar, acumular, comparar o devolver resultados.

**📐 Declaración de variables en MySQL**

Las variables deben declararse **al principio** de un bloque BEGIN...END, normalmente dentro de un procedimiento o función.

DECLARE nombre\_variable TIPO;

**📌 Ejemplos:**

Vamos a utilizar el ejemplo1() creado en el apartado anterior para ir haciendo pruebas.

DECLARE contador INT;

DECLARE nombre\_cliente VARCHAR(50);

DECLARE fecha\_actual DATE;

**🔄 Asignación de valores**

SET contador = 10;

SET nombre\_cliente = 'María López';

SET fecha\_actual = CURDATE(); -- Asigna la fecha actual

También se pueden usar en operaciones:

SET contador = contador + 1;

Para mostrar los datos, utilizamos la sentencia SELECT.

SELECT contador, nombre\_cliente, fecha\_actual;

Guardamos los cambios con el botón **Apply**, revisa y luego en **Finish**.

Una vez guardado el Procedimiento, aparece almacenado como un objeto de la base de datos, en la categoría **Stored Procedures**. Si ahora quisieramos ejecutar este procedimiento, escribimos en una hoja en blanco la siguiente instrucción:

CALL ejemplo1();

**🧪 Ejemplo completo**

Realizamos ahora un ejemplo completo repitiendo los pasos anteriores con el siguiente procedimiento.

DELIMITER //

CREATE PROCEDURE ejemplo\_variables()

BEGIN

DECLARE edad INT;

DECLARE nombre VARCHAR(30);

SET edad = 25;

SET nombre = 'Carlos';

SELECT CONCAT(nombre, ' tiene ', edad, ' años') AS mensaje;

END //

DELIMITER ;

CALL ejemplo\_variables();

**✅ 4. Procedimientos almacenados (Stored Procedures)**

Un **procedimiento almacenado** es un bloque de código SQL que se guarda en la base de datos y se puede ejecutar cuantas veces queramos. Sirve para automatizar tareas, encapsular lógica y reutilizar código.

**🎯 ¿Por qué usar procedimientos?**

* Para **automatizar procesos** complejos o repetitivos.
* Para **organizar mejor** el código SQL.
* Para **reducir errores** al evitar escribir consultas largas cada vez.
* Para **centralizar la lógica** de negocio en la base de datos.

**🧱 Sintaxis básica de un procedimiento en MySQL**

DELIMITER //

CREATE PROCEDURE nombre\_procedimiento()

BEGIN

-- Código SQL aquí

END //

DELIMITER ;

✅ Es importante cambiar el delimitador (DELIMITER //) para que MySQL no interprete el ; del interior del bloque como el final de la sentencia.

**▶️ Ejecutar un procedimiento**

CALL nombre\_procedimiento();

**🧪 Ejemplo 1: Procedimiento sin parámetros**

Este ejemplo le hacemos directamente desde una ventana de trabajo normal, para comprender el uso de los **DELIMITER**.

DELIMITER //

CREATE PROCEDURE saludar()

BEGIN

SELECT '¡Hola desde la base de datos!' AS saludo;

END //

DELIMITER ;

CALL saludar();

Un **parámetro de un procedimiento almacenado** es una variable que se utiliza para **pasar datos al procedimiento** (entrada), **recibir datos desde el procedimiento** (salida), o **ambas cosas** (entrada/salida), permitiendo que el procedimiento sea reutilizable y adaptable a diferentes situaciones.

**🧪 Ejemplo 2: Procedimiento con parámetro de entrada**

Este ejemplo le hacemos desde crear un nuevo procedimiento almacenado.

DELIMITER //

CREATE PROCEDURE saludar\_persona(nombre\_usuario VARCHAR(50))

BEGIN

SELECT CONCAT('Hola, ', nombre\_usuario, '!') AS saludo;

END //

DELIMITER ;

CALL saludar\_persona('María');

CALL saludar\_persona('David');

**🔄 Parámetros en procedimientos**

Un **parámetro de un procedimiento almacenado** es una variable especial que se declara en la cabecera del procedimiento y que permite **intercambiar información entre el procedimiento y el entorno que lo invoca**. Los parámetros hacen que el procedimiento sea **más flexible y reutilizable**, ya que permiten adaptar su comportamiento según los valores que reciba o devuelva.

