

/DEFENSA

FINAL HITO 4

BASE DE DATOS II

EDSON IVER CONDORI CONDORI
SIS10929449



TABLA DE CONTENIDO



/01

/PARTE TEORICA



Se explicara
conceptos
elementales de DBAII
que se vio durante
el Hito 4.



/02

/PARTE PRACTICA

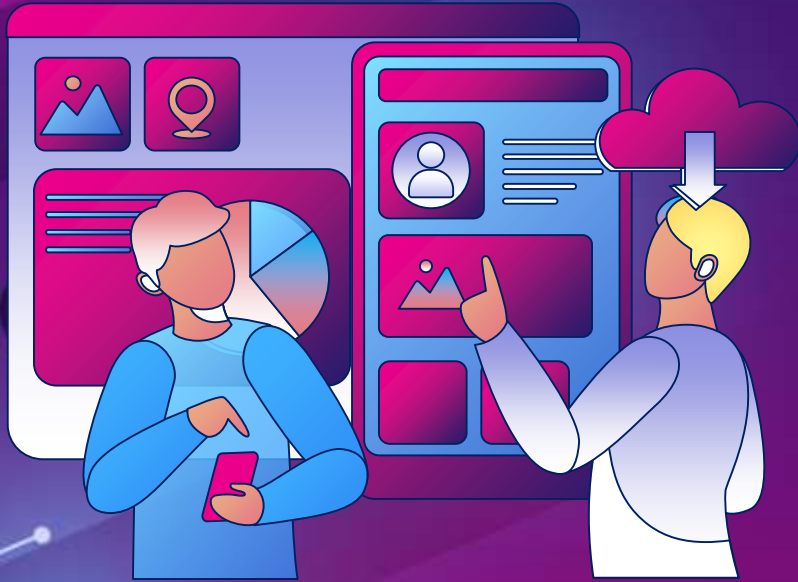


Se presentara la
aplicación de la
parte teórica para
la resolución de los
ejercicios.



COMENZEMOS

> / BDA II



01

PARTE TEORÍCA

Definición de conceptos
vistos en el Hito 4.

I. DEFINA QUE ES LENGUAJE PROCEDURAL EN MYSQL.

El lenguaje procedural en MySQL, da la capacidad a la base de Datos de permitir la creación de Procedimientos Almacenados , Funciones, etc., permitiéndonos de esa forma automatizar y simplificar tareas relacionadas con la Manipulación y Gestión de los datos almacenados en la Base de Datos, con el fin de mejorar la eficiencia y el rendimiento de las operaciones llevadas en la Base de Datos.



2. DEFINA QUE ES UNA FUNCIÓN EN MYSQL.

```
CREATE FUNCTION nombre_de_funcion ()  
RETURNS TEXT  
BEGIN  
    #Lo que se quier realizar  
  
    RETURN /*lo que se desea retornar*/;  
END;
```

Una función en MySQL es un programa que realiza una tarea específica y devuelve un valor como resultado, que puede ser utilizado en diferentes partes de una consulta SQL.

Las funciones son objetos almacenados en la base de datos y se pueden crear utilizando la instrucción "CREATE FUNCTION".

3. CUÁL ES LA DIFERENCIA ENTRE FUNCIONES Y PROCEDIMIENTOS ALMACENADOS.

La diferencia entre funciones y procedimientos almacenados en MariaDB es que las funciones devuelven un valor como resultado, mientras que los procedimientos no devuelven un valor específico.

Las funciones se utilizan principalmente para realizar cálculos y manipulaciones de datos complejas, mientras que los procedimientos se utilizan para realizar tareas y acciones en la base de datos.



FUNCION

Para ejecutar una función usamos:

Creación de
Función usando
Concat

Realizando la
Consulta a la
Función

```
# USO DEL CONCAT
CREATE OR REPLACE FUNCTION uso_de_concat(cad1 text,cad2 text,cad3 text)
RETURNS TEXT
BEGIN
    DECLARE response TEXT DEFAULT '';
    SET response = concat(cad1,' ',cad2,' ',cad3);
    RETURN response;
END;
SELECT uso_de_concat( cad1: 'Bienvenidos', cad2: 'a', cad3: 'BDA-II');
```

Salida de Consola

```
1 uso_de_concat('Bienvenidos','a','BDA-II')
1 Bienvenidos a BDA-II
```


