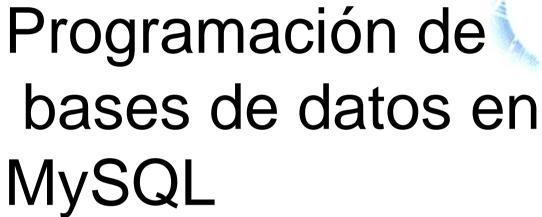


## **Tema**





**Bases** 

De

**Datos** 

# Índice



- 1. Introducción a la programación de BD
- 2. Sintaxis y ejemplos de rutinas
- 3. Instrucciones Condicionales
- 4. Instrucciones Repetitivas
- 5. SQL en rutinas: Cursores
- 6. TRIGGERS



# 1.- Introducción a la programación de

Las BD incorporan lenguajes propios que permiten integrar datos y funcionalidad dentro de la BD.

### Ventajas:

- Independencia del S.O.
- Aplicaciones más ligeras.
- Facilidad de mantenimiento.

MYSQL: Permite la definición de rutinas, disparadores, vistas y eventos mediante un lenguaje de programación sin nombre específico.

## Procedimientos y funciones almacenadas en MYSQL

Las rutinas almacenadas son un conjunto de comandos SQL que pueden guardarse en el servidor y así los clientes pueden llamarlas.

 Mejora del rendimiento al ser ejecutadas por el Servidor.

<u>Funciones</u>: Devuelven un único valor y pueden ser incluidad en otras funciones o procedimientos así como en sentencias SQL.

<u>Procedimientos</u>: Pueden devolver valores a través de parámetros OUT o INOUT.

# Florida

# 2.- Sintaxis y ejemplos de rutinas

### Esquema general rutina almacenada en la BD

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

**Bases de Datos** 

Para desarrollar las rutinas usaremos cualquier editor de textos o mejo la GUI de MySQL Workbench.

### Sintaxis general:

```
CREATE PROCEDURE sp_name ([parameter[,...]])
[characteristic ...] routine_body
CREATE FUNCTION sp_name ([parameter[,...]])
RETURNS type
[characteristic ...] routine_body
parameter:
[ IN | OUT | INOUT ] param_name type
type:
Any valid MySQL data type
characteristic:
LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }
```

*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 (IN), salida (OUT) o entrada/salida (INOUT).

routine\_body: es el cuerpo de la rutina formado generalmente por sentencias SQL. En caso de haber más de una deben ir dentro de un bloque delimitado por sentencias BEGIN y END como veremos en los siguientes ejemplos.

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

Contains SQL/no SQL: 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 debe ejecutarse con permisos del creador (definer) o del que lo invoca (invoker).



#### Resumen:

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

Ejemplos sobre MySQLWorkbench.



#### Parámetros y Variables

#### Manejo de Variables:

DECLARE. Crea una variable de tipo char, varchar, int, float, etc...

SET. Asigna valor a las variables

#### Tipos de parámetros:

IN . Tipo por defecto para incluir parámetros de entrada

OUT. Tipo de parámetro de salida

INOUT. Permite pasar valores a la rutina y ser modificados para su devolución.

Ejemplos sobre MySQLWorkbench

#### EJERCICIOS,

# 3.- Intrucciones Condicionales



#### **IN-THEN-ELSE**

```
IF expr1 THEN
...
ELSEIF expr2 THEN
...
ELSE
...
END IF
```

#### **CASE**

```
CASE Expression

WHEN value THEN

statements

[WHEN value THEN

statements ...]

[ELSE

statements]

END CASE;
```





#### SIMPLE LOOP

#### REPEAT UNTIL LOOP

```
[etiqueta:] REPEAT
instrucciones
UNTIL expresion
END REPEAT [etiqueta];
```

**Bases de Datos** 



# 4.- Instrucciones Repetitivas

#### WHILE LOOP

```
[etiqueta:] WHILE Expression DO
instrucciones
END WHILE [etiqueta];
```

Ejemplos sobre MySQLWorkbench

EJERCICIOS,

Dentro de las rutinas de programación pueden incluirse sentencias SQL tanto de DDL, DML o DCL.

Cuando se quiere recuperar más de una fila para manipular sus datos se requiere el uso de cursores.