Existen tres tipos de parámetros:

* **IN (entrada):** se usan para enviar valores al procedimiento desde el exterior. Son de solo lectura dentro del procedimiento.
* **OUT (salida):** permiten devolver un valor desde el procedimiento hacia el entorno que lo llamó.
* **INOUT (entrada/salida):** sirven tanto para recibir un valor al inicio como para devolver un valor modificado al final.

Estos parámetros son útiles cuando queremos encapsular lógica en procedimientos que puedan trabajar con distintos datos sin reescribir el código.

Vamos a ver tres ejemplos distintos del uso de los parámetros.

**✅ 1. Procedimiento con parámetro de entrada**

**Ejemplo:** Obtener el nombre de un grupo a partir de su ID.  
**Base de datos:** concursoMusica. Tenemos que tener seleccionada la base de datos indicada.

DELIMITER //

CREATE PROCEDURE ObtenerNombreGrupo (IN p\_id\_grupo INT)

BEGIN

SELECT nombre

FROM grupos

WHERE codgrupo = p\_id\_grupo;

END //

DELIMITER ;

**Uso:**

CALL ObtenerNombreGrupo(2);

**✅ 2. Procedimiento con parámetro de salida**

**Ejemplo:** Devolver cuántas canciones tiene un grupo.  
**Base de datos:** concursoMusica

DELIMITER //

CREATE PROCEDURE ContarCancionesGrupo (IN p\_id\_grupo INT,OUT p\_total INT)

BEGIN

SELECT COUNT(\*) INTO p\_total

FROM canciones

WHERE grupo = p\_id\_grupo;

END //

DELIMITER ;

**Uso:**

CALL ContarCancionesGrupo(1, @total);

SELECT @total;

Para poder darle valor a un parámetro de tipo OUT o de salida, se utiliza la instrucción **INTO** dentro de una **SELECT**. En el ejemplo, guarda el total de las canciones dentro de la variable p\_total.

🔹 ¿Qué es **@total**?

@total es una variable definida por el usuario que existe en la sesión actual de MySQL. Se utiliza para interactuar con parámetros de salida o entrada/salida de procedimientos almacenados, ya que estos parámetros no pueden ser directamente mostrados como resultados por el CALL, sino que deben guardarse en una variable externa para consultarlos después.

🔹 ¿Por qué no se usa una variable normal (DECLARE)?

Las variables declaradas con DECLARE solo existen dentro de procedimientos o bloques BEGIN…END, por lo que no pueden usarse en el exterior para capturar resultados de un CALL. En cambio, las variables de usuario (@nombre\_variable) persisten durante la sesión y son accesibles fuera del procedimiento.

**✅ 3. Procedimiento con parámetro de entrada/salida**

**Ejemplo:** Aumentar el número de votos de una cancion concreta y devolver el nuevo valor. (Se trata de modificar el campo total\_votos de la tabla canciones). **Base de datos:** concursoMusica

DELIMITER //

CREATE PROCEDURE AumentarVotos (IN p\_numCancion INT, INOUT p\_nuevo\_total INT)

BEGIN

UPDATE canciones

SET total\_votos = total\_votos + p\_nuevo\_total

WHERE numCancion = p\_numCancion;

SELECT total\_votos INTO p\_nuevo\_total

FROM canciones

WHERE numCancion = p\_numCancion;

END //

DELIMITER ;

**Uso:**

SELECT total\_votos FROM canciones WHERE numCancion=1;

SET @incremento = 2;

CALL AumentarVotos(1, @incremento);

SELECT @incremento; -- tiene que dar lo mismo que la consulta

SELECT total\_votos FROM canciones WHERE numCancion=1;

**🧠 Ejercicios propuestos**

1. Crea un procedimiento llamado mostrar\_fecha que muestre la fecha actual.

DELIMITER $$

USE `tema6ejemplos`$$

-- 1. Crea un procedimiento llamado mostrar\_fecha que muestre la fecha actual.

CREATE PROCEDURE `mostrar\_fecha` ()

BEGIN

select curdate();

END$$

DELIMITER ;

1. Crea un procedimiento llamado cuadrado\_numero que reciba un número y muestre su cuadrado.

DELIMITER $$

USE `tema6ejemplos`$$

-- 2. Crea un procedimiento llamado cuadrado\_numero que reciba un número y muestre su cuadrado.

