BASE DE DATOS

Defina que es lenguaje procedural en MySQL

Los procedimientos almacenados MySQL, también conocidos como *Stored Procedure*, se presentan como conjuntos de instrucciones escritas en el lenguaje SQL. Su objetivo es realizar una tarea determinada, desde operaciones sencillas hasta tareas muy complejas.

Defina que es una FUCNTION en MySQL.

Las funciones son piezas de código que reciben datos de entrada, realizan operaciones con ellos y luego devuelven un resultado.

Cuál es la diferencia entre funciones y procedimientos almacenados.

FUNCIONES

- Las funciones siempre retornan un valor, mientras que un procedimiento almacenado puede que retorne un valor o puede que no lo haga. Es decir que puede comportarse como un método o como una función, haciendo una analogía con el desarrollo de software.

PROCEDIMIENTO ALMACENADOS

- Los procedimientos almacenados pueden ser invocados desde el entorno de desarrollo, es decir desde .NET, pero las funciones no. Así que al momento de desarrollar, siempre nos comunicaremos con procedimientos almacenados.

En resumen

- Las funciones de hecho no pueden ser invocadas por sí solas, mientras que los procedimientos almacenados sí.

Cómo se ejecuta una función y un procedimiento almacenado.

Una funciona se ejecuta:

```
create function NOMBRE
(@PARAMETRO TIPO=VALORPORDEFECTO)
returns TIPO
begin
INSTRUCCIONES
return VALOR
end;
```

- Luego del nombre se colocan (opcionalmente) los parámetros de entrada con su tipo.
- La cláusula "returns" indica el tipo de dato retornado.

Un procedimiento almacenado se ejecuta:

```
CREATE PROCEDURE  Saludar

    AS

    PRINT 'Hola, Como estas?';
```

GO : Indicamos GO para cerrar el lote que crea el procedimiento y empezar otro lote.

EXECUTE Saludar: De esta forma llamamos al procedimiento (y se ejecuta).

Defina que es una TRIGGER en MySQL

Es un objeto que se crea con la sentencia CREATE TRIGGER y tiene que estar asociado a una tabla. Un *trigger* se activa cuando ocurre un evento de inserción, actualización o borrado, sobre la tabla a la que está asociado.

En un trigger que papel juega las variables OLD y NEW

Variable NEW

- NEW almacena el valor que aporta la consulta a la base de datos. Con esta variable podemos acceder a los datos introducidos.
  Con *NEW.nombre_columna* se almacenará la información con el nuevo valor que tendrá ese registro modificado (desde un UPDATE o INSERT) en la tabla.
- Los trigger relacionados con DELETE no tendrán disponible la variable NEW.

Variable OLD

- OLD a diferencia de NEW, almacena el valor de las columnas que van a ser borradas o eliminadas. Al igual que pasa con NEW,
- OLD no está disponible en todas las instrucciones, más concretamente el valor no se puede recuperar cuando la instrucción es un INSERT.

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

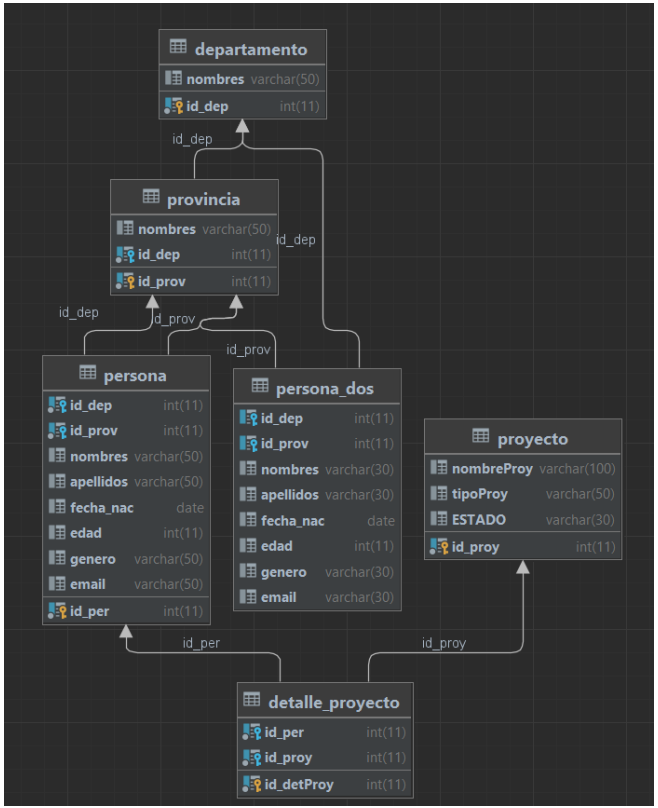
El modificador BEFORE o AFTER indica que el trigger se ejecutará antes o después de ejecutarse la sentencia SQL definida por DELETE, INSERT o UPDATE. Si incluimos el modificador OF el trigger solo se ejecutará cuando la sentencia SQL afecte a los campos incluidos en la lista.