4. CÓMO SE EJECUTA UNA FUNCIÓN Y UN PROCEDIMIENTO ALMACENADO.

FUNCION

```
# USO DEL CONCAT
CREATE OR REPLACE FUNCTION uso_de_concat(cad1 text,cad2 text,cad3 text)
RETURNS TEXT
BEGIN
    DECLARE response TEXT DEFAULT '';
    SET response = concat(cad1,' ',cad2,' ',cad3);
    RETURN response;
END;
SELECT uso_de_concat( cad1: 'Bienvenidos', cad2: 'a', cad3: 'BDA-II');
```

Creación de Función
usando Concat

Realizando la
Consulta a la
Función

Salida de Consola

```
1 uso_de_concat('Bienvenidos','a','BDA-II')
1 Bienvenidos a BDA-II
```

4. CÓMO SE EJECUTA UNA FUNCIÓN Y UN PROCEDIMIENTO ALMACENADO.

PROCEDIMIENTO ALMACENADO

```
CREATE OR REPLACE TRIGGER tr_audit_usuarios_rrhh
AFTER UPDATE
ON usuarios_rrhh
FOR EACH ROW
BEGIN
    DECLARE ANTES_CAMBI TEXT DEFAULT ' ';
    DECLARE DESPUES_CAMBI TEXT DEFAULT ' ';

    SET ANTES_CAMBI= CONCAT(OLD.id_usr, ' - ', OLD.nombre_completo, ' - ', OLD.fecha_nac);
    SET DESPUES_CAMBI = CONCAT(NEW.id_usr, ' - ', NEW.nombre_completo, ' - ', NEW.fecha_nac);

    # con el comando CALL llamamos al procedimiento almacenado
    CALL inserta_datos(
        NOW(),USER(),@HOSTNAME,'UPDATE',ANTES_CAMBI,DESPUES_CAMBI
    );
END;

select * FROM usuarios_rrhh;
```

5. DEFINA QUE ES UNA **TRIGGER** EN MYSQL.

Los TRIGGERS son programas almacenados que se ejecutan automáticamente cuando ocurre un evento.

INSERT - UPDATE - DELETE (EVENTOS)



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

OLD representa los valores antiguos de la fila antes de que se realice la operación.

Por ejemplo, si se está actualizando una fila, OLD contendrá los valores anteriores de esa fila antes de la actualización.

NEW representa los valores nuevos o actualizados de la fila después de que se haya realizado la operación.

Por ejemplo, si se está actualizando una fila, NEW contendrá los valores actualizados de esa fila.

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

```
CREATE OR REPLACE TRIGGER tr_audit_usuarios_rrhh
AFTER UPDATE
ON usuarios_rrhh
FOR EACH ROW
BEGIN
    DECLARE ANTES_CAMBI TEXT DEFAULT ' ';
    DECLARE DESPUES_CAMBI TEXT DEFAULT ' ';

    SET ANTES_CAMBI= CONCAT(OLD.id_usr, ' - ', OLD.nombre_completo, ' - ', OLD.fecha_nac);
    SET DESPUES_CAMBI = CONCAT(NEW.id_usr, ' - ', NEW.nombre_completo, ' - ', NEW.fecha_nac);

    INSERT INTO audit_usuario_rrhh_v2(fecha_mod, usuario_log, hostname, accion, antes_del_cambio,despues_del_cambio)
    SELECT NOW(),user(),@@HOSTNAME, 'UPDATE',ANTES_CAMBI, DESPUES_CAMBI;
END;
```

Diagram illustrating the use of OLD and NEW variables in a trigger:

- valores antiguos** (old values) points to the OLD variables in the SET statement.
- valores nuevos** (new values) points to the NEW variables in the SET statement.
- Uso de OLD y NEW** (Use of OLD and NEW) points to the SET statements.

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

"BEFORE" (antes): Un trigger con la cláusula "BEFORE" se ejecuta antes de que se realice la operación en la tabla. En este punto, los valores antiguos aún están presentes y se pueden modificar o validar antes de que se realice la operación en la tabla.

"AFTER" (después): Un trigger con la cláusula "AFTER" se ejecuta después de que se haya realizado la operación en la tabla. En este punto, los valores nuevos ya están presentes y se pueden utilizar para realizar acciones adicionales basadas en los cambios.

