



# Trabajo Final

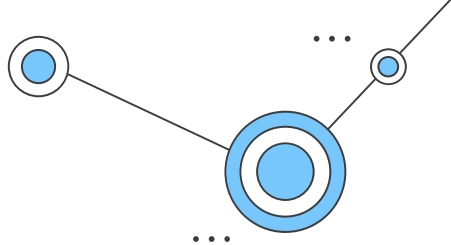
Base de Datos II

# 01

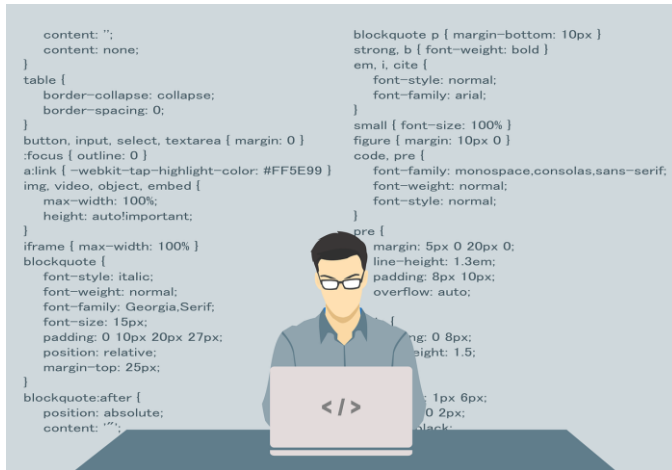
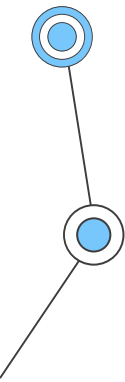
# Teorico

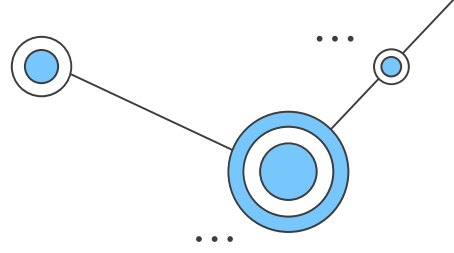
Manejo de Conceptos

# Defina que es lenguaje procedural en MySQL.



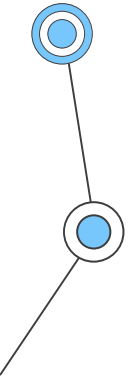
El lenguaje procedural en MySQL en el uso de procedimientos y funciones, es decir una código estructurado (nivel programación)



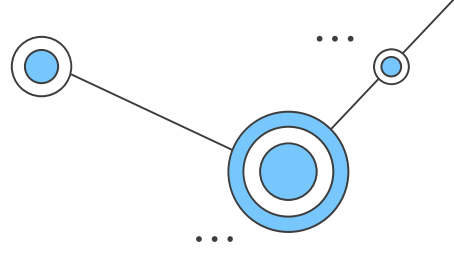


## Defina que es una **FUCNTION** en MySQL.

Es un fragmento de código donde nos retorna un valor con el uso de la clausula "select".

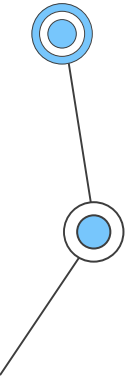


# Cuál es la diferencia entre funciones y procedimientos almacenados.

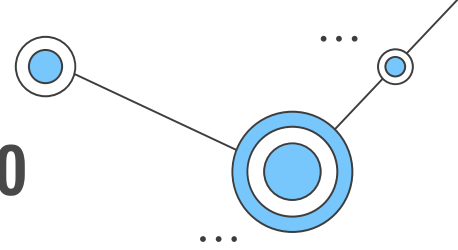


Las funciones retornan un valor predeterminado por el DBA

Los procedimientos no retorna ningún valor, ni con el "return" o "select"



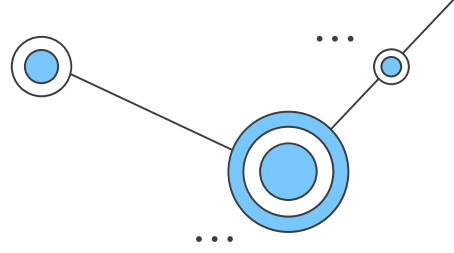
# Cómo se ejecuta una función y un procedimiento almacenado.



Al ejecutar una función nos apoyamos en la clausula "select".

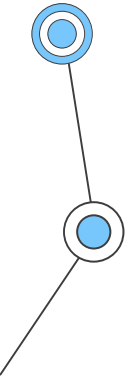
Para ejecutar un procedimiento nos apoyamos en la clausula "call".

