DEFENSA
HITO 4
PROCESUAL

BASE DE DATOS II

DATOS PERSONALES

01

NOMBRE COMPLETO

CHOQUE AMARO MIJAIL OLIVER
SIS12955851

02

CARRERA

INGIENERIA EN SISTEMAS

03

DOCENTE

WILLIAM BARRA PAREDES

04

FECHA DE ENTREGA

2023/06/07

01

MANEJO DE CONCEPTOS

MANEJO DE CONCEPTOS

Lenguaje procedural en MySQL

El lenguaje procedural en MySQL se refiere a la capacidad de escribir y ejecutar bloques de código o instrucciones en un orden secuencial. Permite crear procedimientos almacenados, funciones y triggers en MySQL utilizando sentencias SQL y estructuras de control como bucles y condicionales.

1.1

MANEJO DE CONCEPTOS

Función en MySQL

Una función en MySQL es un bloque de código que acepta parámetros de entrada, realiza cálculos o manipulaciones en la base de datos y devuelve un valor como resultado. Pueden ser utilizadas en consultas SQL para realizar operaciones específicas y simplificar la lógica del código.

1.2

MANEJO DE CONCEPTOS

Diferencia entre funciones y procedimientos almacenados

La principal diferencia entre funciones y procedimientos almacenados en MySQL radica en su uso y comportamiento. Las funciones se utilizan para realizar cálculos y devolver un valor, mientras que los procedimientos almacenados son más adecuados para realizar operaciones que no necesariamente devuelven un resultado y pueden tener efectos secundarios como modificaciones en la base de datos.

1.3

MANEJO DE CONCEPTOS

Ejecución de una función y un procedimiento almacenado

Para ejecutar una función en MySQL, se puede utilizar la sintaxis de una consulta SQL, pasando los parámetros de entrada necesarios y capturando el valor devuelto por la función. Para ejecutar un procedimiento almacenado, se utiliza la sentencia "CALL" seguida del nombre del procedimiento y los parámetros requeridos

1.4

MANEJO DE CONCEPTOS

Trigger en MySQL

Un trigger en MySQL es un objeto de base de datos que se activa automáticamente en respuesta a un evento específico, como la inserción, actualización o eliminación de datos en una tabla. Los triggers permiten ejecutar código SQL personalizado antes o después de que ocurra el evento y son útiles para mantener la integridad de los datos y automatizar ciertas acciones.

1.5

MANEJO DE CONCEPTOS

Variables OLD y NEW en un trigger

En un trigger, las variables OLD y NEW se refieren a los valores antiguos y nuevos de las filas afectadas por el evento que activa el trigger. La variable OLD contiene los valores antes de que se realice la operación y la variable NEW contiene los valores actualizados o nuevos que se van a utilizar.

1.6

MANEJO DE CONCEPTOS

Cláusulas BEFORE y AFTER en un trigger

Las cláusulas BEFORE y AFTER en un trigger indican cuándo se ejecutará el código del trigger en relación con la operación que lo activa. La cláusula BEFORE se ejecuta antes de que se realice la operación y la cláusula AFTER se ejecuta después de que se complete la operación.

1.7

MANEJO DE CONCEPTOS

Eventos en triggers

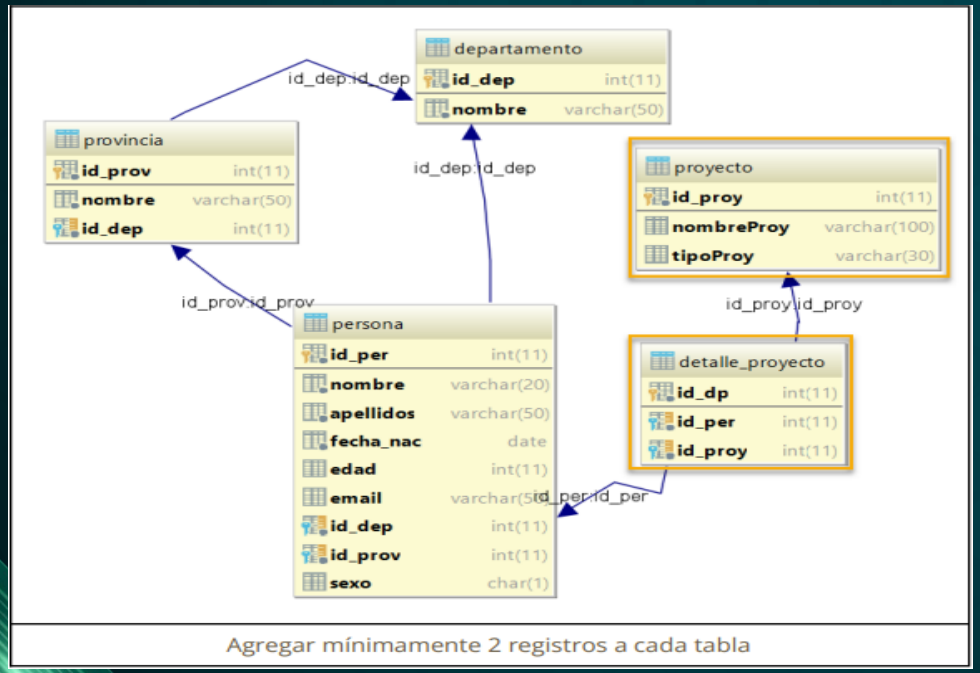
Cuando se habla de eventos en triggers, se refiere a las operaciones específicas que desencadenan la ejecución del código del trigger, como la inserción, actualización o eliminación de registros en una tabla. Los eventos definen cuándo y en qué circunstancias se activa el trigger y se pueden configurar para ejecutarse antes o después del evento.

