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.

Un procedimiento almacenado siempre está incluido como un objeto más dentro de una base de datos, como si fuera una tabla o una vista.

Para realizar los ejemplos del tema que no están asociados a ninguna base de datos concreta, vamos a crear una base de datos denominada Tema8Ejemplos. Aquí crearemos los ejercicios del tema que NO están asociados a ninguna tabla de ninguna base de datos.

🖌 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
```

5. 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.
2. Crea un procedimiento llamado `cuadrado_numero` que reciba un número y muestre su cuadrado.
3. Crea un procedimiento llamado `info_usuario` que reciba un nombre y un apellido y muestre un mensaje de bienvenida personalizado. El nombre y el apellido debe escribirlo en mayúsculas y el mensaje debe ser como este ejemplo: Bienvenido NURIA CELIS a nuestra clase.
4. Crea un procedimiento llamado `mostrar_canciones` que muestre todas las canciones de un grupo a partir de su nombre. (Base de datos concursomusica).
5. 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. Hay que ignorar si hay error al insertar el voto.

6. Crea un procedimiento llamado `Aumentar_segundos` que aumenta en X segundos la duración de una canción y devuelve su nueva duración. Le pasamos como parámetro el título de la canción.
7. Crea un procedimiento llamado `Borrar_canciones` que elimina todas las canciones de un grupo dado por su nombre.
8. Crea un procedimiento llamado `Contar` que devuelva cuántos componentes tiene un grupo. El parámetro que pasamos al procedimiento es el nombre del grupo. El número de componentes lo devuelve como parámetro.
9. Crea un procedimiento llamado `grupos` que crea una tabla llamada resumen 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.

> 💡 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 DETERMINISTIC
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 DETERMINISTIC
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) DETERMINISTIC
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".
2. Crea una función llamada `area_circulo` que reciba el radio de un círculo y devuelva su área ($\pi * r^2$). Usa `PI()` y `POW()` de MySQL.
3. Crea una función llamada `iva_incluido` que reciba un precio y devuelva el precio con un 21% de IVA.
4. 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)
5. Crear una función llamada `nombre_grupo` que, dado el ID de un componente, devuelva el nombre del grupo al que pertenece.
6. Crear una función llamada `duracion_media` que reciba el nombre de un grupo y devuelva la duración media de sus canciones.
7. Crear una función llamada `votado` que, dado el ID de un usuario, devuelva cuantas canciones ha votado.
8. Crear una función llamada `componentes` que reciba el nombre de un grupo y devuelva cuántos componentes forman parte de él.

> 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

1. c
2. c
3. d
4. c
5. b
6. a
7. b
8. a

☑ 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.

2. Crea una función `es_par` que reciba un número y devuelva TRUE si es par, y FALSE si no lo es.

◇ 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.
2. 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)

🔁 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.

```
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;
drop table 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.
2. Crea un procedimiento `pares_hasta` que reciba un número y muestre solo los números pares hasta ese número.

🔁 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. Para ello se guardan los números en una tabla, se muestran y se borra al final la tabla.
2. Crea un procedimiento `contar_hasta_n` que reciba un número y use `REPEAT` para contar hasta él.

📌 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.