

# UNIVERSIDAD PRIVADA FRANZ TAMAYO

## • DEFENSA HITO 4

Estudiante: Jose Luis Yanahuaya Mamani

Asignatura: Base de Datos II

Carrera: Ingeniería de Sistemas

Sede: El Alto

Paralelo: BDA 2

Docente: Lic. William Barra Paredes

Fecha: 07/06/2023

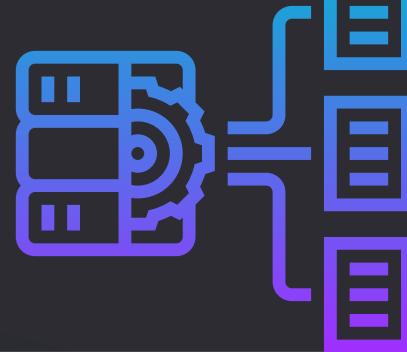
GITHUB:

<https://github.com/JoseYanahuaya/EvaluacionHito3/tree/main/EVALUACION HITO 3>



(use en sql y no en datagrip debido a que se me complica el uso del programa de datagrip debido al cambio de paralelo que tuve y como usted me indico que podria usar en sql lo use ingeniero, espero la comprension muchas gracias)





# Parte Teorica

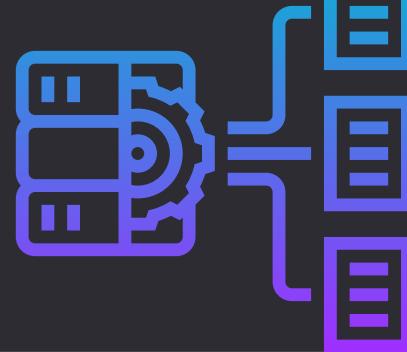
## 1. Defina que es lenguaje procedural en MySQL

El lenguaje procedural en MySQL es como un conjunto de pasos que siguen en orden para hacer cosas dentro de la base de datos. Es como si fuera una receta para cocinar, donde tienes diferentes pasos que sigues uno tras otro para obtener un resultado. En lugar de hacer todo en un solo paso, el lenguaje procedural te permite dividirlo en pequeñas partes más fáciles de entender y seguir. Así, puedes escribir bloques de código con instrucciones claras y ordenadas que ayudan a la base de datos a realizar tareas específicas, como calcular cosas o guardar información.

## 2. Defina que es una FUNCTION en MySQL

Una función en MySQL es como una máquina que toma algunos datos de entrada, hace algo con esos datos y devuelve un resultado. Por ejemplo tenemos que imaginar que tenemos una caja mágica que puede resolver ejercicios matemáticos. Puedes ponerle números y la maquina hará operaciones con esos números y te dará el resultado.

En MySQL, una función es como esa maquina. Le das algunos datos, que llamamos parámetros, y la función realiza ciertas acciones o cálculos con esos datos. Después, la función te devuelve un valor que es el resultado de esos cálculos.



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

Una función en MySQL es una herramienta que permite realizar cálculos o tareas específicas en una base de datos.

Un procedimiento almacenado es un conjunto de pasos que se guardan en la base de datos y se pueden ejecutar cuando sea necesario.

La diferencia principal entre una función y un procedimiento almacenado es que una función devuelve un resultado específico después de hacer cálculos con los datos de entrada, mientras que un procedimiento almacenado consiste en una serie de pasos para llevar a cabo una tarea en la base de datos.

en conclusión una función en MySQL es como una fórmula matemática que produce un resultado específico, mientras que un procedimiento almacenado es como una receta con pasos detallados para llevar a cabo una tarea en la base de datos. y tanto como el procedimiento y la función son para mejorar un requerimiento de un trabajo y hacerlo más fácil

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

### Ejecución de una función

Para ejecutar una función, debes utilizar la sentencia SELECT en una consulta SQL.

```
SELECT calcular_promedio (8.5, 7.8, 9.2) AS promedio;
```

(Esto ejecutará la función "calcular\_promedio" con las notas que daremos y nos dará el promedio como resultado.)

