



# **BASE DE DATOS II**

---

Manejo de triggers

ESTUDIANTE : Elvin Braxail Cussi Aranibar



# **01** | Manejo de Conceptos

# 1. Defina que es lenguaje procedural en MySQL.

El lenguaje procedural en MySQL se refiere a la capacidad de MySQL de admitir la programación procedural mediante la incorporación de funciones y procedimientos almacenados en su sintaxis. Esto permite a los desarrolladores escribir bloques de código más complejos y estructurados que pueden ser invocados y reutilizados en diferentes partes de una base de datos.



## 2. Defina que es una FUNCTION en MySQL.

las funciones se definen mediante la creación de una sentencia CREATE FUNCTION. Esta sentencia especifica el nombre de la función, los parámetros de entrada, el tipo de dato del resultado y el bloque de código que se ejecuta cuando la función es invocada. El bloque de código puede contener instrucciones SQL y lógica procedural para realizar la operación deseada.

```
CREATE FUNCTION calendario (fecha DATE)
RETURNS TEXT
DETERMINISTIC
BEGIN
    DECLARE a TEXT DEFAULT '';
    CASE
        WHEN weekday = 'Monday' THEN SET a = 'Lunes';
        WHEN weekday = 'Tuesday' THEN SET a = 'Martes';
        WHEN weekday = 'Wednesday' THEN SET a = 'Miércoles';
        WHEN weekday = 'Thursday' THEN SET a = 'Jueves';
        WHEN weekday = 'Friday' THEN SET a = 'Viernes';
        WHEN weekday = 'Saturday' THEN SET a = 'Sábado';
        WHEN weekday = 'Sunday' THEN SET a = 'Domingo';
    END
    SET a = ' día de la semana correspondiente';
    RETURN a;
END;

SELECT calendario('2023-01-01');
```



### 3.Cuál es la diferencia entre funciones y procedimientos almacenados.



Retorno de valor: Una función siempre devuelve un valor como resultado, mientras que un procedimiento almacenado no necesariamente devuelve un valor.

Uso en consultas: Las funciones se pueden utilizar en consultas SQL como parte de una expresión, ya sea en una cláusula SELECT, WHERE, JOIN, etc. Por otro lado, los procedimientos almacenados no se pueden utilizar directamente en una consulta, ya que están diseñados para ejecutar instrucciones y no devuelven un resultado que se pueda utilizar en una consulta.

## 4. Cómo se ejecuta una función y un procedimiento almacenado.

Debes reemplazar "nombre\_funcion" con el nombre de la función que deseas ejecutar y proporcionar los valores de los parámetros correspondientes. El resultado devuelto por la función se utilizará en la consulta o expresión donde se realiza la invocación.

```
select calendario2( tipoDia: 'Monday')
```

## 5. Defina que es una TRIGGER en MySQL.

Un trigger se define asociado a una tabla y se dispara antes o después de que se realice una operación específica (como INSERT, UPDATE o DELETE) en esa tabla. Cuando el evento asociado se produce, el trigger se activa y ejecuta un conjunto de instrucciones SQL o un procedimiento almacenado definido por el usuario.

```
1 CREATE OR REPLACE TRIGGER arborfallo
2 BEFORE INSERT
3 ON proyecto
4 FOR EACH ROW
5 BEGIN
6     DECLARE tipodia text default dayname(current_date);
7     DECLARE tipomes text default monthname(current_date);
8
9     IF (new.id_proy = 'Extraccion' and tipodia = 'Wednesday' and tipomes = 'JUNE') then
10         signal sqlstate '45000';
11         set message_text = 'no se puede insertar datos';
12
13     END IF;
14 END;
```

En un trigger de MySQL, las variables OLD y NEW son variables especiales que se utilizan para acceder a los valores antiguos y nuevos de las filas afectadas por un evento (INSERT, UPDATE o DELETE). Estas variables se utilizan principalmente en triggers AFTER UPDATE, BEFORE UPDATE, AFTER DELETE y BEFORE INSERT.

OLD hace referencia a los valores antiguos de la fila que está siendo modificada o eliminada. En un trigger AFTER UPDATE, por ejemplo, OLD permite acceder a los valores originales de las columnas antes de la actualización. En un trigger AFTER DELETE, OLD proporciona los valores de la fila eliminada.