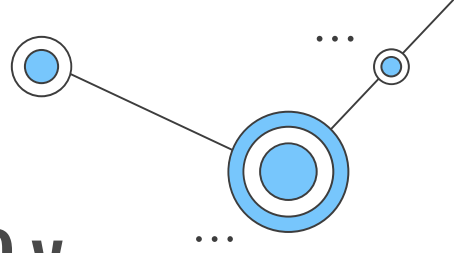




## Defina que es una TRIGGER en MySQL.

Es un procedimiento que se ejecuta de manera automática antes y después en el evento de una tabla

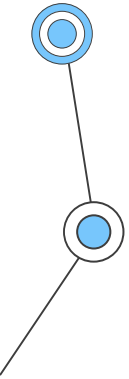




## En un trigger que papel juega las variables OLD y NEW

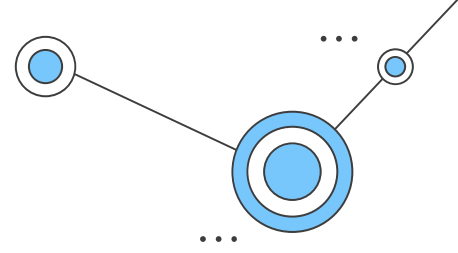
El "NEW" es una variable que tiene el dato que se ejecuta después de ejecutar el Trigger.

El "OLD" es una variable que tiene el dato antes de ejecutar el Trigger.



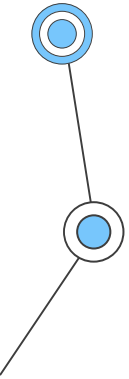


## En un trigger que papel juega los conceptos(cláusulas) BEFORE o AFTER

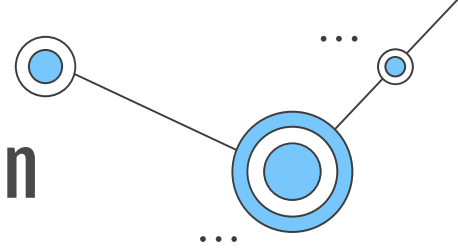


El "BEFORE" hace que el proceso se ejecute antes de realizar un cambio.

El "AFTER" hace que el proceso se ejecute después de realizar un cambio en la tabla

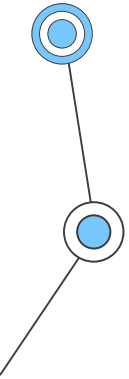


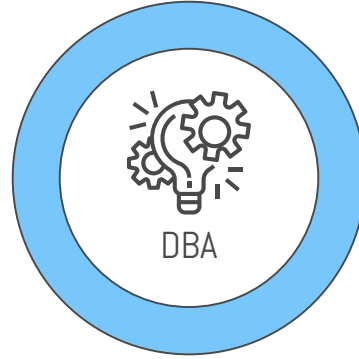
# A que se refiere cuando se habla de eventos en TRIGGERS



Los eventos dentro de los triggers son los que se realizan en una tabla como ser:

- Update
- Insert
- Delete



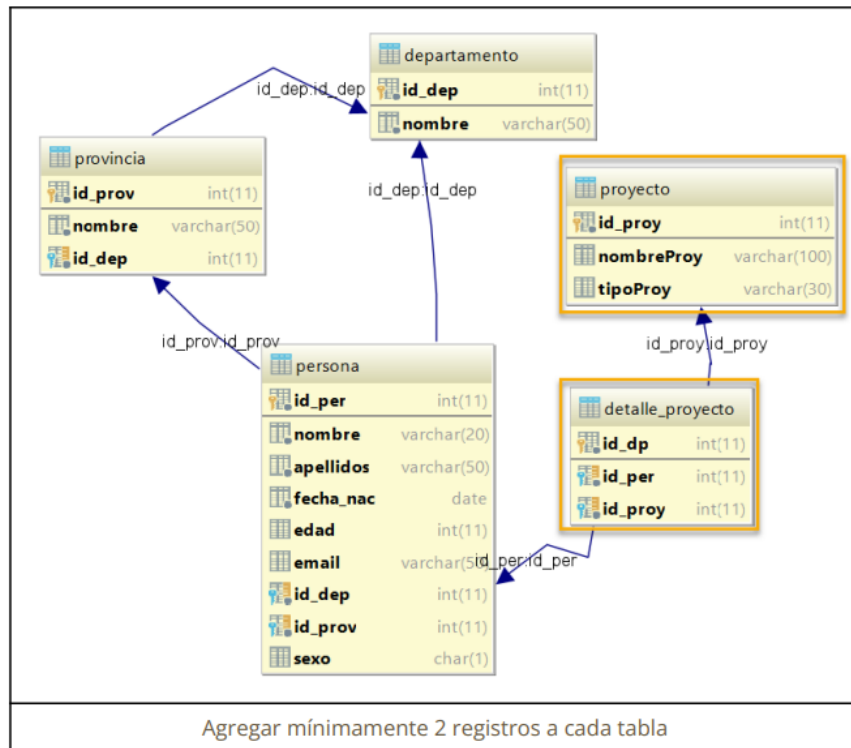


# Parte practica

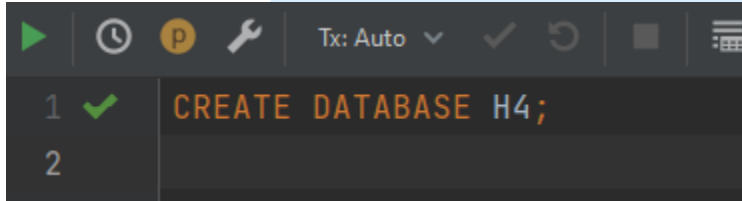
Resolucion de los Ejercicios

...