Para enviar valores a variables desde SELECT se usa lo siguiente:

SELECT expression1 [expression2.....]
INTO variable1, [variable2....]
otras instrucciones SELECT

**CURSOR** Conjunto de filas o consulta sobre una tabla de una Base de Datos *Sintaxis*:

DECLARE cursor\_name CURSOR FOR SELECT\_statement;

#### Comandos para la manipulación de sectores

OPEN: inicializa el conjunto de resultados asociados con el cursor.

```
OPEN cursor_name
```

FETCH: extrae la siguiente fila de valores del conjunto de resultados del cursor moviendo su puntero interno una posición.

```
FETCH cursor name INTO variable list;
```

CLOSE: cierra el cursor liberando la memoria que ocupa y haciendo imposible el acceso a cualquiera de sus datos.

```
CLOSE cursor_name ;
```

**Bases de Datos** 

# Florida

## 5.- SQL en rutinas: Cursores

Para detectar la última fila del cursor y no avanzar más usaremos un manejador de errores (handler).

DECLARE CONTINUE HANDLER FOR NOT FOUND SET lrf=1;

- Establece la variable última fila lrf (last\_row\_fetched) = 1
- Permitirá al programa continuar con la ejecución.



#### Gestión de rutinas almacenadas

Las rutinas se manipulan con los comandos CREATE (ya visto), DROP y SHOW Eliminación:

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp name
```

#### Consulta:

```
SHOW CREATE {PROCEDURE | FUNCTION} sp_name
SHOW {PROCEDURE | FUNCTION} STATUS [LIKE 'pattern']
```

#### Manejo de ERRORES

Cuando un programa produce un error la ejecución se detiene y se devuelve un error a la aplicación llamante. Sin embargo, podemos definir otro comportamiento gracias los controladores de excepciones.

#### SINTAXIS DEL MANEJADOR

```
DECLARE {CONTINUE | EXIT} HANDLER FOR 
{SQLSTATE sqlstate_code| MySQL error code| condition_name} 
handler_actions
```

- Tipo de manejador: EXIT o CONTINUE
- Condición del manejador: estado SQL (SQLSTATE), error propio de MySQL o código de error definido por el usuario.
- Acciones del manejador: acciones a tomar cuando se active el manejador.

Ejemplos sobre MySQLWorkbench.

**EXIT**: Cuando se encuentra el error el bloque que se está ejecutando actualmente termina.

Es más adecuado para errores catastróficos ya que no permite la continuidad

**CONTINUE**: La ejecución continúa en la declaración siguiente a la que ocasionó el error.

Es adecuado cuando existe algún procesamiento alternativo ante el error.

Los códigos de error están definidos en el manual de referencia. Ej. "1062" = "duplicate entry for PRIMARY KEY"

Ejemplos sobre MySQLWorkbench.

## 6.- TRIGGERS

**Trigger** o DISPARADOR es un tipo especial de rutina almacenada que se activa cuando en una tabla ocurre un evento del tipo INSERT, DELETE o UPDATE

#### **CREAR TRIGGER:**

CREATE TRIGGER nombre\_disp momento\_disp evento\_disp ON nombre tabla FOR EACH ROW sentencia disp

#### Donde:

- momento\_disp: es el momento en que el disparador entra en acción. Puede ser BEFORE (antes) o AFTER (después), para indicar que el disparador se ejecute antes o después que la sentencia que lo activa.
- evento\_disp: indica la clase de sentencia que activa al disparador. Puede ser INSERT, UPDATE, o DELETE. Por ejemplo, un disparador BEFORE para sentencias INSERT podría utilizarse para validar los valores a insertar.
  - No puede haber dos disparadores en una misma tabla que correspondan al mismo momento y sentencia. Por ejemplo, no se pueden tener dos disparadores  $BEFORE\ UPDATE$ . Pero sí es posible tener los disparadores  $BEFORE\ UPDATE$  y  $BEFORE\ INSERT$  o  $BEFORE\ UPDATE$  y  $AFTER\ UPDATE$ .
- FOR EACH ROW: hace referencia a las acciones a llevar a cabo sobre cada fila de la tabla indicada.
- Sentencia\_disp.

## 6.- TRIGGERS



#### **ELIMINAR TRIGGER:**

```
DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name
```

#### **CONSULTA TRIGGER:**

```
SHOW TRIGGERS [{FROM | IN} db_name]

[LIKE 'pattern' | WHERE expr]
```