- **BEFORE INSERT** Acciones a realizar antes de insertar uno más o registros en una tabla.
- **AFTER INSERT** Acciones a realizar después de insertar uno más o registros en una tabla.
- **BEFORE UPDATE** Acciones a realizar antes de actualizar uno más o registros en una tabla.
- **AFTER UPDATE** Acciones a realizar después de actualizar uno más o registros en una tabla.
- **BEFORE DELETE** Acciones a realizar antes de eliminar uno más o registros en una tabla.
- **AFTER DELETE** Acciones a realizar después de eliminar uno más o registros en una tabla.

A que se refiere cuando se habla de eventos en TRIGGERS

Los eventos en un trigger son las operaciones de inserción (INSERT), borrado (DELETE) o actualización (UPDATE), ya que modifican los datos de una tabla.
La utilidad principal de un trigger es mejorar la gestión de la base de datos, ya que no requieren que un usuario los ejecute

Crear la siguiente Base de datos ysus registros



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

```
#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.

CREATE OR REPLACE FUNCTION serie_Fibonacci (n INT)
RETURNS VARCHAR(255)
BEGIN
DECLARE serie VARCHAR(255);
DECLARE n1 INT;
DECLARE n2 INT;
DECLARE n3 INT;

SET n1 = 0;
SET n2 = 1;
SET serie = CONCAT(n1, ',', n2);

WHILE n > 2 DO
  SET n3 = n1 + n2;
  SET serie = CONCAT(serie, ',', n3);
  SET n1 = n2;
  SET n2 = n3;
  SET n = n - 1;
END WHILE;

RETURN serie;
END;

select serie_Fibonacci( n: 10);
```

	`serie_Fibonacci(10)`
1	0,1,1,2,3,5,8,13,21,34

```
#Funcion que suma los valores de la serie fibonacci

CREATE OR REPLACE FUNCTION suma_serie_fibonacci (serie VARCHAR(255))
RETURNS INT
BEGIN
DECLARE total INT DEFAULT 0;
DECLARE n INT DEFAULT 0;

WHILE LENGTH(serie) > 0 DO
  SET n = SUBSTRING_INDEX(serie, ',', 1);
  SET serie = SUBSTRING(serie, LENGTH(n) + 2);
  SET total = total + n;
END WHILE;

RETURN total;
END;

SELECT  suma_serie_fibonacci( serie: serie_fibonacci( n: 10));
```

	`suma_serie_fibonacci(serie_fibonacci(10))`
1	88

Manejo de vistas

```
#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 persona_view AS
SELECT CONCAT(p.nombres, ' ', p.apellidos) AS nombres, p.edad, p.fecha_nac, pr.nombreProy
FROM persona p
INNER JOIN detalle_proyecto dp ON p.id_per = dp.id_per
INNER JOIN proyecto pr ON dp.id_proy = pr.id_proy
WHERE p.genero = 'Femenino' AND p.id_dep = 2 AND p.fecha_nac = '2000-10-10';

SELECT * FROM persona_view;
```

	nombres	edad	fecha_nac	nombreProy
1	Maria Ramirez	30	2000-10-10	Proyecto 2

Manejo de TRIGGERS I.

```sql
ALTER TABLE proyecto ADD (ESTADO VARCHAR(30));

INSERT INTO proyecto(nombreProy, tipoProy)
VALUES ('Proyecto 5', 'EDUCACION'),
       ('Proyecto 6', 'FORESTACION'),
       ('Proyecto 7', 'CULTURA');


CREATE OR REPLACE TRIGGER update_proyecto
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;

CREATE OR REPLACE TRIGGER insert_proyecto
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;

INSERT INTO proyecto(nombreProy, tipoProy)
VALUES ('Proyecto 8','EDUCACION');
SELECT * FROM proyecto;
```

	id_proy	nombreProy	tipoProy	ESTADO
1	1	Proyecto 1	Tipo 1	<null>
2	2	Proyecto 2	Tipo 2	<null>
3	3	Proyecto 3	Tipo 3	<null>
4	4	Proyecto 4	Tipo 4	<null>
5	5	Proyecto 5	EDUCACION	<null>
6	6	Proyecto 6	FORESTACION	<null>
7	7	Proyecto 7	CULTURA	<null>
8	8	Proyecto 8	EDUCACION	ACTIVO