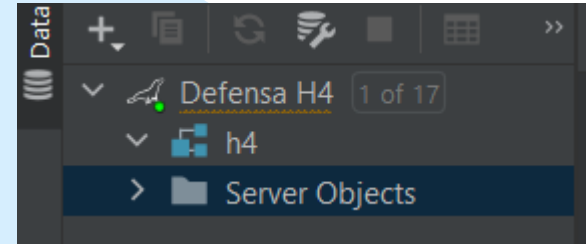
# Crear la siguiente Base de datos y sus registros.



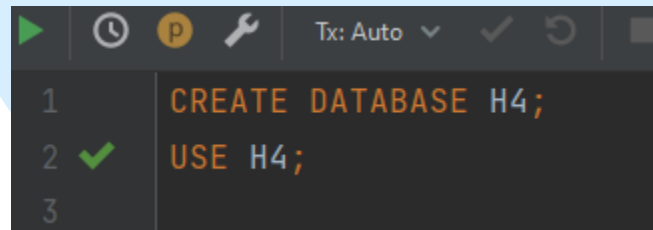
## CREAMOS LA BASE DE DATOS



A screenshot of the SQL Server Enterprise Manager interface. The top toolbar shows icons for execution (green play button), a clock, a yellow 'P' icon, a wrench, and a dropdown menu set to 'Tx: Auto'. Below the toolbar, a list of steps is shown on the left: step 1 has a green checkmark, and step 2 is currently selected. The main pane displays the SQL command: `CREATE DATABASE H4;`



## NOS POSICIONAMOS EN LA BASE DE DATOS



A screenshot of the SQL Server Enterprise Manager interface. The top toolbar shows icons for execution (green play button), a clock, a yellow 'P' icon, a wrench, and a dropdown menu set to 'Tx: Auto'. Below the toolbar, a list of steps is shown on the left: step 1 has a green checkmark, step 2 has a green checkmark, and step 3 is currently selected. The main pane displays the SQL command: `USE H4;`

# CREAMOS LA TABLA DEPARTAMENTO

```
4  
5 ✓ CREATE TABLE departamento  
6 (  
7     id_dep INT PRIMARY KEY AUTO_INCREMENT,  
8     nombre VARCHAR (50)  
9 );  
10
```

# CREAMOS LA TABLA PROVINCIA

```
11
12 ✓ CREATE TABLE provincia
13   (
14     id_prov INT AUTO_INCREMENT PRIMARY KEY,
15     nombre VARCHAR (50),
16     id_dep INT,
17     FOREIGN KEY (id_dep) REFERENCES departamento(id_dep)
18   );
19
```

# CREAMOS LA TABLA PERSONA

```
19
20 ✓ CREATE TABLE persona
21 (
22     id_per INT AUTO_INCREMENT PRIMARY KEY ,
23     nombre VARCHAR (50),
24     apellidos VARCHAR (50),
25     fecha_nac DATE,
26     edad INT,
27     email VARCHAR (50),
28     sexo CHAR,
29
30     id_dep INT,
31     FOREIGN KEY (id_dep) REFERENCES departamento(id_dep),
32
33     id_prov INT,
34     FOREIGN KEY (id_prov) REFERENCES provincia(id_prov)
35 );
```