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

Nos referimos a las operaciones que ocurren en una tabla y que pueden activar un TRIGGER.

Los eventos mas comunes suelen ser:

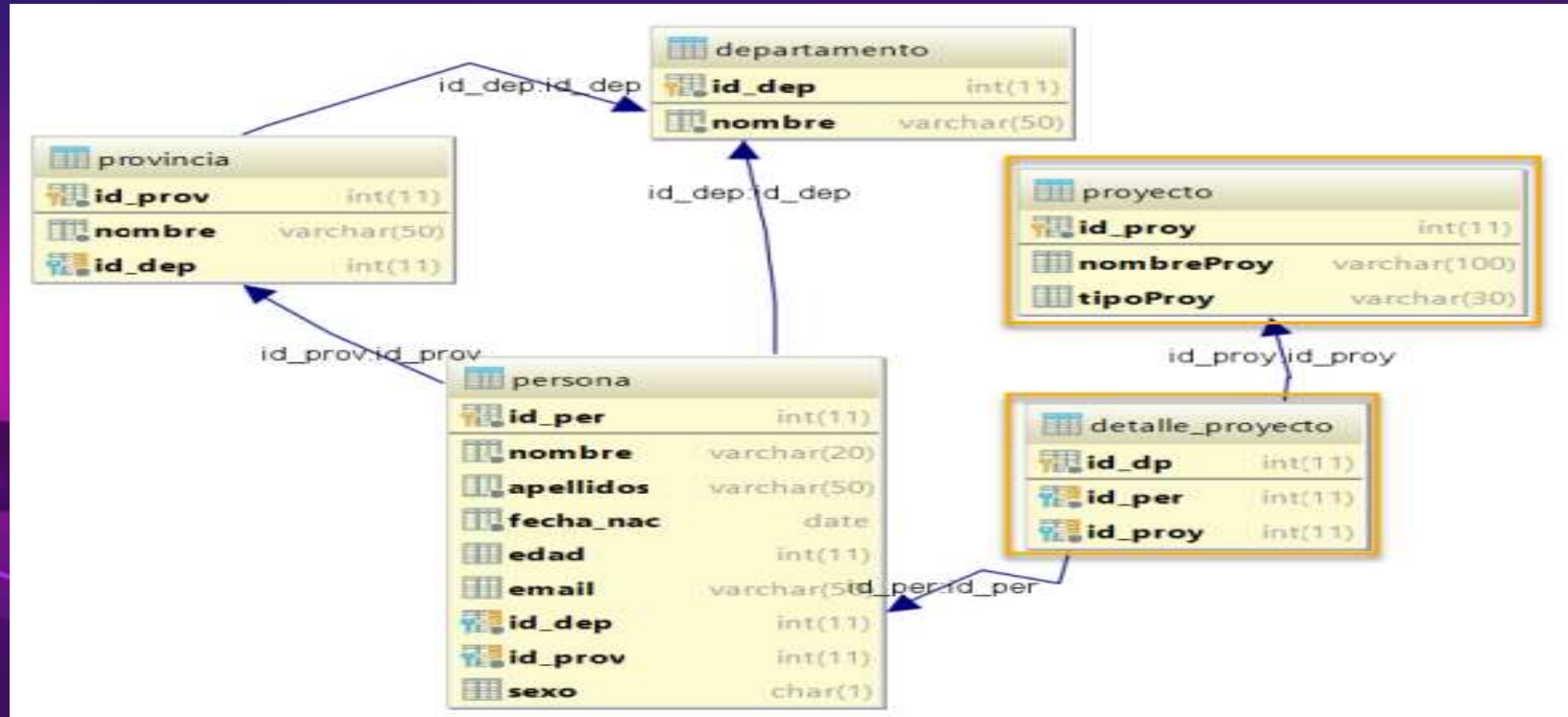
- **INSERT:** Suele ocurrir cuando se insertan nuevos datos en la tabla.
- **UPDATE:** Ocurre cuando se actualiza uno o mas datos / registros de una tabla.
- **DELETE:** Ocurre cuando se elimina una o varios registros de la tabla.