### Ejecución de un procedimiento

Para ejecutar un procedimiento almacenado, usaremos de ejemplo el "CALL" seguido del nombre del procedimiento y los valores de entrada.

```
CALL actualizar_stock ('Producto A', 10);
```

Esto ejecutará el procedimiento almacenado "actualizar\_stock" con el nombre del producto y la cantidad proporcionados.

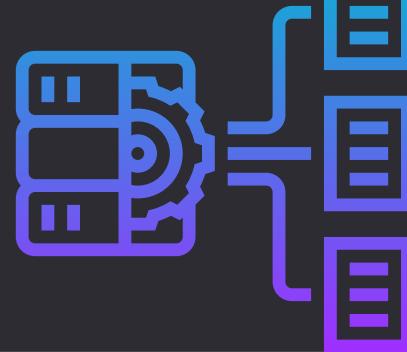


## 5. Defina que es una TRIGGER en MySQL.

Una TRIGGER en MySQL es un objeto de base de datos que se activa automáticamente cuando ocurre un evento específico en una tabla, como una inserción, actualización o eliminación de filas. Este objeto permite ejecutar un conjunto de pasos que la definiremos o predefiniremos en respuesta a ese evento, lo que ayuda a la capacidad de personalizar y controlar el comportamiento de la base de datos según las operaciones realizadas en la tabla.

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

En los triggers, las variables ANTERIOR y NUEVO (old , new) cumplen una tarea importante para acceder a los valores antiguos (ANTERIOR) y nuevos (NUEVO) en las columnas afectadas por la acción que hace el trigger. Estas variables le permiten comparar valores antes y después de una acción y realizar acciones personalizadas en función de esos cambios. En resumen, las variables OLD y NEW brindan información sobre el estado anterior y el estado actualizado de los datos en la tabla para que pueda tomar decisiones y realizar acciones en función de esos valores.



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

El papel que juega un trigger en el concepto de BEFORE y AFTER en un trigger es que determinan cuándo se ejecuta el código del trigger, ya sea antes o después de la operación en la tabla. Esto brinda flexibilidad y fácil manejo para realizar acciones personalizadas en diferentes momentos del proceso de uso de los datos en la base de datos.

BEFORE UPDATE ON usuarios

(Realiza acciones antes de la actualización)

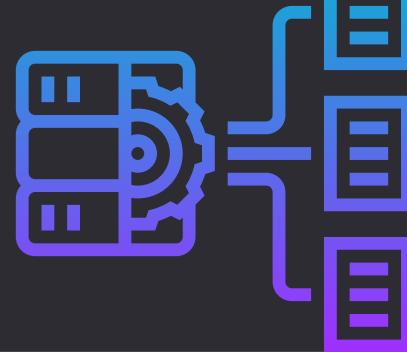
AFTER UPDATE ON usuarios

(Realiza acciones después de la actualización)