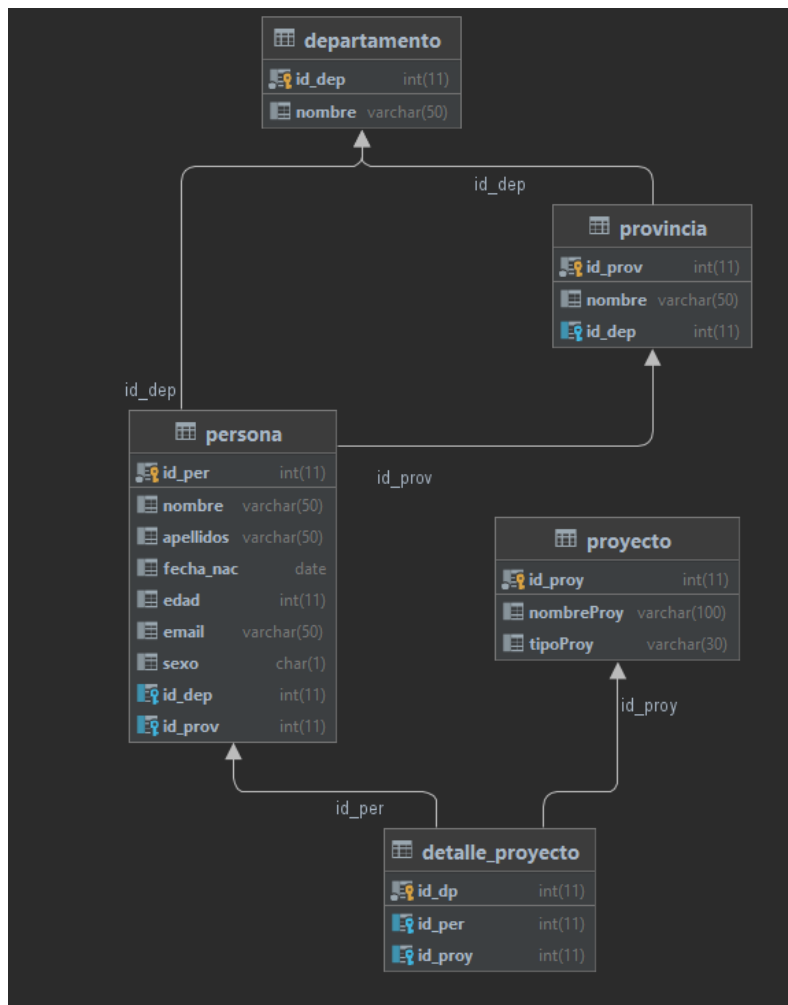
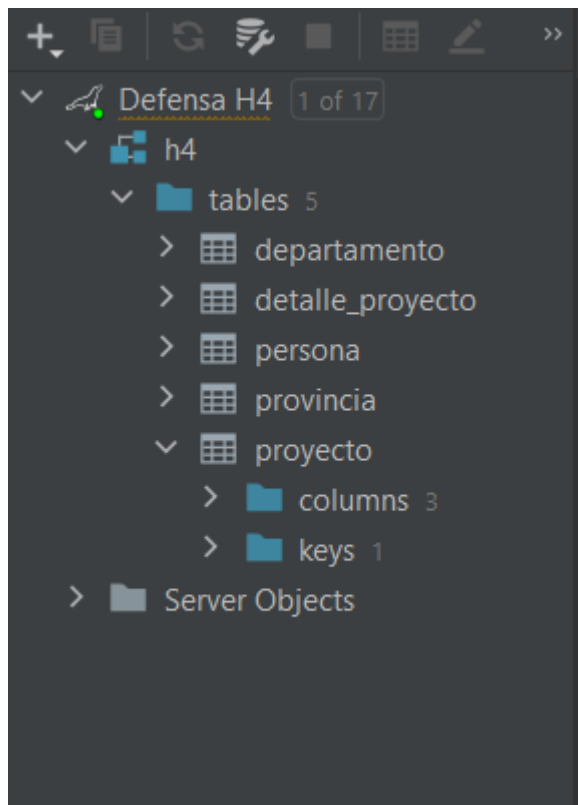


# CREAMOS LA TABLA PROYECTO

```
37  
38 ✓ CREATE TABLE proyecto  
39 (  
40     id_proy INT PRIMARY KEY AUTO_INCREMENT,  
41     nombreProy VARCHAR (100),  
42     tipoProy VARCHAR (30)  
43 );  
44
```

# CREAMOS LA TABLA DETALLE\_PROYECTO

```
45
46 ✓ CREATE TABLE detalle_proyecto
47 (
48     id_dp INT AUTO_INCREMENT PRIMARY KEY,
49     id_per INT,
50     FOREIGN KEY (id_per) REFERENCES persona(id_per),
51
52     id_proy INT,
53     FOREIGN KEY (id_proy) REFERENCES proyecto(id_proy)
54
55 );
56
```



# Llenamos datos a la tabla Departamento

```
✓ INSERT INTO departamento(nombre)
VALUES ('La Paz'),
       ('El Alto'),
       ('Santa Cruz'),
       ('Oruro'),
       ('El Alto');
```

WHERE

	id_dep	nombre
1	1	La Paz
2	2	El Alto
3	3	Santa Cruz
4	4	Oruro
5	5	El Alto

# Llenamos datos a la tabla Provincia

```
✓ INSERT INTO provincia(id_prov, nombre, id_dep)  
VALUES (1, 'provincia1', 1),  
       (2, 'provincia2', 2),  
       (3, 'provincia3', 3),  
       (4, 'provincia4', 4),  
       (5, 'provincia5', 5);
```

WHERE ORDER BY id\_prov

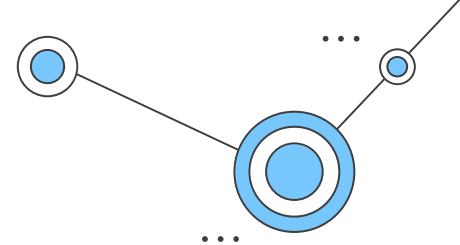
	id_prov	nombre	id_dep
1	1	provincia1	1
2	2	provincia2	2
3	3	provincia3	3
4	4	provincia4	4
5	5	provincia5	5

# Llenamos datos a la Tabla Persona

```
✓ INSERT INTO persona (id_per, nombre, apellidos, fecha_nac, edad, email, sexo, id_dep, id_prov)
VALUES (1, 'nombre1', 'apellido1', '2000-11-18', 18, 'nombre1@gmail.com', 'f', 1, 1),
(2, 'nombre2', 'apellido2', '2000-05-08', 21, 'nombre2@gmail.com', 'm', 2, 2),
(3, 'nombre3', 'apellido3', '2000-06-10', 24, 'nombre3@gmail.com', 'f', 3, 3),
(4, 'nombre4', 'apellido4', '2000-10-10', 28, 'nombre4@gmail.com', 'f', 4, 4),
(5, 'nombre5', 'apellido5', '2000-10-10', 32, 'nombre5@gmail.com', 'f', 5, 5);
```