02

PARTE PRACTICA



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



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

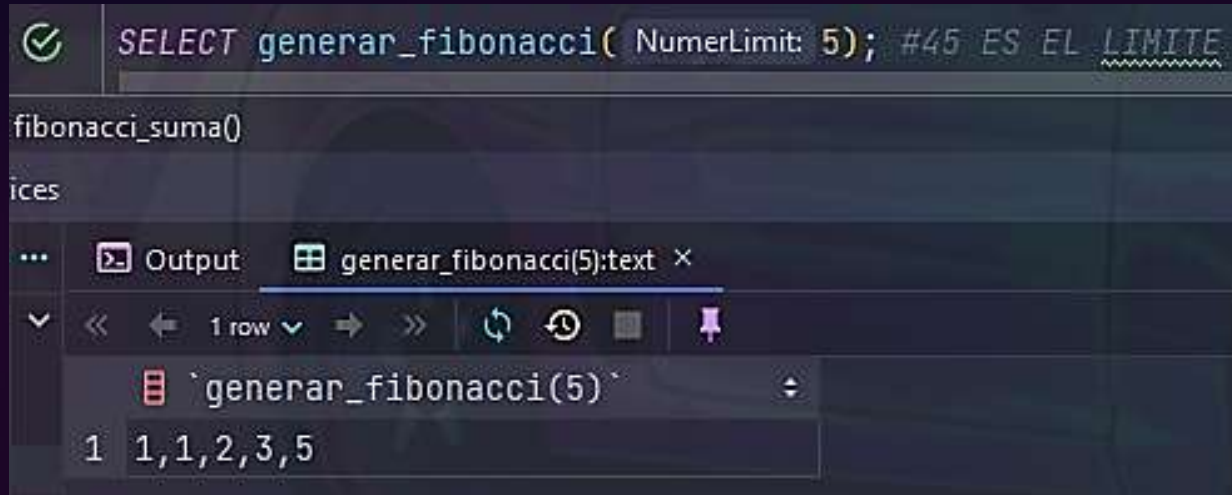
- El objetivo es sumar todos los números de la serie fibonacci desde una cadena.
- Es decir usted tendrá solo la cadena generada con los primeros N números de la serie fibonacci y a partir de ellos deberá sumar los números de esa serie.
- Ejemplo: `suma_serie_fibonacci(mi_metodo_que_retorna_la_serie(10))`
- 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.

<pre>"fibonacci(10)" 1 0, 1, 1, 2, 3, 5, 8, 13, 21, 34,</pre> <p>FUNCTION QUE GENERA LA SERIE</p>	<pre>"sumFibonacci(10)" 1 88</pre> <p>FUNCTION QUE SUMA LA SERIE</p>
---	--

Función que Genera la serie Fibonacci.

```
--# Calculando La Serie Figonaci  
CREATE OR REPLACE FUNCTION generar_fibonacci(NumerLimit INT)  
RETURNS TEXT  
BEGIN  
    DECLARE num1 INT DEFAULT 0;  
    DECLARE cont INT DEFAULT 1;  
    DECLARE num2 INT DEFAULT 1;  
    DECLARE result TEXT DEFAULT '';  
  
    SerieFigonaci: LOOP  
        IF cont > NumerLimit THEN  
            LEAVE SerieFigonaci; #LEAVE SALE DEL BUCLE  
        END IF;  
  
        SET result = CONCAT(result, num2, ',');  
        SET num2 = num1 + num2;  
        SET num1 = num2 - num1;  
        SET cont = cont + 1;  
    END LOOP SerieFigonaci;  
  
    RETURN SUBSTRING(result, 1, LENGTH(result) - 1);  
END;
```

Llamando a la Función que genera la serie



The screenshot shows a SQL query execution interface. The query entered is `SELECT generar_fibonacci(NumerLimit: 5);`. A comment in Spanish, `#45 ES EL LIMITE`, is visible next to the query. Below the query, the function `generar_fibonacci(5):text` is selected in the output pane. The output shows a single row with the value `1,1,2,3,5`.

```
SELECT generar_fibonacci( NumerLimit: 5); #45 ES EL LIMITE
```