NEW hace referencia a los nuevos valores de la fila que está siendo modificada o insertada. En un trigger AFTER UPDATE, NEW permite acceder a los nuevos valores de las columnas después de la actualización. En un trigger AFTER INSERT, NEW proporciona los valores de la fila recién insertada.

## **6. En un trigger que papel juega las variables OLD y NEW**



## **7. En un trigger que papel juega los conceptos(cláusulas) BEFORE o AFTER**

las cláusulas BEFORE y AFTER determinan cuándo se activará el trigger en relación con el evento que lo dispara (INSERT, UPDATE o DELETE).

**BEFORE:** Un trigger definido con la cláusula BEFORE se activa antes de que se realice la operación en la tabla. Esto significa que el trigger se ejecutará antes de que los datos se inserten, actualicen o eliminen de la tabla.

**AFTER:** Un trigger definido con la cláusula AFTER se activa después de que se haya realizado la operación en la tabla. Esto significa que el trigger se ejecutará después de que los datos se hayan insertado, actualizado o eliminado de la tabla.



# **02** | **PARTE PRACTICA**

# 9. Crear la siguiente Base de datos y sus registros.



```
CREATE DATABASE db;
USE db;

CREATE TABLE departamento (
    id_dep INT(11) PRIMARY KEY,
    nombre VARCHAR(50)
);

CREATE TABLE provincia (
    id_prov INT(11) PRIMARY KEY,
    nombre VARCHAR(50),
    id_dep INT(11) FOREIGN KEY REFERENCES departamento(id_dep)
);

CREATE TABLE proyecto (
    id_proy INT(11) PRIMARY KEY,
    nombre_proy VARCHAR(100),
    tipoProy VARCHAR(20)
);

CREATE TABLE persona (
    id_per INT(11) PRIMARY KEY,
    nombre VARCHAR(20),
    apellidos VARCHAR(20),
    fecha_nac DATE,
    edad INT,
    email VARCHAR(50),
    id_dep INT(11) FOREIGN KEY REFERENCES departamento(id_dep),
    id_prov INT(11) FOREIGN KEY REFERENCES provincia(id_prov),
    id_proy INT(11) FOREIGN KEY REFERENCES proyecto(id_proy),
    sexo CHAR(1)
);

CREATE TABLE detalle_proyecto (
    id_dp INT(11) PRIMARY KEY,
    id_per INT(11) FOREIGN KEY REFERENCES persona(id_per),
    id_proy INT(11) FOREIGN KEY REFERENCES proyecto(id_proy)
);
```

## Crear Una función que sume los valores de la serie Fibonacci.

- El objetivo es sumar todos los números de la serie fibonacci desde una cadena.
- Es decir usted tendrá solo la cadena generada con los primeros N números de la serie fibonacci y a partir de ellos deberá sumar los números de esa serie.
- Ejemplo:  
suma\_serie\_fibonacci(mi\_metodo\_que\_retorna\_la\_serie(10)) ■ Note que previamente deberá crear una función que retorne una cadena con la serie Fibonacci hasta un cierto valor. 1.  
Ejemplo: 0,1,1,2,3,5,8,..... ■ Luego esta función se deberá pasar como parámetro a la función que suma todos los valores de esa serie generada.

```
create or replace function suma_serie_fibonacci (cadena varchar2)
return varchar2
as
begin
    declare n varchar2(10);
    declare p varchar2(10);
    declare q varchar2(10);
    declare r varchar2(10);
    declare respuesta varchar2(100);

    -- Inicialización
    n := '0';
    p := '1';
    q := '1';
    r := '2';

    -- Suma de la serie
    while (length(n) < 10)
    loop
        respuesta := respuesta || n || ',';
        n := p;
        p := q;
        q := r;
        r := n + p;
    end loop;

    -- Retorno
    return respuesta;
end;
```

```
create or replace function suma_serie_fibonacci (cadena varchar2)
return varchar2
as
begin
    declare n varchar2(10);
    declare p varchar2(10);
    declare q varchar2(10);
    declare r varchar2(10);
    declare respuesta varchar2(100);

    -- Inicialización
    n := '0';
    p := '1';
    q := '1';
    r := '2';

    -- Suma de la serie
    while (length(n) < 10)
    loop
        respuesta := respuesta || n || ',';
        n := p;
        p := q;
        q := r;
        r := n + p;
    end loop;

    -- Retorno
    return respuesta;
end;
```

```
1 0,1,1,2,3,5,8,
```

```
suma_serie_fibonacci(7) = 28
```



## 11. Manejo de vistas.

○ Crear una consulta SQL para lo siguiente.