Manejo de Triggers II.

```
CREATE OR REPLACE TRIGGER calculaEdad
BEFORE INSERT ON persona
FOR EACH ROW
BEGIN
    SET NEW.edad = YEAR(CURDATE()) - YEAR(NEW.fecha_nac); #CURDATE() devuelve la fecha actual
END;

INSERT INTO persona(id_dep,id_prov,nombres, apellidos,fecha_nac,genero,email)
VALUES (1,1,'Roberto', 'Aguirrez', '1990-10-10','Masculino','roberto@gmail.com');

SELECT * FROM persona;
```

	id_per	id_dep	id_prov	nombres	apellidos	fecha_nac	edad	genero	email
1	1	1	1	Juan	Perez	1990-01-01	30	Masculino	juan@gmail.com
2	2	2	2	Maria	Ramirez	2000-10-10	30	Femenino	maria@gmail.com
3	3	2	2	Maria	Luz	1990-01-01	30	Femenino	maria@gmail.com
4	4	3	3	Pedro	Gomez	1990-01-01	30	Masculino	pedro@gmail.com
5	5	1	1	Roberto	Aguirrez	1990-10-10	32	Masculino	roberto@gmail.com

Manejo de TRIGGERS III.

```
#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_dos(

    id_dep INT,
    id_prov INT,
    nombres VARCHAR(30),
    apellidos VARCHAR(30),
    fecha_nac DATE,
    edad INT,
    genero VARCHAR(30),
    email VARCHAR(30),
    foreign key (id_dep) references departamento(id_dep),
    foreign key (id_prov) references provincia(id_prov)
);

CREATE OR REPLACE TRIGGER insert_persona_dos
BEFORE INSERT ON persona
FOR EACH ROW
BEGIN
    INSERT INTO persona_dos(id_dep,id_prov,nombres,apellidos,fecha_nac,edad,genero,email)
    VALUES (NEW.id_dep,NEW.id_prov,NEW.nombres,NEW.apellidos,NEW.fecha_nac,NEW.edad,NEW.genero,NEW.email);
END;

INSERT INTO persona(id_dep,id_prov,nombres, apellidos,fecha_nac,edad,genero,email)
VALUES (1,1,'Tania', 'jimenez', '1995-02-11',20,'Femenino','tania@gmail.com');

SELECT * FROM persona_dos;
```

	id_dep	id_prov	nombres	apellidos	fecha_nac	edad	genero	email
1	1	1	Tania	jimenez	1995-02-11	27	Femenino	tania@gmail.com

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

```sql
#15.Crear una consulta SQL que haga uso de todas las tablas
#○ La consulta generada convertirlo a VISTA

CREATE VIEW todas_las_tablas AS
SELECT CONCAT(
    persona.nombres, ' ', persona.apellidos) AS nombre_completo, #CONCAT() concatena los valores de las columnas
    persona.edad AS edad,
    dep.nombres AS departamento,
    prov.nombres AS provincia,
    CONCAT(proyecto.nombreProy, ' ', proyecto.tipoProy) AS proyecto

FROM persona
INNER JOIN departamento dep ON persona.id_dep = dep.id_dep
INNER JOIN provincia prov ON persona.id_prov = prov.id_prov
INNER JOIN detalle_proyecto ON persona.id_per = detalle_proyecto.id_proy
INNER JOIN proyecto ON detalle_proyecto.id_proy = proyecto.id_proy;

SELECT * FROM todas_las_tablas;
```

	nombre_completo	edad	departamento	provincia	proyecto
1	Juan Perez	30	La Paz	Inquisivi	Proyecto 1 Tipo 1
2	Maria Ramirez	30	El Alto	Laja	Proyecto 2 Tipo 2
3	Maria Luz	30	El Alto	Laja	Proyecto 3 Tipo 3
4	Pedro Gomez	30	Santa Cruz	Caranavi	Proyecto 4 Tipo 4