fibonacci_suma()
ices

... Output generar_fibonacci(5):text x

1 row

1 1,1,2,3,5

Función que Suma la serie Fibonacci

```
CREATE OR REPLACE FUNCTION fibonacci_suma(NumerLimit INT)
RETURNS TEXT
BEGIN

    DECLARE num1 INT DEFAULT 0;
    DECLARE cont INT DEFAULT 1;
    DECLARE num2 INT DEFAULT 1;
    DECLARE result TEXT DEFAULT '';
    DECLARE suma INTEGER DEFAULT 0;

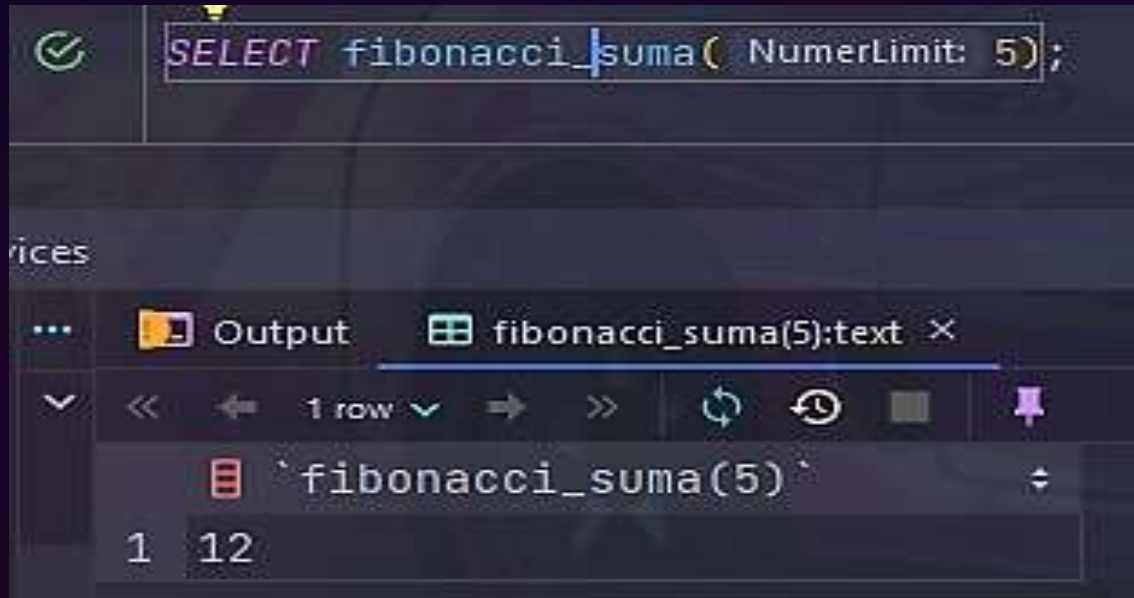
    SerieFibonacci: LOOP
        #verifica si se siguiera cumpliendo la secuencia
        IF cont > NumerLimit THEN
            LEAVE SerieFibonacci; #LEAVE SALE DEL BUCLE
        END IF;

        SET result = CONCAT(result, num2, ',');
        SET num2 = num1 + num2;
        SET num1 = num2 - num1;
        SET cont = cont + 1;

        SET suma = num1 + num2 - 1;

    END LOOP SerieFibonacci;
    RETURN suma;
end;
```

Llamando a la fusión que suma la serie



The screenshot shows a SQL query editor with a query window at the top and a results pane below it. The query window contains the following SQL statement:

```
SELECT fibonacci_suma( NumerLimit: 5);
```

The results pane shows the output of the query. It has a tab labeled "Output" and a sub-tab labeled "fibonacci_suma(5):text". The results are displayed in a table with one row and two columns. The first column is labeled "1" and the second column is labeled "12".

1	12

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

Consulta previa a la vista con salida en Consola

```
# Creando la consulta
SELECT CONCAT(per.nombre , ' ', per.apellido) Nombre_Completo, per.edad, per.fecha_nac , proy.nombreProy
FROM persona as per
INNER JOIN proyecto as proy on per.id_per = proy.id_proy
INNER JOIN departamento as dep on proy.id_proy = dep.id_dep
WHERE per.genero = 'F' and dep.nombre = 'El Alto' and per.fecha_nac = '2000-10-10';
```

ices

... Output # Creando la consulta ×

1 row

Nombre_Completo	edad	fecha_nac	nombreProy
1 Carla Soliz	23	2000-10-10	Reporte General