■ La consulta de la vista debe reflejar como campos:

1. nombres y apellidos concatenados
2. la edad
3. fecha de nacimiento.
4. Nombre del proyecto

○ Obtener todas las personas del sexo femenino que hayan nacido en el departamento de El Alto en donde la fecha de nacimiento sea:

1. fecha\_nac = '2000-10-10'

```
create or replace view IdentificentoPersona as
select concat(per.nombre, ' ', per.apellidos), per.edad, per.fecah_nac, p.nombreProy from persona per
join departamento d on per.id_dep = d.id_dep
join detalle_proyecto dp on per.id_per = dp.id_per
join proyecto p on dp.id_proy = p.id_proy
where d.nombre = 'LaPaz' and per.fecah_nac = '2000-10-10' and per.sexo = 'F';
```

'concat(per.nombre, ' ', per.apellidos)'	edad	fecah_nac	nombreProy
1 Carolina Alanoca Paudara	23	2000-10-10	Conce y Aprende



- Crear TRIGGERS Before or After para INSERT y UPDATE aplicado a la tabla PROYECTO

- Debera de crear 2 triggers minimamente.
- Agregar un nuevo campo a la tabla PROYECTO.
  - El campo debe llamarse ESTADO 6
- Actualmente solo se tiene habilitados ciertos tipos de proyectos.
- EDUCACION, FORESTACION y CULTURA
- Si al hacer insert o update en el campo tipoProy llega los valores EDUCACION, FORESTACIÓN o CULTURA, en el campo ESTADO colocar el valor ACTIVO. Sin embargo se llegat un tipo de proyecto distinto colocar INACTIVO

```
alter table proyecto add column Estado varchar(15);  
  
create or replace trigger Activoinactivo  
before insert  
on proyecto  
for each row  
begin  
    if new.tipoProy in ('Educacion','Forestacion','cultura') then  
        set new.Estado = 'Activo';  
    else  
        set new.Estado='inactivo';  
    end if;  
end;
```

```
create or replace trigger Activoinactivo_update  
before update  
on proyecto  
for each row  
begin  
    if new.tipoProy in ('Educacion','Forestacion','cultura') then  
        set new.Estado = 'Activo';  
    else  
        set new.Estado='inactivo';  
    end if;  
end;
```

```
insert into proyecto (nombreProy, tipoProy) VALUES ('Cortar Arboles','Forestacion');  
insert into proyecto (nombreProy, tipoProy) VALUES ('Cortar Arboles','venta');  
select * from proyecto;  
update proyecto set tipoProy = 'Educacion' where id_proy =2;  
update proyecto set tipoProy = 'Claro' where id_proy =1;  
select * from proyecto;
```

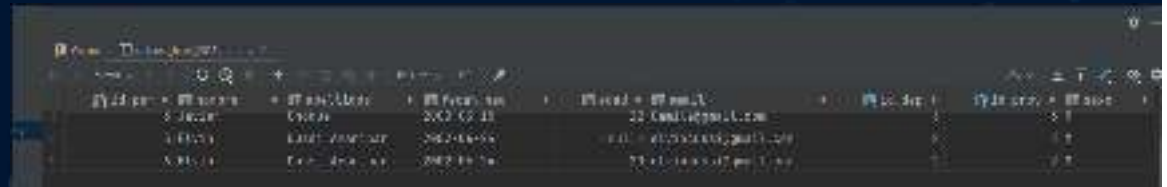
## 12.Manejo de TRIGGERS

- El trigger debe de llamarse **calculaEdad**.
- El evento debe de ejecutarse en un **BEFORE INSERT**.
- Cada vez que se inserta un registro en la tabla **PERSONA**, el trigger debe de calcular la edad en función a la fecha de nacimiento. ○ Adjuntar el código SQL generado y una imagen de su correcto funcionamiento.

```
create or replace trigger calculaEdad
before insert
on persona
for each row
begin
set new.edad:= timestampdiff(year, new.fecha_nac, curdate());
end;

insert into persona(id_per, nombre, apellidos, fecha_nac, email, id_dep, id_prov, sexo)
values (6,'Civici', 'Cusi, Alexander', '2002-06-26', 'alexvincos@gmail.com', 2,2, 'M');

select * from persona;
```