## 8. A que se refiere cuando se habla de eventos en TRIGGERS

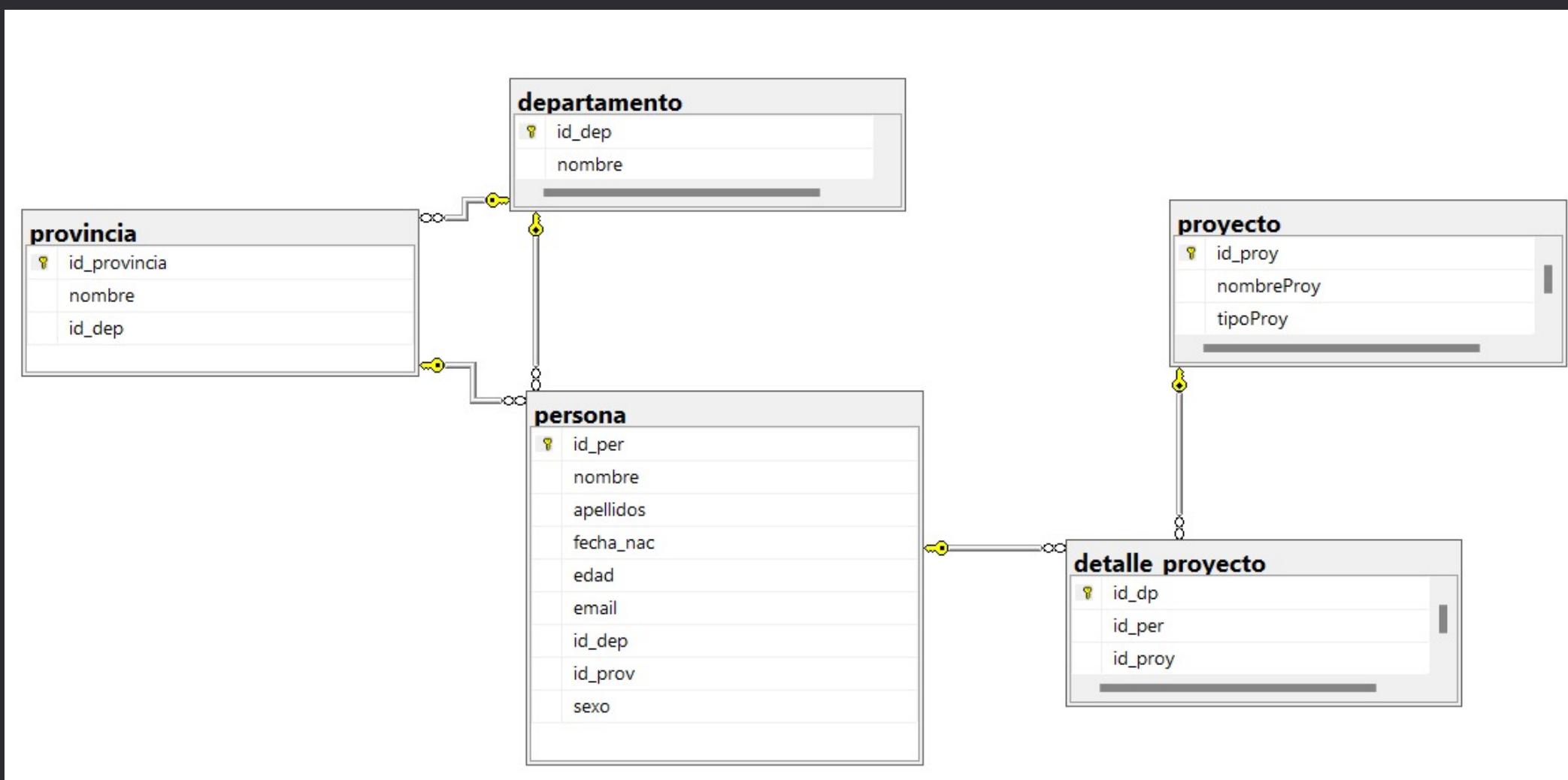
Cuando se habla de eventos en triggers, se refiere a las acciones específicas que ocurren en una tabla y que pueden activar la ejecución de un trigger. estos serian los eventos en un trigger

- **INSERT:** Se refiere a la inserción de una nueva fila en la tabla y se puede ejecutar un trigger asociado para realizar mas acciones.
- **UPDATE:** Se refiere a la actualización de varias filas o solo 1 fila en la tabla.
- **DELETE:** Se refiere a la eliminación de una tabla. Cuando se usa el delete, se desencadena el evento DELETE y se puede ejecutar un trigger para llevar a cabo acciones antes o después de la eliminación.