CREATE PROCEDURE `cuadrado\_numero` (IN num int)

BEGIN

select pow(num,2) as cuadrado;

END$$

DELIMITER ;

1. Crea un procedimiento llamado info\_usuario que reciba un nombre y un apellido y muestre un mensaje de bienvenida personalizado.

DELIMITER $$

USE `tema6ejemplos`$$

CREATE DEFINER=`root`@`%` PROCEDURE `info\_usuario`(in nombre varchar(30), in apellido varchar(50))

BEGIN

select concat('Bienvenido ', upper(nombre),' ',upper(apellido),' a nuestra clase') as mensaje;

END$$

DELIMITER ;

1. Crea un procedimiento llamado mostrar\_canciones que muestre todas las canciones de un grupo a partir de su nombre.

DELIMITER $$

USE `concursomusica`$$

-- 4. Crea un procedimiento llamado mostrar\_canciones que muestre todas las canciones de un grupo a partir de su nombre.

CREATE PROCEDURE `mostrar\_canciones` (in nomgrupo varchar(30))

BEGIN

select \* from canciones inner join grupos on codgrupo=grupo where nombre=nomgrupo;

END$$

DELIMITER ;

1. Crea un procedimiento llamado `insertar\_voto' que recibe como parámetro el título de la canción y el nombre y apellido de la persona que vota. Se entiende que el voto se hace en el día de hoy. El procedimiento devuelve el número de votos que ha hecho esa persona.

DELIMITER $$

USE `concursomusica`$$

/\*Crea un procedimiento llamado `insertar\_voto' que recibe como parámetro el título de la canción y el

nombre y apellido de la persona que vota. Se entiende que el voto se hace en el día de hoy. El procedimiento

devuelve el número de votos que ha hecho esa persona. (Si la inserción a realizar da error, bien por que no existe

el usuario o porque el voto está duplicado, no devuelve el número de votos).\*/

CREATE DEFINER=`root`@`%` PROCEDURE `insertar\_voto`(IN tit varchar(40),in nom varchar(15),in ape varchar(30), out resul int)

BEGIN

set resul=0;

insert ignore into votos values

((select user from usuarios where nombre=nom and apellidos=ape),

curdate(),

(select numCancion from canciones where titulo=tit));

select count(\*) into resul from votos where usuario=(select user from usuarios where nombre=nom and apellidos=ape);

END$$

DELIMITER ;

1. Crea un procedimiento llamado Aumentar\_segundos que aumenta en X segundos la duración de una canción y devolve su nueva duración.

CREATE DEFINER=`root`@`%` PROCEDURE `aumentar\_segundos`(in tit varchar(40), in X int, out tiempo time)

BEGIN

update canciones

set duracion=addtime(duracion, concat('00:00:',x))

where titulo=tit;

select duracion into tiempo from canciones where titulo=tit;

END

1. Crea un procedimiento llamado Borrar\_canciones que elimina todas las canciones de un grupo dado por su nombre.

CREATE DEFINER=`root`@`%` PROCEDURE `borrar\_canciones`(in nom varchar(30))

BEGIN

delete canciones from canciones inner join grupos on codgrupo=grupo where nombre=nom;

END

1. Crea un procedimiento llamado Contar cuántos componentes tiene un grupo. El parámetro que pasamos al procedimiento es el nombre del grupo. El número de componentes lo devuelve como parámetro.

CREATE DEFINER=`root`@`%` PROCEDURE `contar`(in nom varchar(30), out resul int)

BEGIN

select count(\*) into resul from componentes inner join grupos on codgrupo=grupo where grupos.nombre=nom;

END

9. Crea un procedimiento llamado `grupos` que crea una tabla con dos campos, nombre, que guardará el nombre de los grupos, y canciones, que será un campo que guarda cuantas canciones tiene cada grupo. Una vez creada la tabla, hay que rellenarla con los datos correspondientes. Por último, muestra el contenido de la tabla.