ID per	Nombre	Apellidos	Fecha_nac	Email	ID dep	ID prov	Sexo
6	Civici	Cusi, Alexander	2002-06-26	alexvincos@gmail.com	2	2	M

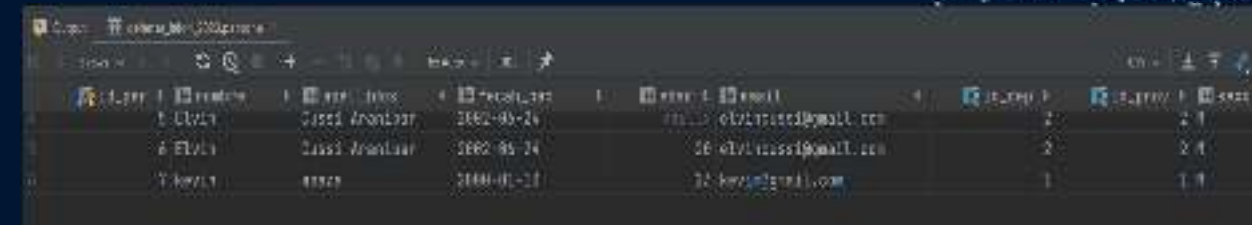
## 13. Manejo de Triggers II

## 14. Manejo de TRIGGERS III.

- Crear otra tabla con los mismos campos de la tabla persona (Excepto el primary key id\_per).
    - No es necesario que tenga PRIMARY KEY.
  - Cada vez que se haga un INSERT a la tabla persona estos mismos valores deben insertarse a la tabla copia.
  - Para resolver esto deberá de crear un trigger before insert para la tabla PERSONA.
- Adjuntar el código SQL generado y una imagen de su correcto funcionamiento.

```
create table persona_copia(  
  nombre varchar(20),  
  apellidos varchar(50),  
  fecha_nac date,  
  edad int,  
  email varchar(50),  
  id_dep int,  
  id_prov int,  
  sexo char(1)  
);
```

```
create or replace trigger copia_persona  
before insert  
on persona  
for each row  
begin  
  insert into persona_copia  
  (nombre, apellidos, fecha_nac, edad, email, id_dep, id_prov, sexo)  
  values  
  (new.nombre, new.apellidos, new.fecha_nac, new.edad, new.email, new.id_dep, new.id_prov, new.sexo);  
end;
```



id_per	nombre	apellidos	fecha_nac	edad	email	id_dep	id_prov	sexo
1	LUIS	José Antonio	1982-05-24	36	olivia123@gmail.com	2	2	M
2	LUIS	José Antonio	1982-05-24	36	olivia123@gmail.com	2	2	M
3	Kevin	apaza	2008-01-11	15	12-kevin@gmail.com	1	1	M



nombre	apellidos	fecha_nac	edad	email	id_dep	id_prov	text
Kevin	apaza	2008-01-11	15	12-kevin@gmail.com	1	1	M

## 15. Crear una consulta SQL que haga uso de todas las tablas.

```
create or replace view tablas_en_general as
select concat(per.nombre, ' ', per.apellidos), per.edad, per.fecha_nac,
p.nombreProy as Proyecto, p.tipoProy,
prov.nombre as provincia, d.nombre as departamento, dp.id_proy
from persone as per
join departamento d on per.id_dep = d.id_dep
join detalle_proyecto dp on per.id_per = dp.id_per
join proyecto p on dp.id_proy = p.id_proy
join provincia prov on per.id_prov = prov.id_prov;

select * from tablas_en_general;
```

ADIOS