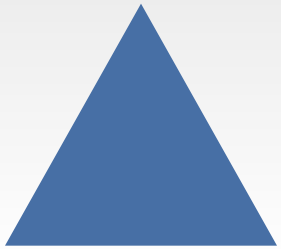
IES Valle del Jerte
PLASENCIA
Bilingual Section

TEMA 9: PROGRAMACIÓN EN BASES DE DATOS (I)

PROCEDIMIENTOS ALMACENADOS



UNION EUROPEA
Fondo Social Europeo
"Una manera de hacer Europa"

"**Una manera de hacer Europa**". Cofinanciación a cargo del Programa Operativo del FSE 2014-2020 para Extremadura gastos de Ciclos Formativos de Grados Medio y Superior.

Índice

INTRODUCCIÓN

- Hasta ahora hemos interactuado con la base de datos utilizando las sentencias del lenguaje SQL desde un cliente.
- Recordamos que el SQL es un lenguaje declarativo basado en el álgebra relacional. Este tipo de lenguaje presenta algunas limitaciones y es necesario utilizar un lenguaje más potente que resuelva ciertas limitaciones.
- Ejemplos de estos lenguajes son PL/SQL (Oracle, TRANSACT-SQL (SQL Server) y SPL (Mysql), que es el que utilizaremos nosotros.

INTRODUCCIÓN

Hoy en día casi todos los sistemas gestores tanto comerciales como libres cuentan con herramientas para la automatización. En la mayoría de los casos estas herramientas consisten en el uso de rutinas (procedimientos y funciones) almacenados, disparadores o triggers y eventos

En este tema comenzaremos a trabajar con procedimientos almacenados.

RUTINAS ALMACENADAS

- Las rutinas (procedimientos y funciones) son un conjunto de comandos SQL que se almacena en el servidor
- Una vez almacenados, los clientes no necesitan lanzar cada comando de forma individual sino que en su lugar llaman a la rutina como un único comando.
- MySQL soporta 3 tipos de rutinas:
  a. Procedimientos Almacenados,
  b. Funciones Almacenadas,
  c. Triggers o Disparadores.

UTILIDAD DE LAS RUTINAS ALMACENADAS

Utilizar rutinas almacenadas es útil:

1. 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.
2. Cuando la seguridad es muy importante.

RUTINAS ALMACENADAS: creación y estructura

- Se crean utilizando el comando **CREATE.**
- Tienen el siguiente esquema:
  - Nombre rutina + parámetros (entrada, salida o entrada/salida)
  - Declaración e inicialización de variables
  - Procesamiento de datos: Bloques BEGIN/END con instrucciones de control (condicionales y repetitivas)
  - Fin: Con la instrucción RETURN para devolver un valor en el caso de funciones almacenadas.

RUTINAS ALMACENADAS: creación y estructura

1. Por defecto, se crean en la base de datos actual.
2. Cuando se invocan, se realiza una operación **USE Nombre_BD implícito** (que se deshace cuando la rutina termina). La sentencia USE no está permitida dentro de las Rutinas almacenadas.
3. Si queremos crearlas en una BD distinta, deberemos poner delante de su nombre el nombre de la BD de destino.
4. Cuando se borra una DB, todas las Rutinas almacenadas asociadas, se borran también.

RUTINAS ALMACENADAS: PARÁMETROS

- Tanto los procedimientos como las funciones admiten parámetros.
- Los parámetros son variables que se envían y reciben en las rutinas.
- Se definen en la cláusula **CREATE** de creación del procedimiento de la forma:

CREATE PROCEDURE ([IN | OUT | INOUT] nombre_parámetro tipo de datos…])

La sintaxis obliga a poner una lista de parámetros aunque esté vacía () .

RUTINAS ALMACENADAS:TIPOS DE PARÁMETROS

En los procedimientos:

- **IN:** Parámetro de entrada. Es la opción por defecto, no hace falta especificarla.
- **OUT:** Salida.
- **INOUT:** Entrada/Salida.

En las funciones:

- sólo se utilizan parámetros **IN** (no es necesario especificarlo)
- La variable de retorno de la función se especifica con la **cláusula RETURNS**.