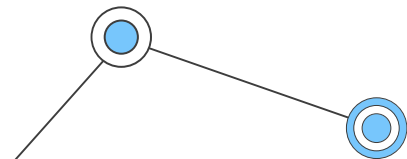
WHERE		ORDER BY							
	id_per	nombre	apellidos	fecha_nac	edad	email	sexo	id_dep	id_prov
1	1	nombre1	apellido1	2000-11-18	18	nombre1@gmail.com	f	1	1
2	2	nombre2	apellido2	2000-05-08	21	nombre2@gmail.com	m	2	2
3	3	nombre3	apellido3	2000-06-10	24	nombre3@gmail.com	f	3	3
4	4	nombre4	apellido4	2000-10-10	28	nombre4@gmail.com	f	4	4
5	5	nombre5	apellido5	2000-10-10	32	nombre5@gmail.com	f	5	5

# Llenamos datos a la tabla Proyecto

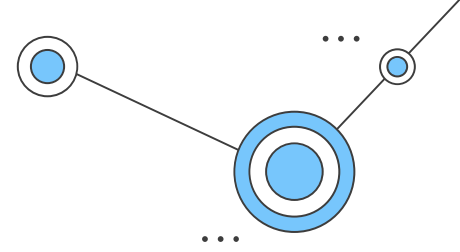


```
✓ INSERT INTO proyecto(nombreProy, tipoProy)
VALUES ('ProyectoA', 'Educacion'),
       ('ProyectoB', 'Forestacion'),
       ('ProyectoC', 'Cultural'),
       ('ProyectoD', 'Deportivo'),
       ('ProyectoE', 'Forestacion');
```

	id_proy	nombreProy	tipoProy
1	1	ProyectoA	Educacion
2	2	ProyectoB	Forestacion
3	3	ProyectoC	Cultural
4	4	ProyectoD	Deportivo
5	5	ProyectoE	Forestacion

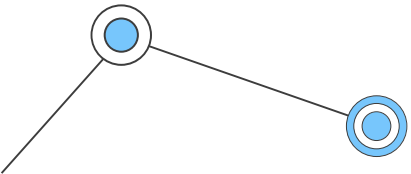


# Llenamos datos a la tabla Detalle\_Proyecto



```
✓ INSERT INTO detalle_proyecto (id_per, id_proy)
VALUES (1,1),
(2,2),
(3,3),
(4,4),
(5,5);
```

	id_dp	id_per	id_proy
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4
5	5	5	5

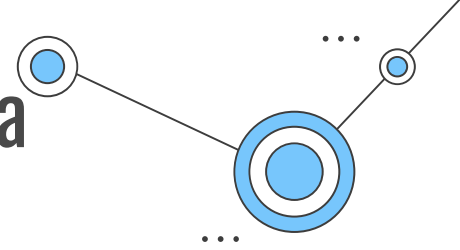




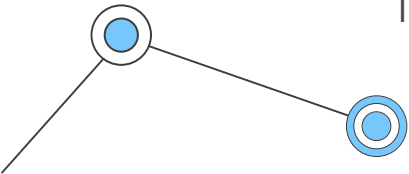


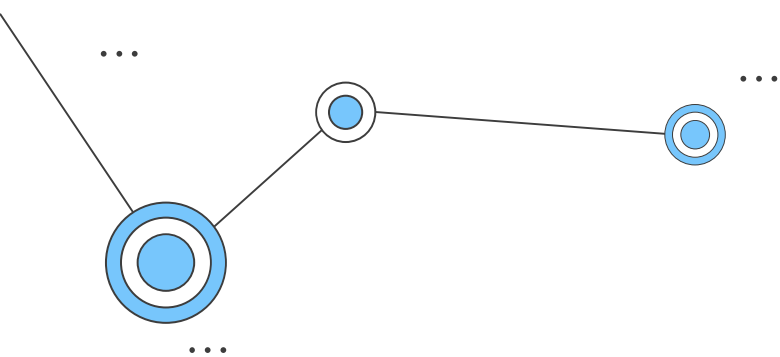
# Ejercicios Prácticos

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

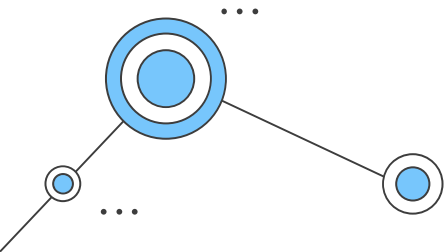


- El objetivo es sumar todos los números de la serie Fibonacci desde una cadena.
  - Es decir usted tendrá solo la cadena generada con los primeros N números de la serie Fibonacci y a partir de ellos deberá sumar los números de esa serie.
  - Ejemplo:  
`suma_serie_fibonacci(mi_metodo_que_retorna_la_serie(10))`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