# Parte practica

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



Agregar mínimamente 2 registros a cada tabla

```
INSERT INTO `persona` (`id_per`, `nombre`, `apellidos`, `fecha_nac`, `edad`, `email`, `id_dep`, `id_prov`, `sexo`) VALUES
(1, 'Froilan ', 'Mamani Nina', '2003-07-12', 21, 'froimamani4@gmail.com', 1, 1, 'M'),
(2, 'Jose Luis ', 'Yanahuaya Mamani', '2004-05-12', 25, 'joseluis56@gmail.com', 3, 3, 'M'),
(3, 'Karen', 'Alizon Chuquimia', '2000-10-10', 64, 'karencita5454@gmail.com', 3, 3, 'F'),
(4, 'Miguel', 'Mamani Caiza', '2005-04-01', 17, 'tiomiguel@gmail.com', 2, 2, 'M');
```



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

```
3 CREATE FUNCTION generar_serie_fibonacci(n INT) RETURNS TEXT
4 BEGIN
5     DECLARE fibonacci TEXT DEFAULT '0,1';
6     DECLARE a INT DEFAULT 0;
7     DECLARE b INT DEFAULT 1;
8     DECLARE c INT;
9     DECLARE i INT DEFAULT 2;
10
11    WHILE i <= n DO
12        SET c = a + b;
13        SET fibonacci = CONCAT(fibonacci, ',', CAST(c AS CHAR));
14        SET a = b;
15        SET b = c;
16        SET i = i + 1;
17    END WHILE;
18
19    RETURN fibonacci;
20 END //
```



```
22 CREATE FUNCTION suma_serie_fibonacci(fibonacci_str TEXT) RETURNS INT
23 BEGIN
24     DECLARE sum INT DEFAULT 0;
25     DECLARE value CHAR(10);
26     DECLARE done INT DEFAULT 0;
27     DECLARE fibonacci_cursor CURSOR FOR SELECT TRIM(SUBSTRING_INDEX(SUBSTRING_INDEX(fibonacci_str, ',', numbers.n), ',', -1)) AS value
28         FROM (SELECT 1 + units.i + tens.i * 10 AS n
29             FROM (SELECT 0 AS i UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL SELECT 3 UNION ALL SELECT 4 UNION ALL SELECT 5 UNION ALL SELECT 6 UNION ALL SELECT 7 UNION ALL
30             SELECT 8 UNION ALL SELECT 9) units
31                 CROSS JOIN (SELECT 0 AS i UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL SELECT 3 UNION ALL SELECT 4 UNION ALL SELECT 5 UNION ALL SELECT 6 UNION ALL SELECT 7 UNION
32             ALL SELECT 8 UNION ALL SELECT 9) tens
33                 ORDER BY n
34             ) numbers
35             WHERE numbers.n <= LENGTH(fibonacci_str) - LENGTH(REPLACE(fibonacci_str, ',', '')) + 1;
36
37     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
38
39     OPEN fibonacci_cursor;
40
41     fibonacci_loop: LOOP
42         FETCH fibonacci_cursor INTO value;
43         IF done = 1 THEN
44             LEAVE fibonacci_loop;
45         END IF;
46         SET sum = sum + CAST(value AS INT);
47     END LOOP;
48
49     CLOSE fibonacci_cursor;
50
51     RETURN sum;
52 END //
```

## Opciones extra

suma\_fibonacci

88



Mostrar todo

Número de filas:



```
1 SET @serie_fibonacci = generar_serie_fibonacci(9);
2 SELECT @serie_fibonacci AS serie_fibonacci;
3
4 SET @suma_fibonacci = suma_serie_fibonacci(@serie_fibonacci);
5 SELECT @suma_fibonacci AS suma_fibonacci;
```

### Opciones extra

**serie\_fibonacci**

0,1,1,2,3,5,8,13,21,34



Mostrar todo

Número de filas



# 11. Manejo de vistas.

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

```
1 CREATE VIEW vista_persona_proyecto AS
2 SELECT CONCAT(p.nombre, ' ', p.apellidos) AS nombres_apellidos,
3        p.edad,
4        p.fecha_nac,
5        pr.nombreProy AS nombre_proyecto
6 FROM persona p
7 INNER JOIN detalle_proyecto dp ON dp.id_per = p.id_per
8 INNER JOIN proyecto pr ON pr.id_proy = dp.id_proy;
9
```

mostrar terminal de comandos SQL

✓ MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0,0109 segundos.)