Uso de la VISTA con salida en Consola

```
# CREANDO LA VISTA
CREATE OR REPLACE VIEW vista_de_datos_1 as
  SELECT CONCAT(per.nombre , ' ', per.apellido) Nombre_Completo, per.edad, per.fecha_nac , proy.nombreProy
  FROM persona as per
  INNER JOIN proyecto as proy on per.id_per = proy.id_proy
  INNER JOIN departamento as dep on proy.id_proy = dep.id_dep
  WHERE per.genero = 'F' and dep.nombre = 'El Alto' and per.fecha_nac = '2000-10-10';

SELECT * FROM vista_de_datos_1;
```

ces

... Output defh4.vista_de_datos_1 X

VISTA

1 row

	Nombre_Completo	edad	fecha_nac	nombreProy
1	Carla Soliz	23	2000-10-10	Reporte General

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
- 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

Añadiendo la nueva columna y creando Trigger de Insert


```
#Añadiendo un nuevo campo a la tabla proyecto
ALTER TABLE proyecto
ADD COLUMN estado VARCHAR(20);

# CREANDO EL TRIGGER PARA INSERT
CREATE OR REPLACE TRIGGER tr_insert_estado
BEFORE INSERT ON proyecto
FOR EACH ROW
BEGIN
    IF (NEW.tipoProy = 'Educacion' or NEW.tipoProy = 'Forestacion' or NEW.tipoProy = 'Cultura') THEN
        SET NEW.estado = 'ACTIVO';
    ELSE
        SET NEW.estado = 'INACTIVO';
    end if;
END;
```

Insertando datos nuevos, llamando y viendo resultados por consola

```
INSERT INTO proyecto (nombreProy, tipoProy)
VALUES ('Tarea 5', 'CULTURA');

INSERT INTO proyecto (nombreProy, tipoProy)
VALUES ('TAREA 2', 'ANTROPOLOGIA');
```

 `SELECT * from proyecto;`

tr_insert_estado

rices

Output defh4.proyecto x

7 rows

	id_proy	nombreProy	tipoProy	estado
1	1	Maquetacion	Practico	<null>
2	2	Reporte General	HISTORIA	<null>
3	3	Enseñansa	EDUCACION	<null>
4	4	Reporte General	FORESTACION	<null>
5	5	Informe	CULTURA	<null>
6	8	Tarea 5	CULTURA	ACTIVO
7	9	TAREA 2	ANTROPOLOGIA	INACTIVO

Creando Trigger de Update

```
CREATE OR REPLACE TRIGGER tr_update_estado
BEFORE UPDATE ON proyecto
FOR EACH ROW
BEGIN
    IF (NEW.tipoProy = 'Educacion' or NEW.tipoProy = 'Forestacion' or NEW.tipoProy = 'Cultura') THEN
        SET NEW.estado = 'ACTIVO';
    ELSE
        SET NEW.estado = 'INACTIVO';
    end if;
END;
```

Actualizando el Campo de una tabla y mostrando el cambio por consola

```
# ACTUALIZANDO EL REGISTRO DE LA TABLA PROYECTO
UPDATE proyecto
SET tipoProy = 'Educacion'
WHERE id_proy = 1;

SELECT * FROM proyecto;
```

Output defh4.proyecto x

	id_proy	nombreProy	tipoProy	estado
1	1	Maquetacion	Educacion	ACTIVO
2	2	Reporte General	HISTORIA	<null>
3	3	Enseñansa	EDUCACION	<null>
4	4	Reporte General	FORESTACION	<null>
5	5	Informe	EDUCACION	<null>
6	8	Tarea 5	CULTURA	ACTIVO
7	9	TAREA 2	ANTROPOLOGIA	INACTIVO

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.

Creacion del Trigger que calcula de Edad

```
CREATE OR REPLACE TRIGGER tr_calculaEdad
BEFORE INSERT ON persona
FOR EACH ROW
BEGIN
    # NOS PERMITIRA SABER LA EDAD DE LA PERSONA
    SET NEW.edad = TIMESTAMPDIFF(YEAR, NEW.fecha_nac, CURDATE());
END;
```


Insertando nuevos registros sin mandar edad, el trigger genera la edad, mostrando resultado en consola

```
INSERT INTO persona (nombre, apellido, fecha_nac, email, genero, id_dep, id_prov)
VALUES ('EDSON', 'CONDORI', '2004-01-04', 'edsco@gmail.com', 'M', 2, 2);

SELECT * from persona;
```

Output defh4.persona ×

6 rows						Txc Auto	DDL	
	id_per	nombre	apellido	fecha_nac	edad			
4	4	Carla	SOLIZ	2000-10-10	23			
5	5	Pedro	Ramirez	1995-02-05	28			
6	6	EDSON	CONDORI	2004-01-04	19			

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.