CREATE DEFINER=`root`@`%` PROCEDURE `insertar\_voto`(IN tit varchar(40),in nom varchar(15),in ape varchar(30), out resul int)

BEGIN

set resul=0;

insert ignore into votos values

((select user from usuarios where nombre=nom and apellidos=ape),

curdate(),

(select numCancion from canciones where titulo=tit));

select count(\*) into resul from votos where usuario=(select user from usuarios where nombre=nom and apellidos=ape) and fecha=curdate();

END

💡 Consejo: prueba cada procedimiento y modifica valores para ver cómo se comporta.

**✅ 5. Funciones**

Una **función** es un bloque de código almacenado en la base de datos que **devuelve un único valor**. A diferencia de los procedimientos, las funciones **pueden utilizarse dentro de una consulta**, como si fueran una función de MySQL (NOW(), CONCAT(), etc.).

**🔍 Diferencias entre procedimiento y función**

| **Característica** | **Procedimiento** | **Función** |
| --- | --- | --- |
| Devuelve un valor | No necesariamente | Sí, siempre |
| Se usa en consultas | No | Sí (SELECT, WHERE) |
| Se invoca con… | CALL procedimiento() | SELECT funcion() |
| Puede modificar datos | Sí | No (solo lectura) |

**🧱 Sintaxis básica de una función en MySQL**

DELIMITER //

CREATE FUNCTION nombre\_funcion(parametro TIPO) RETURNS TIPO

BEGIN

DECLARE resultado TIPO;

-- lógica

SET resultado = ...;

RETURN resultado;

END //

DELIMITER ;

**🧪 Ejemplo 1: Función que devuelve el doble de un número**

DELIMITER //

CREATE FUNCTION doble(numero INT) RETURNS INT

BEGIN

RETURN numero \* 2;

END //

DELIMITER ;

SELECT doble(5); -- Resultado: 10

**🧪 Ejemplo 2: Función que indica si alguien es mayor de edad**

DELIMITER //

CREATE FUNCTION es\_mayor\_edad(edad INT) RETURNS VARCHAR(20)

BEGIN

RETURN IF(edad >= 18, 'Mayor de edad', 'Menor de edad');

END //

DELIMITER ;

SELECT es\_mayor\_edad(20); -- Resultado: 'Mayor de edad'

**🧠 Ejercicios propuestos**

1. Crea una función llamada saludo\_personal que reciba un nombre y devuelva un saludo tipo “Hola, Juan”.

CREATE DEFINER=`root`@`%` FUNCTION `saludo\_personal`(nombre varchar(30)) RETURNS varchar(50) CHARSET utf8mb4

DETERMINISTIC

BEGIN

RETURN concat("Hola, ", nombre);

END

1. Crea una función llamada area\_circulo que reciba el radio de un círculo y devuelva su área (π \* r²). Usa PI() y POW() de MySQL.

CREATE DEFINER=`root`@`%` FUNCTION `area\_circulo`(r float) RETURNS float

DETERMINISTIC

BEGIN

RETURN pi()\*pow(r,2);

END

1. Crea una función llamada iva\_incluido que reciba un precio y devuelva el precio con un 21% de IVA.

CREATE DEFINER=`root`@`%` FUNCTION `iva\_incluido`(precio float) RETURNS float

DETERMINISTIC

BEGIN

RETURN precio\*1.21;

END

1. Crear una función llamada `total\_canciones` que, dado el ID de un grupo, devuelva el número total de canciones registradas de ese grupo. (Base de datos ConcursoMusica)

CREATE DEFINER=`root`@`%` FUNCTION `total\_canciones`(p\_idgrupo int) RETURNS int

DETERMINISTIC

BEGIN

declare resul int;

select count(\*) into resul from canciones where grupo=p\_idgrupo;

RETURN resul;

END

1. Crear una función llamada `nombre\_grupo` que, dado el ID de un componente, devuelva el nombre del grupo al que pertenece.

CREATE DEFINER=`root`@`%` FUNCTION `nombre\_grupo`(p\_idcompo int) RETURNS varchar(30) CHARSET utf8mb4

DETERMINISTIC

BEGIN

declare resul varchar(30);

select grupos.nombre into resul from grupos inner join componentes on grupo=codgrupo where idComp=p\_idcompo;

RETURN resul;

END

1. Crear una función llamada `duracion\_media` que reciba el nombre de un grupo y devuelva la duración media de sus canciones.