CREATE VIEW vista\_persona\_proyecto AS SELECT CONCAT(p.nombre, ' ', p.apellidos) AS nombres\_apellidos, p.edad, p.fecha\_nac, pr.nombreProy AS nombre\_proyecto FROM persona p INNER JOIN detalle\_proyecto dp ON dp.id\_per = p.id\_per INNER JOIN proyecto pr ON pr.id\_proy = dp.id\_proy;

[ Editar en línea ] [ Editar ] [ Crear código PHP ]



## 11. Manejo de vistas.

```
1 SELECT * FROM vista_persona_proyecto;
```

**Perfilando** [ Editar en línea ] [ Editar ] [ Explicar SQL ] [ Crear ]

nombres_apellidos	edad	fecha_nac	nombre_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'

Ejecutar la(s) consulta(s) SQL en la base de datos tarea\_mito\_4:

```
1 SELECT CONCAT(p.nombre, ' ', p.apellidos) AS nombres_apellidos, p.edad, p.fecha_nac, pr.nombreProy AS nombre_proyecto
2 FROM persona p
3 JOIN departamento d ON p.id_dep = d.id_dep
4 JOIN provincia pv ON p.id_prov = pv.id_provincia
5 JOIN detalle_proyecto dp ON p.id_per = dp.id_per
6 JOIN proyecto pr ON dp.id_proy = pr.id_proy
7 WHERE p.sexo = 'F'
8 AND d.nombre = 'El Alto'
9 AND p.fecha_nac = '2000-10-10';
10
```

**Perfilando** [ Editar en línea ] [ Editar ] [ Explicar SQL ] [ Crear ]

nombres_apellidos	edad	fecha_nac	nombre_proyecto
-------------------	------	-----------	-----------------

Operaciones sobre los resultados de la consulta

Crear vista

COMO PUEDE VER NO ENCONTRO NADA DEBIDO QUE EN NUESTRA BASE DE DATOS NO EXISTE ESA EDAD NI EL SEXO NI EL DEPARTAMENTO DE EL ALTO



## 12. Manejo de TRIGGERS I.

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

Agregar un nuevo campo a la tabla PROYECTO.

- El campo debe llamarse ESTADO

```
1 ALTER TABLE PROYECTO
2 ADD COLUMN ESTADO VARCHAR(10);
```

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 llega un tipo de proyecto distinto colocar INACTIVO

```
1 DELIMITER $$ 
2 CREATE TRIGGER trg_proyecto_insert
3 BEFORE INSERT ON PROYECTO
4 FOR EACH ROW
5 BEGIN
6     IF NEW.tipoProy IN ('EDUCACION', 'FORESTACION', 'CULTURA') THEN
7         SET NEW.ESTADO = 'ACTIVO';
8     ELSE
9         SET NEW.ESTADO = 'INACTIVO';
10    END IF;
11 END$$
12 DELIMITER ;
```



## 12. Manejo de TRIGGERS I.

```
1 DELIMITER $$  
2 CREATE TRIGGER trg_proyecto_update  
3 BEFORE UPDATE ON PROYECTO  
4 FOR EACH ROW  
5 BEGIN  
6     IF NEW.tipoProy IN ('EDUCACION', 'FORESTACION', 'CULTURA') THEN  
7         SET NEW.ESTADO = 'ACTIVO';  
8     ELSE  
9         SET NEW.ESTADO = 'INACTIVO';  
10    END IF;  
11 END$$  
12 DELIMITER ;
```

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

<b>id_proy</b>	<b>nombreProy</b>	<b>tipoProy</b>	<b>ESTADO</b>
1	Proyecto A	EDUCACION	ACTIVO





# 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.

```
1 -- Crear el trigger
2 DELIMITER $$ 
3 CREATE TRIGGER calculaEdad
4 BEFORE INSERT ON PERSONA
5 FOR EACH ROW
6 BEGIN
7     DECLARE fecha_nac DATE;
8     DECLARE edad INT;
9
10    SET fecha_nac = NEW.fecha_nac;
11    SET edad = YEAR(CURRENT_DATE) - YEAR(fecha_nac);
12
13    IF DATE_FORMAT(CURRENT_DATE, '%m-%d') < DATE_FORMAT(fecha_nac, '%m-%d') THEN
14        SET edad = edad - 1;
15    END IF;
16
17    SET NEW.edad = edad;
18 END$$
19 DELIMITER ;
```