CAMBIO DE DELIMITADORES

Como ya sabemos, en MySQL, el caracter “;” (punto y coma), se utliza para terminar con una instrucción. Por ejemplo: SELECT * FROM CLIENTES;

Cuando escribimos un Procedimiento Almacenado que está formado por varias instrucciones separadas por ;, es necesario especificar un carácter distinto que delimite o separe un procedimiento del siguiente.

Para ello, al principio del procedimiento se define un nuevo delimitador (generalmente se usa $$ o //)para indicar que el procedimiento es un bloque que finaliza “del todo” en esa instrucción y no en el “;” que se ha encontrado al finalizar una instrucción.

CAMBIO DE DELIMITADORES

El nuevo delimitador se activa en la primera línea y en la última se vuelve a activar el delimitador por defecto (;)

```
DELIMITER //
CREATE PROCEDURE  NombreBD.Nombre(
parametro1 tipo_parametro)
BEGIN
……….
END; //
DELIMITER ;
```

CAMBIO DE DELIMITADORES. EJEMPO:

```
DELIMITER //
DROP PROCEDURE IF EXISTS ejemplo1_sp//
CREATE PROCEDURE ejemplo1_sp()
BEGIN
 select "esto es un ejemplo de procedimiento" as 'mensaje';
END //
```

El delimitador // indica que el procedimiento incluye todas las sentencias comprendidas entre los símbolos // y que todo se compilará en un único bloque que se almacenará con el nombre del procedimiento, para su posterior ejecución.

PROCEDIMIENTOS ALMACENADOS

PROCEDIMIENTOS: DEFINICIÓN

- Son las rutinas almacenadas más frecuentes.
- Sirven para resolver determinados problemas y pueden aceptar y devolver determinados parámetros.
- Los procedimientos NO devuelven un valor de manera explícita.
- Una vez compilado el código, se ejecutan mediante el comando **CALL**

PROCEDIMIENTOS: SINTAXIS

CREATE PROCEDURE nombre_procedimiento ([parámetro[,…]])

[características]

begin

    instrucciones del procedimiento

end;

PROCEDIMIENTOS: SINTAXIS

Dentro de la declaración de los procedimientos, el bloque opcional [características] se refiere a distintas configuraciones para la ejecución del procedimiento, que pueden tomar los siguientes valores:

- LANGUAGE SLQ | [NOT] DETERMINISTIC | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA } | SQL SECURITY { DEFINER | INVOKER }
- En las diapositivas siguientes veremos a qué se refieren estos valores, aunque generalmente, utilizaremos los valores por defecto.

PROCEDIMIENTOS: SINTAXIS

Aunque **solemos dejar los valores por defecto,** conviene conocer el significado de cada entrada:

- **LANGUAGE SQL:** Indica que el procedimiento está escrito en SQL
- **[NOT] DETERMINISTIC:** Una rutina se considera determinista si siempre produce el mismo resultado para los mismos parámetros de entrada, y no determinista en cualquier otro caso. Por defecto, se asume que un procedimiento es determinista.
- **CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA: CONTAINS SQL** indica que la rutina no contiene comandos que leen o escriben datos. **NO SQL** indica que la rutina no contiene comandos SQL.

PROCEDIMIENTOS: SINTAXIS

- **READS SQL DATA** indica que el procedimiento contiene comandos que leen datos, pero no comandos que escriben datos.
- **MODIFIES SQL DATA** indica que la rutina contiene comandos que pueden escribir datos. **CONTAINS SQL** es el valor por defecto si no se dan explícitamente ninguna de estas características, lo que permite al procedimiento lecturas y escrituras o modificaciones.
- **SQL SECURITY { DEFINER | INVOKER }:** Se utiliza para especificar si la rutina debe ser ejecutada usando los permisos del usuario que crea la rutina o el usuario que la invoca. El valor por defecto es **DEFINER.** Para poder ejecutar una rutina es necesario tener el permiso EXECUTE.

BORRADO DE PROCEDIMIENTOS . SENTENCIA DROP PROCEDURE

Para borrar un procedimiento de la BD, se utiliza el comando DROP PROCEDURE.