CREATE DEFINER=`root`@`%` FUNCTION `duracion\_media`(p\_nombre varchar(30)) RETURNS time

DETERMINISTIC

BEGIN

declare resul TIME;

select avg(duracion) into resul from canciones inner join grupos on codgrupo=grupo where nombre=p\_nombre;

RETURN resul;

END

1. Crear una función llamada `votado` que, dado el ID de un usuario, devuelva cuantas canciones ha votado.

CREATE DEFINER=`root`@`%` FUNCTION `votado`(p\_idusuario varchar(15)) RETURNS int

DETERMINISTIC

BEGIN

declare resul int;

select count(\*) into resul from votos where usuario=p\_idusuario;

RETURN resul;

END

1. Crear una función llamada `componentes` que reciba el nombre de un grupo y devuelva cuántos componentes forman parte de él.

CREATE DEFINER=`root`@`%` FUNCTION `componentes`(p\_nombre varchar(30)) RETURNS int

DETERMINISTIC

BEGIN

declare resul integer;

select count(\*) into resul from componentes inner join grupos on codgrupo=grupo where grupos.nombre=p\_nombre;

RETURN resul;

END

💡 Prueba las funciones usando SELECT directamente.

**🧭 Sección final: Resumen visual / Mapa de conceptos**

**📌 Resumen visual: Programación en bases de datos**

**🧠 Conceptos clave**

* **Variable**: espacio de memoria temporal dentro de un bloque BEGIN...END.
* **Procedimiento (Stored Procedure)**: bloque de código que ejecuta acciones (puede modificar datos).
* **Función**: bloque que devuelve un valor y se puede usar en consultas.
* **Parámetros**:
  + IN: entrada.
  + OUT: salida.
  + INOUT: ambos.
* **DELIMITER**: se usa para indicar a MySQL dónde empieza y termina un bloque de código.

**🧩 Diferencias rápidas**

| **Elemento** | **Usa CALL** | **Devuelve valor** | **Se usa en SELECT** | **Modifica datos** |
| --- | --- | --- | --- | --- |
| Procedimiento | ✅ | Opcional | ❌ | ✅ |
| Función | ❌ | ✅ | ✅ | ❌ |

**🧪 Buenas prácticas**

* Usa nombres claros y descriptivos (calcular\_total, mostrar\_fecha).
* Comenta tu código con -- comentario.
* Prueba tus funciones y procedimientos paso a paso.

**✅ Autoevaluación: Programación en Bases de Datos – Nivel básico**

Responde a las siguientes preguntas tipo test. Solo una opción es correcta en cada caso.

**🧠 Preguntas**

**1. ¿Cuál de estas afirmaciones sobre las variables es correcta?**

a) No pueden usarse dentro de procedimientos.  
b) Solo pueden almacenar números enteros.  
c) Se declaran con DECLARE y pueden usarse dentro de bloques BEGIN...END.  
d) Siempre deben tener el mismo nombre que el procedimiento.

**2. ¿Qué palabra clave se utiliza para crear un procedimiento en MySQL?**

a) MAKE PROCEDURE  
b) NEW PROCEDURE  
c) CREATE PROCEDURE  
d) SET PROCEDURE

**3. ¿Qué instrucción permite ejecutar un procedimiento almacenado?**

a) EXEC  
b) SELECT  
c) RUN  
d) CALL

**4. ¿Qué diferencia principal hay entre una función y un procedimiento?**

a) La función se guarda en la base de datos y el procedimiento no.  
b) El procedimiento devuelve un valor, la función no.  
c) La función puede usarse en consultas SELECT, el procedimiento no.  
d) No hay ninguna diferencia.

**5. ¿Cuál de estas funciones sería válida en una función de MySQL?**

a) CALL saludar();  
b) SELECT saludar();  
c) CREATE PROCEDURE saludar();  
d) RUN saludar();

**6. ¿Qué delimitador se suele usar para definir procedimientos o funciones en MySQL?**

a) DELIMITER //  
b) DELIMITER --  
c) DELIMITER END  
d) DELIMITER $

**7. ¿Cuál de estas opciones es incorrecta sobre las funciones?**

a) Siempre devuelven un valor.  
b) Pueden modificar datos en la base de datos.  
c) Pueden usarse dentro de SELECT.  
d) Se crean con CREATE FUNCTION.