1.8

02 PARTE PRACTICA

PARTE PRACTICA

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



Agregar mínimamente 2 registros a cada tabla

2.1

RESPUESTA

```
CREATE DATABASE defensa_hit4_2023;#notacion anderscorp
USE defensa_hit4_2023;


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

CREATE TABLE provincia
(
   id_prov INT PRIMARY KEY,
   nombre VARCHAR(50),
   id_dep INT,

   FOREIGN KEY (id_dep) REFERENCES departamento(id_dep)
);
```

RESPUESTA

```
CREATE TABLE persona
(
    id_per INT PRIMARY KEY,
    nombre VARCHAR(20),
    apellidos VARCHAR(50),
    fecha_nac DATE,
    edad INT,
    email VARCHAR(20),
    id_dep INT,
    id_prov INT,
    Ssexo CHAR(1),

    FOREIGN KEY (id_dep) REFERENCES departamento(id_dep),
    FOREIGN KEY (id_prov) REFERENCES provincia(id_prov)
);
```

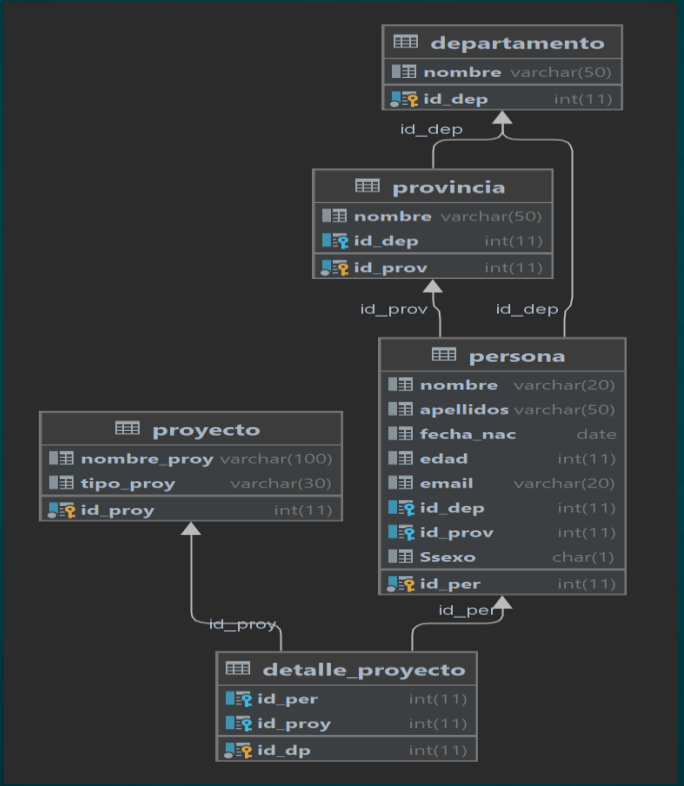
RESPUESTA

```sql
CREATE TABLE proyecto
(
    id_proy INT PRIMARY KEY,
    nombre_proy VARCHAR(100),
    tipo_proy VARCHAR(30)
);

CREATE TABLE detalle_proyecto
(
    id_dp INT PRIMARY KEY,
    id_per INT,
    id_proy INT,

    FOREIGN KEY (id_proy) REFERENCES proyecto(id_proy),
    FOREIGN KEY (id_per) REFERENCES persona(id_per)
);
```

RESPUESTA


departamento
nombre varchar(50)
id_dep int(11)
id_dep
provincia
nombre varchar(50)
id_dep int(11)
id_prov int(11)
id_prov
id_dep
persona
nombre varchar(20)
apellidos varchar(50)
fecha_nac date
edad int(11)
email varchar(20)
id_dep int(11)
id_prov int(11)
Ssexo char(1)
id_per int(11)
proyecto
nombre_proy varchar(100)
tipo_proy varchar(30)
id_proy int(11)
id_proy
id_per
detalle_proyecto
id_per int(11)
id_proy int(11)
id_dp int(11)

PARTE PRACTICA

10. 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.