EJEMPLO:

```
DROP PROCEDURE EJEMPO1_SP;
```

VARIABLES

VARIABLES DEL SISTEMA

1. El servidor mantiene variables de sistema que indican cómo está configurado.
2. Todas ellas tienen valores por defecto.
3. Pueden cambiarse al arrancar el servidor usando opciones en la línea de comandos o modificando los ficheros de opciones.
4. La mayoría variables pueden cambiarse en tiempo de ejecución usando el comando SET.
5. Para cambiar una variable global, el usuario debe tener permisos suficientes.

VARIABLES DEL SISTEMA

1. El servidor mysqld mantiene dos clases de variables:
   a. Las variables globales, que afectan a las operaciones globales del servidor.
   b. Las variables de sesión que afectan a las operaciones para conexiones individuales de clientes.
2. Se pueden consultar las variables de sistema y sus valores usando el comando SHOW VARIABLES .
3. Enlace información VARIABLES DEL SISTEMA.

VARIABLES DEL SISTEMA

```
mysql> SHOW VARIABLES;

+---------------------------------+-----------------------------------------------------+
| Variable_name                   | Value                                               |
+---------------------------------+-----------------------------------------------------+
| auto_increment_increment        | 1                                                   |
| auto_increment_offset           | 1                                                   |
| automatic_sp_privileges         | ON                                                  |
| date_format                     | %Y-%m-%d                                            |
| datetime_format                 | %Y-%m-%d %H:%i:%s                                   |
| basedir                         | /                                                   |
```

VARIABLES DECLARADAS POR EL USUARIO Y FORMAS DE ASIGNAR VALORES A VARIABLES

VARIABLES GLOBALES DECLARADAS POR EL USUARIO

1. Su nombre comienza por el carácter @
2. Se le asigna un valor y pueden ser consultadas por una sentencia SELECT.
3. Aunque se le asigne valor dentro de un procedimiento, puede ser consultada fuera de él.

Ejemplo:

```
SET @MIVARIABLE="EJEMPLO DE VARIABLE GLOBAL"
SELECT @MIVARIABLE;
```

@MIVARIABLE
EJEMPLO DE VARIABLE GLOBAL

VARIABLES LOCALES DECLARADAS POR EL USUARIO

1. Se declaran para poder ser utilizadas dentro de una rutina almacenada (procedimiento, función…).
2. Guardan resultados obtenidos al ejecutar una consulta dentro del procedimiento o función.
3. Si no guardamos ese resultado en una variable, no podremos utilizarlo en comparaciones, sentencias de control, etc.
4. La visibilidad de una variable local se limita al bloque BEGIN ... END donde está declarado (no podemos referenciarla fuera del procedimiento o función).

VARIABLES LOCALES DECLARADAS POR EL USUARIO

1. Para declarar una variable se utiliza el comando DECLARE.
2. Sintaxis del comando:

   DECLARE nombre_var[,...] type [DEFAULT value]

   Donde:

   nombre_var es el nombre de la variable.

   type es el tipo de datos que va a guardar

   DEFAULT value: valor inicial de la variable por defecto.

ASIGNACIÓN DE VALORES A VARIABLES: 1ª FORMA: USO DE LA SENTENCIA SET

1. La sentencia SET sirve para asignar valores a una variable dentro de un procedimiento o función.
2. Las variables a las que se pueden asignar valores pueden ser declaradas dentro de una rutina, o ser variables de servidor o globales.
3. La sintaxis del comando de asignación es:

SET nombre_var= expr [nombre_var = expr] …

donde:

nombre_var: nombre de la variable

expr=valor/expresión que se asignará a la variable.