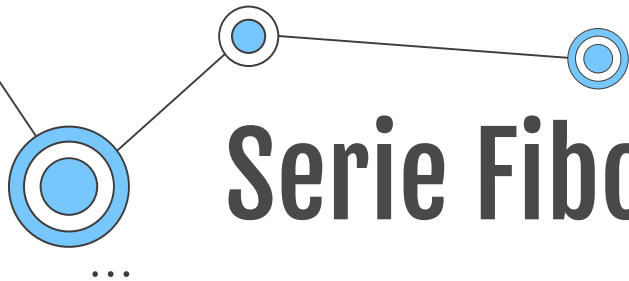
# Generamos la serie Fibonacci



```
✓ CREATE OR REPLACE FUNCTION serie_fibonacci(limite INT)
  RETURNS TEXT
  BEGIN
    DECLARE x INT DEFAULT 0;
    DECLARE y INT DEFAULT 1;
    DECLARE z INT DEFAULT 0;
    DECLARE aux INT DEFAULT 0;

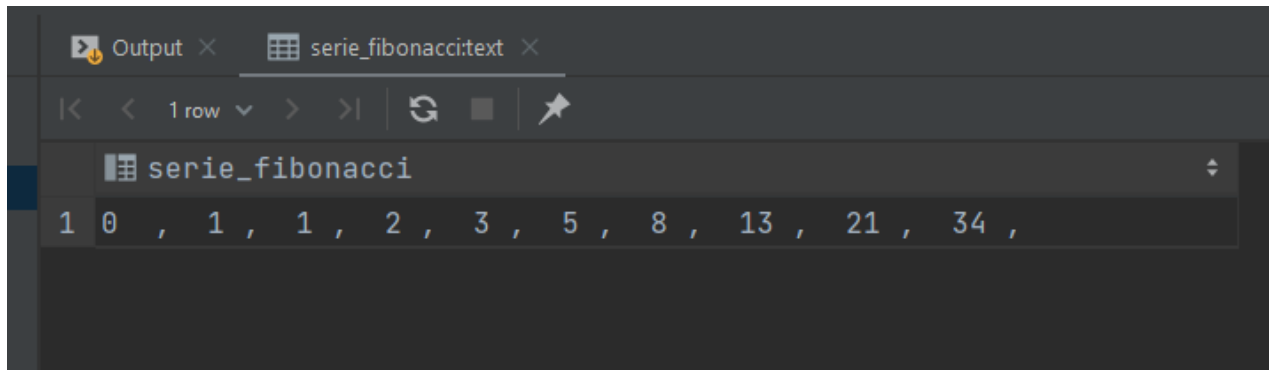
    DECLARE respuesta TEXT DEFAULT '';

    WHILE z < limite
      DO
        IF z = 0
          THEN
            SET respuesta = '0 ', ' ';
          ELSE
            SET respuesta = concat(respuesta, ' ', y, ' ', ' ');
            SET aux = x;
            SET x = y;
            SET y = aux + x;
          end if;
          SET z = z + 1;
        end while;
    RETURN respuesta;
  end;
```



# Serie Fibonacci

```
✓ SELECT serie_fibonacci( limite: 10) AS serie_fibonacci;
```



The screenshot shows a database interface with a tab labeled 'serie\_fibonacci:text'. Below the tab, there is a table with one row of data. The table header is 'serie\_fibonacci' and the data row contains the sequence of Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34.

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

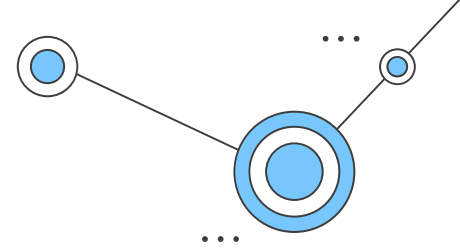


# Sumamos los valores de la serie Fibonacci

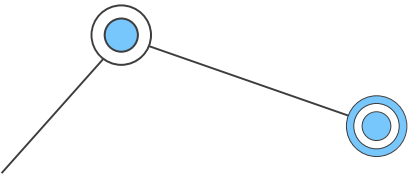


```
CREATE OR REPLACE FUNCTION sumaNumeros(cadena VARCHAR(100))  
  RETURNS INT  
  BEGIN  
    DECLARE num TEXT DEFAULT '';  
    DECLARE suma INT DEFAULT 0;  
    DECLARE limite TEXT DEFAULT 1;  
    DECLARE c INT DEFAULT 1;  
    DECLARE x INT DEFAULT 0;  
  
    IF LENGTH(cadena) > 0  
      THEN  
        WHILE(c <= LENGTH(cadena))  
          DO  
            SET limite = SUBSTRING(cadena, c, 1);  
  
            IF limite = ','  
              THEN  
                SET suma = suma+ num;  
                SET num = '';  
              ELSE  
                SET num = CONCAT(num, limite);  
              END IF;  
  
            SET c = c + 1;  
          END WHILE;  
  
          RETURN (suma);  
        ELSE  
          RETURN (x);  
        END IF;  
      END;
```