Creando tabla de copia

```
# CRENADO LA TABLA DE COPIA
CREATE TABLE persona_COPIA (
    id_per INTEGER NOT NULL ,
    nombre VARCHAR (20) NOT NULL ,
    apellido VARCHAR (50) NOT NULL ,
    fecha_nac DATE,
    edad INTEGER,
    email VARCHAR (50) NOT NULL,
    genero CHAR NOT NULL ,

    id_dep INTEGER NOT NULL ,
    id_prov INTEGER NOT NULL ,

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

Crenado Trigger

```
# CREACION DEL TRIGGER QUE COPIARA LOS DATOS
~~~~~
CREATE OR REPLACE TRIGGER tr_copia_tabla_persona
BEFORE INSERT ON persona
FOR EACH ROW
BEGIN
    INSERT INTO persona_COPIA(id_per, nombre, apellido, fecha_nac, edad, email, genero, id_dep, id_prov)
    SELECT NEW.id_per, NEW.nombre, NEW.apellido, NEW.fecha_nac, NEW.edad, NEW.email, NEW.genero, NEW.id_dep, NEW.id_prov;
END;
```

Insertando nuevo registro a la tabla personas y mostrando consola

```
INSERT INTO persona (nombre, apellido, fecha_nac, edad, email, genero, id_dep, id_prov)  
VALUES ('Iver', 'Condori', '2003-04-01', 20, 'edco@yahoo.com', 'M', 4, 4);
```

```
SELECT * from persona;  
select * from persona_COPIA;
```

Output defh4.persona X

7 rows								
	id_per	nombre	apellido	fecha_nac	edad	email	genero	id_dep
6	6	EDSON	CONDORI	2004-01-04	19	edsco@gmail.com	M	2
7	7	Iver	Condori	2003-04-01	20	edco@yahoo.com	M	4

Insertando nuevo registro a la tabla personas y mostrando consola

```
INSERT INTO persona (nombre, apellido, fecha_nac, edad, email, genero, id_dep, id_prov)  
VALUES ('Iver', 'Condori', '2003-04-01', 20, 'edco@yahoo.com', 'M', 4, 4);
```

```
SELECT * from persona;  
select * from persona_COPIA;
```

Output defh4.persona X

7 rows								
	id_per	nombre	apellido	fecha_nac	edad	email	genero	id_dep
6	6	EDSON	CONDORI	2004-01-04	19	edsco@gmail.com	M	2
7	7	Iver	Condori	2003-04-01	20	edco@yahoo.com	M	4

Mostrando consola de la tabla copia_persona

```
select * from persona_COPIA;
```

Output defh4.persona_copia X

1 row

	id_per	nombre	apellido	fecha_nac	edad	email	genero	id_dep	id_prov
1	0	Iver	Condori	2003-04-01	20	edco@yahoo.com	M	4	4

15. Crear una consulta SQL que haga uso de todas las tablas, luego hacerla vista.

```
# CREANDO LAS VISTAS
CREATE OR REPLACE VIEW vista_general as
SELECT proye.id_proy, proye.nombreProy, dep.nombre DEPARTAMENTO, prov.nombre,
       CONCAT(pers.nombre, ' ', pers.apellido) NOMBRE_COMPLETO, detpro.id_pd
FROM proyecto as proye
INNER JOIN departamento as dep on proye.id_proy = dep.id_dep
INNER JOIN provincia as prov on dep.id_dep = prov.id_prov
INNER JOIN persona as pers on prov.id_prov = pers.id_per
INNER JOIN detalle_proyecto as detpro on pers.id_per = detpro.id_pd
WHERE dep.nombre = 'EL ALTO';

|
SELECT * FROM vista_general;
```

REALIZADO POR: EDSON CANDORI

Output defh4.vista_general x

	id_proy	nombreProy	DEPARTAMENTO	nombre	NOMBRE_COMPLETO	id_pd
1	4	Reporte General	El Alto	Carrasco	Carla Soliz	4

GRACIAS POR SU ATENCION !!