DECLARACIÓN DE VARIABLES Y USO DE LA SENTENCIA SET. EJEMPLO

```
2   DELIMITER //
3 • DROP PROCEDURE IF EXISTS VAR_EJEMPLO1//
4 • CREATE PROCEDURE VAR_EJEMPLO1()
5   BEGIN
6   DECLARE VARIABLE1 INT;
7   DECLARE VARIABLE2 VARCHAR(20);
8   SET VARIABLE1=25;
9   SET VARIABLE2="ESTO ES UNA PRUEBA";
10  select CONCAT_WS('******',VARIABLE1, VARIABLE2) AS RESULTADO;
11  END //
```

```
call VAR_EJEMPLO1();
```

RESULTADO
25******ESTO ES UNA PRUEBA

ASIGNACIÓN DE VALORES A VARIABLES: 2ª FORMA: USO DE LA SENTENCIA SELECT ... INTO

1. Se utiliza para almacenar el contenido de las columnas seleccionadas directamente en variables.

SELECT columna[,...] INTO nombre_var[,...] from tabla

SELECT id,nombre INTO x,y FROM tabla1 LIMIT 1;

2. Para poder utilizar esta sentencia, debemos saber que la consulta sólo devuelve un valor, que será que se almacene en la variable.
3. Si la consulta devuelve más de una fila se provocará un error:

USO DE LA SENTENCIA SELECT ... INTO. EJEMPLO1

```
/*PROCEDIMIENTO EJEMPLO DE CREACION DE VARIABLES */
CREATE PROCEDURE EJEMPLO_VAR1()
BEGIN
    DECLARE V_1 INT;
    DECLARE V_2 VARCHAR(20);
    SELECT ID,ALUMNO INTO V_1,V_2
    FROM ALUMNOS
    WHERE ID=4;
    SELECT V_1,V_2;
END //

DELIMITER ;
```

```
CALL EJEMPLO_VAR1();
```

V_1	V_2
4	ALUMNO4

La instrucción SELECT solo devuelve una fila.

LA SENTENCIA SELECT ... INTO. EJEMPLO2

```
DELIMITER //
/*PROCEDIMIENTO EJEMPLO DE CREACION DE VARIABLES  CUANDO LA CONSULTA DEVUELVE VARIAS FILAS*/
CREATE PROCEDURE EJEMPLO_VAR2()
BEGIN
    DECLARE V_1 INT;
    DECLARE V_2 VARCHAR(20);
    SELECT ID,ALUMNO INTO V_1,V_2
    FROM ALUMNOS;
    SELECT V_1,V_2;
END //
```

```
CALL EJEMPLO_VAR2();
```

Ahora la consulta devuelve más de una fila, se produce un error.

```
Error Code: 1172. Result consisted of more than one row
```

EJEMPLOS DE PROCEDIMIENTOS

Ejemplos de creación de Procedimientos

Vamos a crear una base de datos para practicar:

1. **Crear la base de datos VIDEOTECA:**

```
CREATE DATABASE videoteca;
USE videoteca;
```

2. Crear una tabla ACTOR

```
CREATE TABLE actor (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(64) NOT NULL,
  apellidos VARCHAR(64) NOT NULL,
  imdb VARCHAR(32) NOT NULL,
);
```

Ejemplos de creación de Procedimientos

3. **Inserta dos registros en la tabla anterior:**

```
INSERT INTO actor(nombre, apellidos, imdb)
VALUES('Harrison', 'Ford', 'nm0000148');
INSERT INTO actor(nombre, apellidos, imdb)
VALUES('Russell', 'Crowe', 'nm0000128');
```

Ejemplo 1.

```
DELIMITER //
DROP PROCEDURE IF EXISTS hola_mundo//
CREATE PROCEDURE hola_mundo()
BEGIN
    SELECT ‘hola mundo’;
END //
```

Este es un procedimiento muy simple cuyo único objeto es imprimir por pantalla la cadena 'hola mundo'. Vamos a describir el procedimiento:

- **DELIMITER** indica el carácter de comienzo y fin del procedimiento. Debería ser el típico ";" pero ya que necesitamos un ";" para cada sentencia SQL dentro del procedimiento, debemos usar otro carácter y normalmente se usa **$$ o //**.
- En la línea **DROP PROCEDURE IF EXISTS hola_mundo$$** eliminamos el procedimiento si es que ya existe. Es opcional, pero suele usarse y evita errores cuando queremos modificar un procedimiento ya existente.
- La línea **CREATE PROCEDURE hola_mundo()** indica el comienzo de la definición del procedimiento. Este procedimiento No tiene parámetros, pero hay que incluir () paréntesis vacíos.
- **BEGIN** indica el comienzo de las sentencias SQL que componen el cuerpo del procedimiento. En este caso sólo hay un SELECT que imprime la cadena de texto 'hola mundo' por pantalla.
- **END** indica el fin del bloque. Finaliza en **//** para indicar que hemos terminado.

