




# BASE DE DATOS II

FREDDY MACHACA MAMANI



01

# PARTE TEORICA

# PROCEDURE 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. Los procedimientos almacenados MySQL contienen una o más instrucciones SQL además de un procesamiento manipulador o lógico.



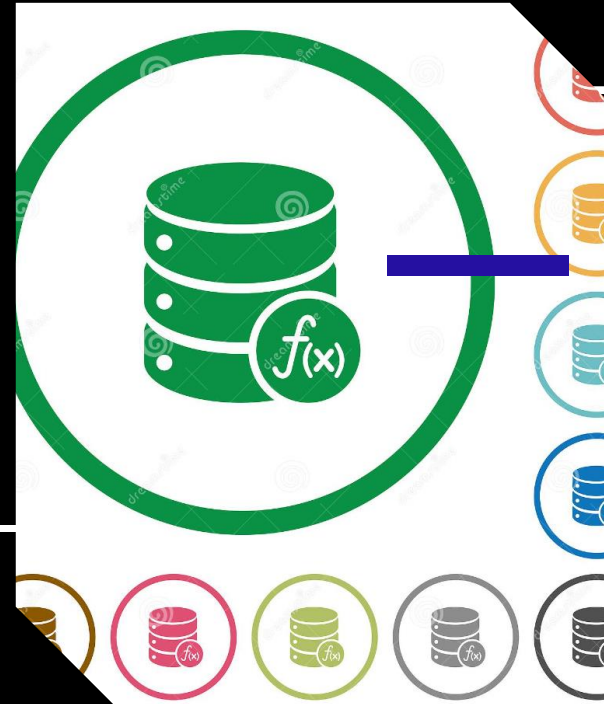
# FUNCTION MYSQL

Las funciones almacenadas de MySQL nos permiten procesar y manipular datos de forma procedural de un modo muy eficiente. Podrás usarlas en las sentencias SQL independientemente del lenguaje de programación del servidor sobre el que se ejecuten las consultas.



# DIFERENCIA ENTRE FUNCIONES Y PROCEDIMIENTOS ALMACENADOS.

Las funciones y los procedimientos almacenados son subprogramas que se ejecutan dentro de una base de datos. En términos generales, la principal diferencia entre las dos es que las funciones devuelven un valor, mientras que los procedimientos almacenados no lo hacen. Las funciones pueden ser llamadas desde instrucciones SQL como parte de una expresión, mientras que los procedimientos almacenados se deben llamar explícitamente. Las funciones también se pueden utilizar para restringir el acceso a los datos, mientras que los procedimientos almacenados se pueden usar para realizar tareas complejas.



## EJECUCION DE UNA FUNCION

Para ejecutar una función en MySQL, primero debe crear la función. Esto se hace con la sentencia CREATE FUNCTION. Una vez creada la función, se puede invocar usando la sentencia CALL. Por ejemplo:

```
CALL mi_funcion();
```

Tenga en cuenta que si la función necesita parámetros, debe proporcionarlos entre paréntesis luego del nombre de la función. Por ejemplo:

```
CALL mi_funcion(parámetro1, parámetro2);
```

Después de llamar a una función, MySQL devolverá cualquier valor que haya devuelto la función.

## EJECUCION DE UN PROCEDIMIENTO

Para ejecutar un procedimiento almacenado en MySQL, se usa el comando CALL. El sintaxis básica para usar el comando CALL es la siguiente: ■

```
CALL nombre_procedimiento (parametro1, parametro2, ...);
```

Donde nombre\_procedimiento es el nombre del procedimiento almacenado y los parámetros son los argumentos opcionales pasados al procedimiento.

Por ejemplo, para ejecutar un procedimiento almacenado llamado 'CalcularSalario' que toma dos parámetros (horas trabajadas y tarifa por hora), la sentencia podría ser algo como esto:

```
CALL CalcularSalario(40, 15.00);
```

# TRIGGER MYSQL

Una Trigger en MySQL es una instrucción SQL que se ejecuta automáticamente al ocurrir un evento específico en la base de datos. Estos eventos pueden ser la inserción, actualización o eliminación de datos, o incluso la ejecución de otras operaciones. Los Triggers son usadas para garantizar la consistencia de los datos, ya que se ejecutan automáticamente cada vez que se actualiza la base de datos.

Los Triggers también se usan para realizar operaciones complejas que no pueden ser ejecutadas directamente con SQL. Por ejemplo, un Trigger podría ser usado para enviar un correo electrónico o para actualizar otra base de datos.



# TRIGGER

## OLD - NEW

Las variables OLD y NEW son usadas en los triggers para referirse al valor antiguo o nuevo de una columna que ha sido modificada.

- OLD hace referencia al valor antiguo de la columna antes de la modificación.

- NEW hace referencia al valor nuevo de la columna después de la modificación.

Estas variables permiten a los programadores escribir código que realice acciones diferentes en función del valor antiguo o nuevo de la columna.