# Sumamos la serie Fibonacci



```
✓ SELECT sumaNumeros( cadena: serie_fibonacci( limite: 10)) as suma;
```



A screenshot of a database application interface. At the top, there are two tabs: 'Output' with a cursor icon and 'suma:int(11)' with a table icon. Below the tabs is a navigation bar with arrows, '1 row', and a refresh icon. The main area displays a table with one row. The column header is 'suma' with a dropdown arrow. The row contains the value '88'.

	suma
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'



# Manejo de Vistas

```
✓ CREATE OR REPLACE VIEW muestra_datos AS
  SELECT concat(per.nombre, ' ', per.apellidos) AS Nombre_Completo, per.edad, per.fecha_nac, pro.nombreProy
  FROM persona AS per
    INNER JOIN detalle_proyecto AS dep ON per.id_per = dep.id_per
    INNER JOIN proyecto AS pro ON dep.id_proy = pro.id_proy
    INNER JOIN departamento AS depa ON per.id_dep = depa.id_dep
  WHERE per.sexo = 'f' AND depa.nombre = 'El Alto' AND per.fecha_nac = '2000-10-10';
```

```
✓ SELECT datos.*
  FROM muestra_datos AS datos;
```

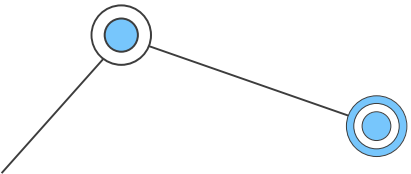
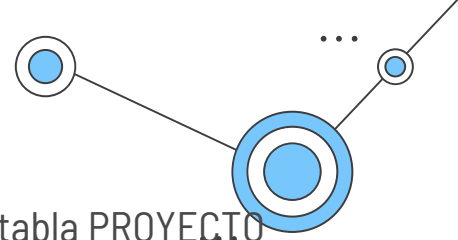
The screenshot shows a database client window with a tab labeled 'n4.muestra\_datos'. The interface includes a toolbar with navigation and execution icons. Below the toolbar, a table displays the results of the query. The table has four columns: 'Nombre\_Completo', 'edad', 'fecha\_nac', and 'nombreProy'. The first row of data shows 'nombre5 apellido5', '32', '2000-10-10', and 'ProyectoE'.

	Nombre_Completo	edad	fecha_nac	nombreProy
1	nombre5 apellido5	32	2000-10-10	ProyectoE



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



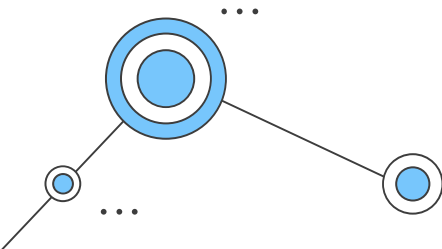






# Manejo de Triggers

Alteramos la tabla Proyecto, agregándole el campo de estado



```
ALTER TABLE proyecto ADD estado VARCHAR(8) NOT NULL;
```



	 id_proy ▾	 nombreProy ▾	 tipoProy ▾	 estado ▾
1	1	ProyectoA	Educacion	
2	2	ProyectoB	Forestacion	
3	3	ProyectoC	Cultural	
4	4	ProyectoD	Deportivo	
5	5	ProyectoE	Forestacion	

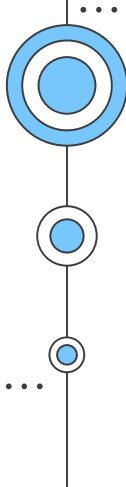
# Generamos la consulta para llenar el campo de estado (INSERT)

```
✓ CREATE OR REPLACE TRIGGER insertando_datos
  BEFORE INSERT ON proyecto
  FOR EACH ROW
  BEGIN
    CASE
      WHEN new.tipoProy = 'Educacion' OR new.tipoProy = 'Forestacion' OR new.tipoProy = 'Cultura'
      THEN
        SET new.estado = 'Activo';
      ELSE
        SET new.estado = 'Inactivo';
      END CASE;
  END;
```

# Generamos la consulta para llenar el campo de estado (UPDATE)

```
✓ CREATE TRIGGER actualizando_datos
  BEFORE UPDATE ON proyecto
  FOR EACH ROW
  BEGIN
    CASE
      WHEN new.tipoProy = 'Educacion' OR new.tipoProy = 'Forestacion' OR new.tipoProy = 'Cultura'
      THEN
        SET new.estado = 'Activo';
      ELSE
        SET new.estado = 'Inactivo';
    END CASE;
  END;
```






	id_proy	nombreProy	tipoProy	estado
1	1	ProyectoA	Educacion	Activo
2	2	ProyectoB	Forestacion	Activo
3	3	ProyectoC	Cultura	Activo
4	4	ProyectoD	Deportivo	Inactivo
5	5	ProyectoE	Forestacion	Activo