**8. ¿Cuál es el tipo de parámetro que solo recibe datos (sin devolver)?**

a) IN  
b) OUT  
c) INOUT  
d) RETURN

**✅ Soluciones (ocultas)**

Mostrar respuestas

**✅ Estructuras de programación en MySQL**

Al igual que en otros lenguajes de programación, MySQL permite controlar el flujo del código mediante **estructuras de control**: condiciones (IF, CASE) y bucles (WHILE, LOOP, REPEAT).

Estas estructuras se usan normalmente dentro de procedimientos y funciones.

**🔹 IF – Condicional simple**

Se utiliza para ejecutar código solo si se cumple una condición.

**🧱 Sintaxis**

IF condición THEN

-- código

END IF;

**🧪 Ejemplo**

DELIMITER //

CREATE PROCEDURE evaluar\_nota(nota INT)

BEGIN

IF nota >= 5 THEN

SELECT 'Aprobado' AS resultado;

END IF;

END //

DELIMITER ;

CALL evaluar\_nota(6);

**🧠 Ejercicios propuestos – IF**

1. Crea un procedimiento `evaluar\_edad` que reciba una edad y muestre "Mayor de edad" solo si es mayor o igual a 18, y que muestre "No es mayor de edad" si no lo es.

CREATE DEFINER=`root`@`%` PROCEDURE `evaluar\_edad`(edad int)

BEGIN

if edad >=18 then

select "Es mayor de edad" as mensaje;

else

select "No es mayor de edad" as mensaje;

end if;

END

1. Crea una función `es\_par` que reciba un número y devuelva TRUE si es par, y FALSE si no lo es.

CREATE DEFINER=`root`@`%` FUNCTION `es\_par`(num int) RETURNS tinyint(1)

DETERMINISTIC

BEGIN

declare resul boolean;

if num%2=0 then

set resul=true;

else

set resul=false;

end if;

RETURN resul;

END

**🔹 CASE – Condicional múltiple**

Permite evaluar varias condiciones posibles.

**🧱 Sintaxis**

CASE

WHEN condición1 THEN acción1

WHEN condición2 THEN acción2

ELSE acción\_por\_defecto

END CASE;

**🧪 Ejemplo**

DELIMITER //

CREATE PROCEDURE calificar\_nota(nota INT)

BEGIN

CASE

WHEN nota >= 9 THEN SELECT 'Sobresaliente';

WHEN nota >= 7 THEN SELECT 'Notable';

WHEN nota >= 5 THEN SELECT 'Aprobado';

ELSE SELECT 'Suspenso';

END CASE;

END //

DELIMITER ;

CALL calificar\_nota(6);

**🧠 Ejercicios propuestos – CASE**

1. Crea un procedimiento evaluar\_dia que reciba un número (1-7) y devuelva el nombre del día de la semana.

CREATE DEFINER=`root`@`%` PROCEDURE `evaluar\_dia`(dia int, out resul varchar(30))

BEGIN

CASE

WHEN dia = 1 THEN set resul='Lunes';

WHEN dia = 2 THEN set resul='Martes';

WHEN dia = 3 THEN set resul='Miercoles';

WHEN dia = 4 THEN set resul='Jueves';

WHEN dia = 5 THEN set resul='Viernes';

WHEN dia = 6 THEN set resul='Sabado';

WHEN dia = 7 THEN set resul='Domingo';

ELSE set resul="Dia erroneo";

END CASE;

END

1. Crea una función clasificar\_temperatura que reciba una temperatura y devuelva “Frío”, “Templado” o “Calor”. (Si es menos de 5 grados, Frio, si está entre 5 y 15, Templado y a partir de 15, Calor)

CREATE DEFINER=`root`@`%` FUNCTION `clasificar\_temperatura`(temp float) RETURNS varchar(30) CHARSET utf8mb4

DETERMINISTIC

BEGIN

declare resul varchar(30);

CASE

WHEN temp < 5 THEN set resul='Frio';

WHEN temp <=15 THEN set resul='Templado';

ELSE set resul='Caliente';

END CASE;

RETURN resul;

END

**🔁 WHILE – Bucle mientras se cumpla una condición**

Repite un bloque de código mientras la condición sea verdadera.