# TRIGGER

## BEFORE - AFTER

Las cláusulas BEFORE y AFTER son usadas dentro de un trigger para especificar el momento en el que desea que se ejecute el código. BEFORE se ejecutará justo antes de que se ejecute la acción que desencadene el trigger, mientras que AFTER se ejecutará justo después de que se haya ejecutado la acción que desencadene el trigger.

Es decir, BEFORE se ejecuta antes de que se ejecute la acción que desencadena el trigger y AFTER se ejecutará después de que se haya ejecutado la acción que desencadena el trigger.



# TRIGGER EVENTOS


Un trigger es una instrucción SQL que se ejecuta automáticamente cuando se producen ciertos eventos dentro de una base de datos.

Estos eventos pueden ser actualizaciones, inserciones o eliminaciones de datos.

Los triggers se utilizan para asegurar la consistencia de los datos y para asegurar que se cumplan todas las reglas relacionadas con la base de datos.

Por ejemplo, un trigger podría ser usado para verificar que una columna de una tabla siempre contenga un valor válido antes de permitir que se realice una actualización.



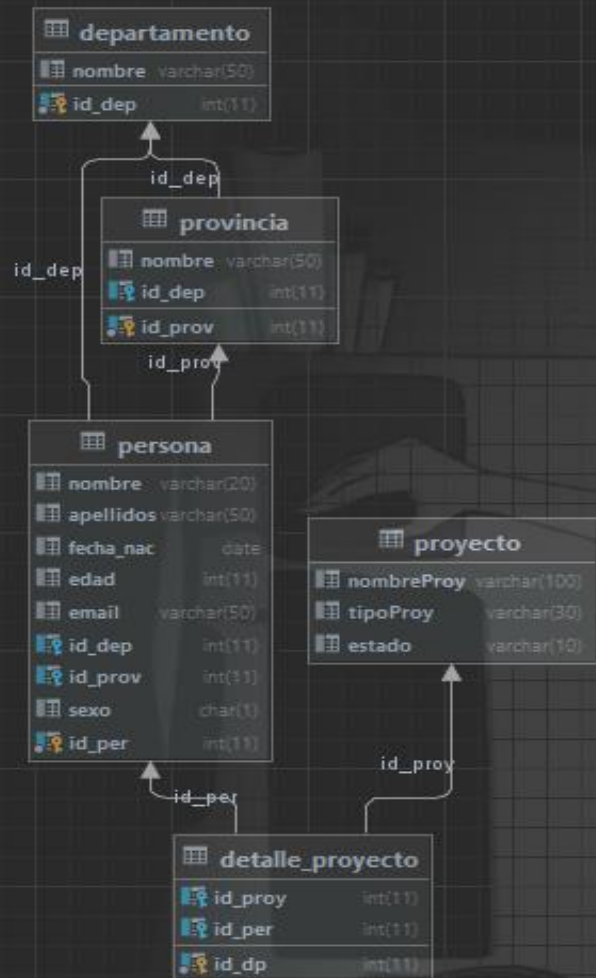
The background is black with several thin, white, hand-drawn style lines. A small cyan square is located in the upper-middle section. On the left side, there are horizontal white lines of varying lengths. On the right side, there are horizontal white lines, a horizontal blue bar, and a small cyan square at the bottom.

02

# PARTE PRACTICA

02

# MODELO E-R



# 02

## CODIGO SQL

```
create table departamento( id_dep ... );
```

```
create table provincia( id_prov ... );
```

```
create table proyecto( id_proy ... );
```

```
create table persona(  
    id_per int primary key not null,  
    nombre varchar(20),  
    apellidos varchar(50),  
    fecha_nac date,  
    edad int,  
    email varchar(50),  
  
    id_dep int,  
    id_prov int,  
    foreign key (id_dep) references departamento (id_dep),  
    foreign key (id_prov) references provincia (id_prov),  
    sexo char(1)  
);
```

```
create table detalle_proyecto(  
    id_dp int primary key not null,  
  
    id_proy int,  
    id_per int,  
    foreign key (id_proy) references proyecto (id_proy),  
    foreign key (id_per) references persona (id_per)  
);
```

# 02

## REGISTROS

```
insert into departamento values(1,'El alto');  
insert into departamento values(2,'Santa Cruz');  
insert into departamento values(3,'Cochabamba');
```

```
insert into provincia values(1,'provincia 1',1);  
insert into provincia values(2,'provincia 2',2);  
insert into provincia values(3,'provincia 3',3);
```

```
insert into persona values(2,'Maria','Gustamante', '2000-10-10', 30, 'maria@gmail.com', 1, 1, 'F');  
insert into persona values(1,'Joao','Salazar', '2000-10-10', 20, 'joao@gmail.com', 1, 1, 'M');  
insert into persona values(3,'Pedro','Alvez', '2000-10-10', 40, 'pedro@gmail.com', 3, 3, 'M');
```

```
insert into proyecto values(1,'proyecto1','tipo1');  
insert into proyecto values(2,'proyecto2','tipo2');  
insert into proyecto values(3,'proyecto3','tipo3');
```

```
insert into detalle_proyecto values(1,1,1);  
insert into detalle_proyecto values(2,2,2);  
insert into detalle_proyecto values(3,3,3);
```