Ejemplo 2.

Crear un procedimiento que lea todos los registros de la tabla ACTOR.

En este ejemplo utilizaremos un procedimiento almacenado para acceder a una tabla y mostrar su contenido. Es muy simple, ya que sólo tendrá dentro una sentencia SELECT.

```
DELIMITER //
DROP PROCEDURE IF EXISTS EJEMPLO2_SP//
CREATE PROCEDURE EJEMPLO2_SP()
BEGIN
    SELECT * FROM ACTOR;
END; //
```

Para ejecutar el procedimiento almacenado:

```
CALL EJEMPLO2_SP();
```

	id	nombre	apellidos	imdb
▶	1	Harrison	Ford	nm0000148
	2	Russell	Crowe	nm0000128

Ejemplo 3.

Crear un procedimiento que CUENTE el número de actores que hay en la tabla ACTOR.

```
DELIMITER //
DROP PROCEDURE IF EXISTS EJEMPLO3_SP//
CREATE PROCEDURE EJEMPLO3_SP()
BEGIN
    SELECT COUNT(*) AS “NUMERO DE ACTORES” FROM ACTOR;
END; //
```

```
CALL EJEMPLO3_SP();
```

NUMERO DE ACTORES
2

Ejemplo 4

El mismo procedimiento del ejemplo anterior, pero usando una variable para almacenar el resultado.

```
DELIMITER //
DROP PROCEDURE IF EXISTS EJEMPLO4_SP//
CREATE PROCEDURE EJEMPLO4_SP()
BEGIN
  DECLARE NUM_ACTORES INT;
  SELECT COUNT(*) INTO NUM_ACTORES FROM ACTOR;
   SELECT NUM_ACTORES;
END; //
```

```
CALL EJEMPLO4_SP();
```

NUM_ACTORES
2

Ejemplo 5

Procedimiento con parámetros. Escribir un procedimiento almacenado que muestra los actores cuyo nombre comienza con una determinada letra que le pasamos como parámetro:

```
DELIMITER //
DROP PROCEDURE IF EXISTS EJEMPLO5_SP//
CREATE PROCEDURE EJEMPLO5_SP(letra CHAR(1))
BEGIN
    SELECT * FROM actor WHERE nombre LIKE CONCAT(letra,'%');
END; //
```

Llamamos al procedimiento para que muestre los actores cuyo nombre empieza por H (utilizamos el comodín %)

```
CALL EJEMPLO5_SP('H');
```

id	nombre	apellidos	imdb
1	Harrison	Ford	nm0000148

Ejemplo 6

Procedimiento con 1 parámetro de entrada y otro de salida. Muestra los actores con el apellido que pasamos por parámetro (entrada) y cuenta cuántos hay (parámetro de salida)

```
DELIMITER //
DROP PROCEDURE IF EXISTS EJEMPLO6_SP//
CREATE PROCEDURE EJEMPLO6_SP(letra CHAR(2),OUT NUM_ACT INT)
BEGIN
    SELECT * FROM actor WHERE nombre LIKE letra;
    SELECT COUNT(*)  FROM ACTOR
    WHERE NOMBRE LIKE concat(LETRA,'%') INTO NUM_ACT;
END; //
DELIMITER ;
```

```
CALL EJEMPLO6_sp('H',@NUM_ACTORES);
```

id	nombre	apellidos	imdb
1	Harrison	Ford	nm0000148

El parámetro de salida debe ser una variable global para que pueda acceder a su contenido mediante una sentencia select.

```
SELECT @NUM_ACTORES;
```

@NUM_ACTORES
1

PROCEDIMIENTOS ACTIVOS

Procedimientos “Activos”:

Se llaman ACTIVOS a aquellos procedimientos que producen algún cambio en datos que contiene una tabla, es decir, qeu las sentencias del cuerpo del procedimiento incluyen:

- SENTENCIAS INSERT
- SENTENCIAS UPDATE
- SENTENCIAS DELETE

Procedimientos “Activos”: EJEMPLO7

Crear un procedimiento que inserta un nuevo registro en la tabla actor. El procedimiento tomará por parámetro el valor de los campos a insertar.

```
DELIMITER //
DROP PROCEDURE IF EXISTS EJEMPLO7_SP//
CREATE PROCEDURE EJEMPLO7_SP(nuevo_nombre VARCHAR(64), nuevo_apellidos VARCHAR(64),nuevo_imdb VARCHAR(32))
BEGIN
    INSERT INTO ACTOR VALUES (nuevo_nombre,nuevo_apellidos, nuevo_imdb);
  SELECT * FROM ACTOR;
END; //
```

```
CALL EJEMPLO7_SP('Tim','Robbins','nm0000209');
```

id	nombre	apellidos	imdb
1	Harrison	Ford	nm0000148
2	Russell	Crowe	nm0000128
3	Tim	Robbins	nm0000209

ESTRUCTURAS DE CONTROL DE FLUJO

ESTRUCTURAS DE CONTROL DE FLUJO EN PROCEDIMIENTOS Y FUNCIONES.

- Al igual que en cualquier lenguaje de programación, tanto en procedimientos como en funciones almacenadas, es frecuente tener que utilizar estructuras de control de flujo, ya sean instrucciones repetitivas o condicionales.
- En este apartado veremos las diferentes estructuras de control y su uso en los procedimientos almacenados.

ESTRUCTURAS CONDICIONALES

ESTRUCTURA CONDICIONAL.
EL COMANDO IF

- Funciona de manera similar que en otros lenguajes de programación.
- La acción se ejecuta cuando la condición es cierta.
- Si la condición no se cumple, se ejecuta la acción asociada a la sentencia ELSE.

Su estructura es:

```
IF condición1 THEN instrucciones
    [ELSEIF condición2 THEN instrucciones ....]
    [ELSE instrucciones]
END IF;
```

EL COMANDO IF. EJEMPLO 1

```
DELIMITER //
DROP PROCEDURE IF EXISTS CONDICIONAL//
create PROCEDURE CONDICIONAL(EDAD int)
BEGIN
    IF EDAD<=12 THEN SELECT 'NIÑO' AS "RESULTADO";
    END IF;
    IF EDAD>12 AND EDAD<=30 THEN SELECT 'JOVEN' AS "RESULTADO";
    end if;
    IF EDAD>30 THEN SELECT 'ADULTO' AS "RESULTADO";
    END IF;
   END //
```

```
CALL CONDICIONAL(10);
```

RESULTADO
NIÑO

ESTRUCTURA CONDICIONAL MÚLTIPLE: EL COMANDO CASE

1. Similar a la sentencia condicional IF anterior.
2. Todo lo que se puede expresar con la sentencia IF se puede expresar con la sentencia CASE.
3. Se suele utilizar cuando el número de expresiones o condiciones a evaluar es elevado y por tanto la lectura del código se hace más fácil y agradable.

EL COMANDO CASE

Existen 2 formas posibles si lo que se evalúa es una **EXPRESIÓN** o una **CONDICIÓN**:

```
CASE expresión
    WHEN valor1 THEN
        instrucciones
    [WHEN valor2 THEN
        instrucciones ...]
    [ELSE
        instrucciones]
END CASE;
```

```
CASE condición
    WHEN valor1 THEN
        instrucciones
    [WHEN valor2 THEN
        instrucciones ...]
    [ELSE
        instrucciones]
END CASE;
```

EL COMANDO CASE. EJEMPLO 1

```
create PROCEDURE CONDICIONAL3(EDAD int)
BEGIN
    CASE
     WHEN EDAD<=12 THEN
        SELECT 'NIÑO';
     WHEN EDAD>12 AND EDAD<=30 THEN
        SELECT 'JOVEN';
    ELSE
        select 'ADULTO';
    END CASE;
   END //
DELIMITER ;
CALL CONDICIONAL3 (50);
```