2.2

PARTE PRACTICA

- Adjuntar el **código SQL generado y una imagen de su correcto funcionamiento.**

2.2

RESPUESTA

```
#EJERCICIO 10
CREATE OR REPLACE FUNCTION serieFibonacci(n INT)
RETURNS TEXT
BEGIN
    DECLARE a INT DEFAULT 1;
    DECLARE b INT DEFAULT 0;
    DECLARE c INT DEFAULT 0;
    DECLARE R TEXT DEFAULT '';
    DECLARE cont INT DEFAULT 0;

    WHILE (n > cont) DO
        SET R = CONCAT(R,c,' ,');
        SET c = a + b;
        SET a = b;
        SET b = c;
        SET cont = cont + 1;
    END WHILE;
    RETURN R;
END;
```

RESPUESTA

```
CREATE OR REPLACE FUNCTION sumarFibonacci(n INT)
RETURNS INT
BEGIN
   DECLARE serie TEXT;
   DECLARE suma INT DEFAULT 0;
   DECLARE numero INT;

   SET serie = serieFibonacci(n);

   WHILE serie != '' DO
      SET numero = CAST(SUBSTRING_INDEX(serie, ',', 1) AS UNSIGNED);
      SET suma = suma + numero;
      SET serie = SUBSTRING(serie FROM LOCATE(',', serie) + 1);
   END WHILE;

   RETURN suma;
END;
```

PARTE PRACTICA

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'

LA CONSULTA GENERADA PREVIAMENTE CONVERTIR EN UNA VISTA

2.3

RESPUESTA

```sql
#EJERCICIO 11
CREATE OR REPLACE VIEW  muestar_personas_del_sexo_Femenino AS
    SELECT CONCAT(p.nombre, ' ', p.apellidos) AS nombres_apellidos,
        p.edad,
        p.fecha_nac,
        pr.nombre_proy AS nombre_proyecto
    FROM persona p
        JOIN provincia pv ON p.id_prov = pv.id_prov
        JOIN departamento d ON pv.id_dep = d.id_dep
        JOIN detalle_proyecto dp ON p.id_per = dp.id_per
        JOIN proyecto pr ON dp.id_proy = pr.id_proy
    WHERE p.Ssexo = 'F'
      AND d.nombre = 'El Alto'
      AND p.fecha_nac = '2000-10-10';

SELECT * FROM muestar_personas_del_sexo_Femenino;
```

PARTE PRACTICA

12.Manejo de TRIGGERS I.

- 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**

2.4

PARTE PRACTICA

- 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**
- Adjuntar el **código SQL generado y una imagen de su correcto funcionamiento.**

2.4

RESPUESTA

```
#EJERCICIO 12
ALTER TABLE proyecto
ADD COLUMN estado VARCHAR(50);

CREATE OR REPLACE TRIGGER proyecto_insert_trigger
BEFORE INSERT ON proyecto
FOR EACH ROW
BEGIN
    IF NEW.tipo_proy = 'EDUCACION' OR NEW.tipo_proy = 'FORESTACIÓN' OR NEW.tipo_proy = 'CULTURA' THEN
        SET NEW.estado = 'ACTIVO';
    ELSE
        SET NEW.estado = 'INACTIVO';
    END IF;
END;
```

RESPUESTA

```
CREATE OR REPLACE TRIGGER proyecto_update_trigger
BEFORE UPDATE ON proyecto
FOR EACH ROW
BEGIN
   IF NEW.tipo_proy = 'EDUCACION' OR NEW.tipo_proy = 'FORESTACIÓN' OR NEW.tipo_proy = 'CULTURA' THEN
      SET NEW.estado = 'ACTIVO';
   ELSE
      SET NEW.estado = 'INACTIVO';
   END IF;
END;
```

PARTE PRACTICA

13. Manejo de Triggers II.

- 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.**

2.5

RESPUESTA

```
#EJERCICIO 13
CREATE TRIGGER calculaEdad
BEFORE INSERT ON persona
FOR EACH ROW
BEGIN
   SET NEW.edad = YEAR(CURDATE()) - YEAR(NEW.fecha_nac);
   IF MONTH(CURDATE()) < MONTH(NEW.fecha_nac) OR
     (MONTH(CURDATE()) = MONTH(NEW.fecha_nac) AND DAY(CURDATE()) < DAY(NEW.fecha_nac)) THEN
      SET NEW.edad = NEW.edad - 1;
   END IF;
END;
```

PARTE PRACTICA

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.**

2.6

RESPUESTA

```
#EJERCICIO 14
CREATE TABLE copia_persona (
    nombres VARCHAR(20),
    apellidos VARCHAR(50),
    fecha_nac DATE,
    edad INT,
    email VARCHAR(20),
    id_dep INT,
    id_prov INT,
    Ssexo CHAR(1)
);

CREATE TRIGGER insert_copia_persona
BEFORE INSERT ON persona
FOR EACH ROW
BEGIN
    INSERT INTO copia_persona (nombres, apellidos, fecha_nac, edad, email, id_dep, id_prov, Ssexo)
    VALUES (NEW.nombre, NEW.apellidos, NEW.fecha_nac, NEW.edad, NEW.email, NEW.id_dep, NEW.id_prov, NEW.Ssexo);
END;
```

PARTE PRACTICA

15. Crear una consulta SQL que haga uso de todas las tablas.
    - La consulta generada convertirlo a VISTA

2.7

RESPUESTA

```
#EJERCICIO 15
CREATE VIEW copia_persona_vista AS
SELECT nombres, apellidos, fecha_nac, edad, email, id_dep, id_prov, Ssexo
FROM copia_persona;

SELECT * FROM copia_persona_vista;
```

MUCHAS GRACIAS