Mostrar ventana de consultas SQL

MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0.0107 segundos.)

```
CREATE TRIGGER calculaEdad BEFORE INSERT ON PERSONA FOR EACH ROW BEGIN DECLARE fecha_nac DATE; DEC
DATE_FORMAT(fecha_nac, '%m-%d') THEN SET edad = edad - 1; END IF; SET NEW.edad = edad; END;
```



# 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.

```
1
2 CREATE TABLE copia_persona (
3   nombre varchar(50) DEFAULT NULL,
4   apellidos varchar(50) DEFAULT NULL,
5   fecha_nac date DEFAULT NULL,
6   edad int(11) DEFAULT NULL,
7   email varchar(50) DEFAULT NULL,
8   id_dep int(11) DEFAULT NULL,
9   id_prov int(11) DEFAULT NULL,
10  sexo varchar(10) DEFAULT NULL
11 );
12
13
14 DELIMITER $$ 
15 CREATE TRIGGER copia_persona_trigger
16 BEFORE INSERT ON persona
17 FOR EACH ROW
18 BEGIN
19   INSERT INTO copia_persona (nombre, apellidos, fecha_nac, edad, email, id_dep, id_prov, sexo)
20   VALUES (NEW.nombre, NEW.apellidos, NEW.fecha_nac, NEW.edad, NEW.email, NEW.id_dep, NEW.id_prov, NEW.sexo);
21 END$$
22 DELIMITER ;
```

Mostrar ventana de consultas SQL

MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0.0142 segundos.)

```
CREATE TABLE copia_persona ( nombre varchar(50) DEFAULT NULL, apellidos varchar(50) DEFAULT NULL, fecha_nac date DEFAULT NULL, edad int(11) DEFAULT NULL, email varchar(50) DEFAULT NULL, id_dep int(11) DEFAULT NULL, id_prov int(11) DEFAULT NULL, sexo varchar(10) DEFAULT NULL );
```

[ Editar en línea ] [ Editar ] [ Crear código PHP ]

MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0.0046 segundos.)

```
CREATE TRIGGER copia_persona_trigger BEFORE INSERT ON persona FOR EACH ROW BEGIN INSERT INTO copia_persona (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;
```

[ Editar en línea ] [ Editar ] [ Crear código PHP ]



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

La consulta generada convertirlo a VISTA

```
1 |SELECT CONCAT(p.nombre, ' ', p.apellidos) AS nombres_apellidos,
2 |      p.edad,
3 |      p.fecha_nac,
4 |      pr.nombreProy AS nombre_proyecto,
5 |      d.nombre AS nombre_departamento,
6 |      pv.nombre AS nombre_provincia
7 |FROM persona p
8 |JOIN detalle_proyecto dp ON p.id_per = dp.id_per
9 |JOIN proyecto pr ON dp.id_proy = pr.id_proy
10 |JOIN departamento d ON p.id_dep = d.id_dep
11 |JOIN provincia pv ON p.id_prov = pv.id_provincia;
12 |
13 -- Convertir la consulta en una vista
14 CREATE VIEW vista_persona_proyecto_departamento AS
15 SELECT CONCAT(p.nombre, ' ', p.apellidos) AS nombres_apellidos,
16 |      p.edad,
17 |      p.fecha_nac,
18 |      pr.nombreProy AS nombre_proyecto,
19 |      d.nombre AS nombre_departamento,
20 |      pv.nombre AS nombre_provincia
21 |FROM persona p
22 |JOIN detalle_proyecto dp ON p.id_per = dp.id_per
23 |JOIN proyecto pr ON dp.id_proy = pr.id_proy
24 |JOIN departamento d ON p.id_dep = d.id_dep
25 |JOIN provincia pv ON p.id_prov = pv.id_provincia;
26 |
```



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