# 02

## 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.
    - 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 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);
```

```
#suma de la serie fibonnaci
CREATE 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 serie_Fibonacci( n: 10);
```

Output serie\_Fibonacci(10):varchar(255) X

1 row

serie\_Fibonacci(10)

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

```
select suma_serie_fibonacci( serie: serie_Fibonacci( n: 10)) as 'suma de la serie fibonacci';
```

la\_view > pe

Output suma de la serie fibonacci(11) X

1 row

suma de la serie fibonacci

1 88

### 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

```
create or replace view persona_view as
SELECT CONCAT(pe.nombre,' ',pe.apellidos) as nombres_completo,pe.edad,pe.fecha_nac,proy.nombreProy
from persona as pe
inner join detalle_proyecto dp on pe.id_per = dp.id_per
inner join proyecto proy on dp.id_proy = proy.id_proy
inner join departamento d on pe.id_dep = d.id_dep
where pe.sexo = 'F' and d.nombre = 'El Alto' and pe.fecha_nac = '2000-10-10';

select * from persona_view;
```

Output procesual\_hito4.persona\_view x

	nombres_completo	edad	fecha_nac	nombreProy
1	Maria Gustamante	30	2000-10-10	proyecto2



# 02

## 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 llegat un tipo de proyecto distinto colocar **INACTIVO**
- Adjuntar el **código SQL generado y una imagen de su correcto funcionamiento.**

```
#estado es un nuevo campo de la tabla proyecto  
alter table proyecto add estado varchar(10);
```

```
CREATE or replace TRIGGER estado_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 (id_proy,nombreProy,tipoProy) VALUES (5,'proyecto5','Learning');  
SELECT * FROM proyecto;
```

Output procesual\_hito4.proyecto x

	id_proy	nombreProy	tipoProy	estado
1	1	proyecto1	tipo1	<null>
2	2	proyecto2	tipo2	<null>
3	3	proyecto3	tipo3	<null>
4	4	proyecto4	EDUCACION	ACTIVO
5	5	proyecto5	Learning	INACTIVO

# 02

## 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**.

```
CREATE or replace TRIGGER calculaEdad
BEFORE INSERT ON persona
FOR EACH ROW
BEGIN
SET NEW.edad = YEAR(CURDATE()) - YEAR(NEW.fecha_nac);
END;

INSERT INTO persona (id_per,nombre,apellidos,sexo,fecha_nac,id_dep) VALUES (4,'persona4','persona4','M','1990-10-10',1);
SELECT * FROM persona;
```

Output procesual\_hito4.persona X

	id_per	nombre	apellidos	fecha_nac	edad	email
1	1	Joao	Salazar	2000-10-10	20	joao@gmail.com
2	2	Maria	Gustamante	2000-10-10	30	maria@gmail.com
3	3	Pedro	Alvez	2000-10-10	40	pedro@gmail.com
4	4	persona4	persona4	1990-10-10	32	<null>
5	5	persona5	persona5	1990-10-10	32	<null>

## 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 copia_persona (  
    nombre varchar(20),  
    apellidos varchar(50),  
    fecha_nac date,  
    edad int,  
    email varchar(50),  
    id_dep int,  
    id_prov int,  
    sexo char(1),  
    foreign key (id_dep) references departamento(id_dep),  
    foreign key (id_prov) references provincia(id_prov)  
);  
  
CREATE or replace TRIGGER copia_persona  
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;  
  
INSERT INTO persona (id_per,nombre,apellidos,sexo,fecha_nac,id_dep) VALUES (5,'persona5','persona5','M','1990-10-10',2);  
SELECT * FROM copia_persona;
```

Output procesual\_hito4.copia\_persona

	nombre	apellidos	fecha_nac	edad	email	id_dep	id_prov	sexo
1 row	persona5	persona5	1990-10-10	32	<null>	2	<null>	M

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

- La consulta generada convertirlo a VISTA

```
#crear una consulta con todas las tablas en una vista
# ○ La vista debe de mostrar los siguientes campos:
# 1. nombres y apellidos concatenados
# 1. Nombre del departamento
# 2. detalle del proyecto
# 3. provincia
# 4. Nombre del proyecto

create or replace view persona_view_example as
SELECT CONCAT(pe.nombre, ' ', pe.apellidos) as nombres_completo, pe.edad, pe.fecha_nac, proy.nombreProy
from persona as pe
inner join detalle_proyecto dp on pe.id_per = dp.id_per
inner join proyecto proy on dp.id_proy = proy.id_proy
inner join departamento d on pe.id_dep = d.id_dep
inner join provincia p on pe.id_prov = p.id_prov;

select * from persona_view_example;
```

Output procesual\_hito4.persona\_view\_example

nombres_completo	edad	fecha_nac	nombreProy
1 Joao Salazar	20	2000-10-10	proyecto1
2 Maria Gustamante	30	2000-10-10	proyecto2
3 Pedro Alvez	40	2000-10-10	proyecto3