ADULTO
ADULTO

EL COMANDO CASE. EJEMPLO 2

```
DELIMITER //
CREATE PROCEDURE CONDICIONAL4 ()
BEGIN
DECLARE PAGO ENUM ('TARJETA', 'METALICO','APLAZADO');
SET PAGO='TARJETA';
CASE PAGO
    WHEN 'TARJETA' THEN
        SELECT 'FORMA DE PAGO ELEGIDA: TARJETA';
    WHEN 'METALICO' THEN
        SELECT 'FORMA DE PAGO ELEGIDA: METALICO';
    ELSE
        SELECT 'FORMA DE PAGO ELEGIDA: APLAZADO';
    END CASE;
END //
```

ESTRUCTURAS REPETITIVAS

EL COMANDO LOOP, LEAVE E ITERATE

Sintaxis:

```
[etiqueta:] LOOP
                instrucciones
END LOOP [etiqueta];
```

Todas las instrucciones comprendidas entre las palabras reservadas LOOP Y END LOOP (bucle), se ejecutan un número de veces hasta que la ejecución del bucle se encuentra con la instrucción:

```
LEAVE etiqueta
```

En ese momento se abandona el bucle.

SENTENCIA ITERATE.

SINTAXIS:

```
ITERATE etiqueta
```

Se utiliza para forzar que la ejecución del bucle termine en el momento en que se encuentra la instrucción y continúa por el principio del bucle.

EL COMANDO LOOP, LEAVE E ITERATE.- EJEMPLO 1

En este ejemplo, el bucle se repite 4 veces.

```
DELIMITER //;
CREATE PROCEDURE BUCLE1 ()
BEGIN
DECLARE I INT;
SET I=0;
MIBUCLE: LOOP
    SET I=I+1;
    SELECT 'VALOR DE I='+I AS I;
    IF I=4 THEN
        LEAVE MIBUCLE;
    END IF;
END LOOP MIBUCLE;
END //;
DELIMITER ;
CALL BUCLE1();
```

EL COMANDO LOOP, LEAVE E ITERATE.- EJEMPLO 2

En este ejemplo, la instrucción SELECT se ejecuta 3 veces, para los valores de i desde 1 a 4 excepto el 3.

```
DELIMITER //;
CREATE PROCEDURE BUCLE2 ()
BEGIN
DECLARE I INT;
SET I=0;
MIBUCLE: LOOP
    SET I=I+1;
    IF I=3 THEN
        ITERATE MIBUCLE;
    END IF;
    SELECT 'VALOR DE I='+I AS I;
    IF I=4 THEN
        LEAVE MIBUCLE;
    END IF;
END LOOP MIBUCLE;
END //;
DELIMITER ;
```

EL COMANDO REPEAT … UNTIL

Sintaxis:

```
[etiqueta:] REPEAT
      instrucciones
       UNTIL expresión
END REPEAT [etiqueta]
```

- Las instrucciones se ejecutarán HASTA que sea cierta la expresión.
- El conjunto de instrucciones se ejecuta al menos una vez.
- La evaluación de la expresión se hace DESPUÉS de la ejecución de las instrucciones.

EL COMANDO REPEAT … UNTIL. EJEMPLO

```
DELIMITER //
DROP PROCEDURE IF EXISTS BUCLE3//
CREATE PROCEDURE BUCLE3()
BEGIN
    DECLARE I INT;
    SET I=0;
    REPEAT
        SET I=I+1;
        UNTIL I=4
    END REPEAT;
    SELECT CONCAT ('EL VALOR DE I ES:      ',I) AS I;
END; //
DELIMITER ;
```

I
EL VALOR DE I ES: 4

EL COMANDO WHILE

Sintaxis:

```
[etiqueta:] WHILE EXPRESION DO
    instrucciones
END WHILE [etiqueta]
```

Se ejecuta el conjunto de instrucciones MIENTRAS sea cierta la expresión.

EL COMANDO WHILE. EJEMPLO

```
DELIMITER //
DROP PROCEDURE IF EXISTS BUCLE4//
CREATE PROCEDURE BUCLE4()
BEGIN
    DECLARE I INT;
    SET I=0;
    WHILE I<4 DO
        SET I=I+1;
        SELECT CONCAT ('EL VALOR DE I ES:      ',I) AS I;
    END WHILE;
END; //
DELIMITER ;
CALL BUCLE4();
```

I
EL VALOR DE I ES: 4