**🧱 Sintaxis**

WHILE condición DO

-- código

END WHILE;

**🧪 Ejemplo**

DELIMITER //

CREATE PROCEDURE contar\_hasta(d INT)

BEGIN

DECLARE i INT DEFAULT 1;

WHILE i <= d DO

SELECT i;

SET i = i + 1;

END WHILE;

END //

DELIMITER ;

CALL contar\_hasta(5);

En este ejemplo, queda un poco feo que cada vez que escribe un número, nos lo muestra en una pestaña diferente. Vamos a modificar el ejemplo para crear una tabla temporal con un solo campo de tipo entero, y vamos guardando el resultado de como se cuenta en el bucle dentro de la tabla. Al finalizar el procedimiento, hay que borrar la tabla.

```sql

DELIMITER //

CREATE PROCEDURE `contar\_hasta`(d INT)

BEGIN

   DECLARE i INT DEFAULT 1;

   create temporary table tablaContarHasta(num int);

   WHILE i <= d DO

      insert into tablaContarHasta values(i);

      SET i = i + 1;

   END WHILE;

select \* from tablaContarHasta;

END //

DELIMITER ;

CALL contar\_hasta(5);

```

**🧠 Ejercicios propuestos – WHILE**

1. Crea un procedimiento cuenta\_regresiva que reciba un número y cuente hacia atrás hasta 1. El resultado lo guarda en una tabla temporal y lo muestra al final.

CREATE DEFINER=`root`@`%` PROCEDURE `cuenta\_regresiva`(num int)

BEGIN

DECLARE i INT;

create temporary table tablaContarRegresiva(num int);

set i=num;

WHILE i > 0 DO

insert into tablaContarRegresiva values(i);

SET i = i - 1;

END WHILE;

select \* from tablaContarRegresiva;

drop table tablaContarRegresiva;

END

1. Crea un procedimiento pares\_hasta que reciba un número y guarde solo los números pares hasta ese número en una tabla que, despues de mostrar su contenido, borraremos.

CREATE DEFINER=`root`@`%` PROCEDURE `pares\_hasta`(num int)

BEGIN

declare i int default 2;

create table pares(numero int);

WHILE i <= num DO

insert into pares values(i);

SET i = i + 2;

END WHILE;

select \* from pares;

drop table pares;

END

**🔁 REPEAT – Ejecuta al menos una vez y repite mientras se cumpla una condición**

**🧱 Sintaxis**

REPEAT

-- código

UNTIL condición

END REPEAT;

**🧪 Ejemplo**

DELIMITER //

CREATE PROCEDURE ejemplo\_repeat()

BEGIN

DECLARE i INT DEFAULT 1;

REPEAT

SELECT i;

SET i = i + 1;

UNTIL i > 3

END REPEAT;

END //

DELIMITER ;

CALL ejemplo\_repeat();

**🧠 Ejercicios propuestos – REPEAT**

1. Crea un procedimiento mostrar\_impares que muestre los números impares del 1 al 9.

CREATE DEFINER=`root`@`%` PROCEDURE `mostrarimpares`()

BEGIN

declare i int default 1;

create table numimpares(num int);

REPEAT

insert into numimpares values(i);

SET i = i + 1;

UNTIL i > 9

END REPEAT;

select \* from numimpares;

drop table numimpares;

END

1. Crea un procedimiento contar\_hasta\_n que reciba un número y use REPEAT para contar hasta él. El resultado lo guardamos en una tabla que creamos dentro del procedimiento y borramos al final.

CREATE DEFINER=`root`@`%` PROCEDURE `contar\_hasta\_n`(in n int)

BEGIN

declare i int default 1;

create table numeros(num int);

REPEAT

insert into numeros values(i);

SET i = i + 1;

UNTIL i > n

END REPEAT;

select \* from numeros;

drop table numeros;

END

**📌 Nota**

Estas estructuras deben ir siempre dentro de bloques BEGIN...END, normalmente dentro de un procedimiento.

Recuerda: ¡es mejor practicar! Prueba cada estructura y modifica los ejemplos para entender cómo funcionan.

This page was generated by [GitHub Pages](https://pages.github.com).