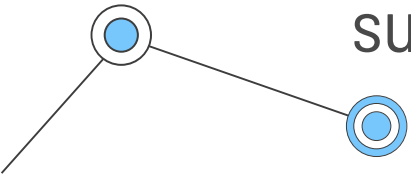
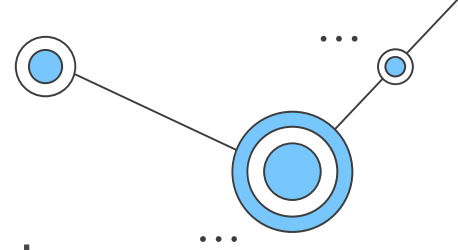
```
5 SELECT datos.*  
6 FROM proyecto AS datos;  
7  
8 INSERT INTO proyecto (id_proy, nombreProy, tipoProy)  
9 VALUES (6, 'ProyectoMinicipal', 'Educacion');
```

	id_proy	nombreProy	tipoProy	estado
1	1	ProyectoA	Educacion	Activo
2	2	ProyectoB	Forestacion	Activo
3	3	ProyectoC	Cultura	Activo
4	4	ProyectoD	Deportivo	Inactivo
5	5	ProyectoE	Forestacion	Activo
6	6	ProyectoMinicipal	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.



# Manejo de Triggers II

```
✓ CREATE TRIGGER calculaEdad
  BEFORE INSERT ON persona
  FOR EACH ROW
  BEGIN
    DECLARE year INT DEFAULT 0;
    DECLARE year_actual INT DEFAULT 0;

    SET
      year = (SELECT MAX(SUBSTR(new.fecha_nac, 1, 4))
      FROM persona AS per);

    SET
      year_actual = (SELECT substr(curdate(), 1, 4));
    SET
      new.edad = year_actual - year;
  END;
```

# Manejo de Triggers II

```
✓ INSERT INTO persona(id_per, nombre, apellidos, fecha_nac, email, sexo, id_dep, id_prov)  
VALUES (8,'Matias', 'Gonzalez','1985-10-06', 'matias123@gmail.com', 'm', 1, 3);
```

```
✓ SELECT per.*  
FROM persona AS per;
```

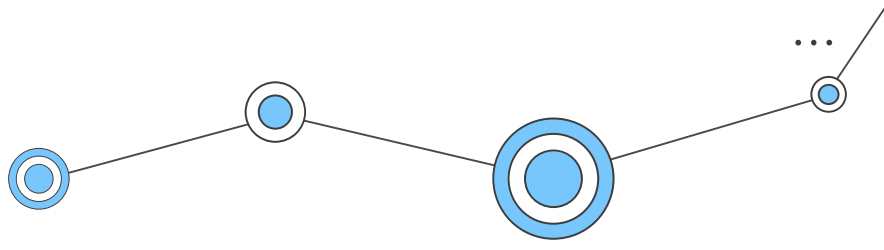
	id_per	nombre	apellidos	fecha_nac	edad	email	sexo	id_dep	id_prov
1	1	nombre1	apellido1	2000-11-18	18	nombre1@gmail.com	f	1	1
2	2	nombre2	apellido2	2000-05-08	21	nombre2@gmail.com	m	2	2
3	3	nombre3	apellido3	2000-06-10	24	nombre3@gmail.com	f	3	3
4	4	nombre4	apellido4	2000-10-10	28	nombre4@gmail.com	f	4	4
5	5	nombre5	apellido5	2000-10-10	32	nombre5@gmail.com	f	5	5
6	6	nombre6	apellido6	2000-10-10	48	nombre6@gmail.com	f	3	3
7	7	nombre7	apellido7	2012-06-19	10	apellidosnom7@gmail.com	m	2	1
8	8	Matias	Gonzalez	1985-10-06	37	matias123@gmail.com	m	1	3





# 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.
- 



# Creamos la Tabla



```
✓ CREATE TABLE datos_persona
(
    nombre VARCHAR (50),
    apellidos VARCHAR (50),
    fecha_nac DATE,
    edad INT,
    email VARCHAR (50),
    sexo CHAR
);
```

...

...



...

```
✓ CREATE TRIGGER guardar_datos
  BEFORE INSERT ON persona
  FOR EACH ROW
  BEGIN
    INSERT INTO datos_persona (nombre, apellidos, fecha_nac, edad, email, sexo)
    VALUES (new.nombre, new.apellidos, new.fecha_nac, new.edad, new.email, new.sexo);
  END;
```

```
✓ INSERT INTO persona(nombre, apellidos, fecha_nac, edad, email, sexo, id_dep, id_prov)
VALUES ('Julio', 'Ruiz', '1999-10-10', 23, 'Julios123@gmail.com', 'm', 1, 1);
```

WHERE		ORDER BY				
	nombre	apellidos	fecha_nac	edad	email	sexo
1	Julio	Ruiz	1999-10-10	23	Julios123@gmail.com	m

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

```
✓ CREATE OR REPLACE VIEW Cadenas AS
SELECT concat(per.nombre, ' ', per.apellidos) AS Nombre_Completo,
       depa.nombre, pro.nombre AS Nombre_Proyecto, concat(proy.nombreProy, ' - ', proy.tipoProy) AS Proyecto
FROM departamento AS depa
INNER JOIN provincia AS pro ON depa.id_dep = pro.id_dep
INNER JOIN persona AS per ON depa.id_dep = per.id_dep
INNER JOIN detalle_proyecto AS dep ON per.id_per = dep.id_per
INNER JOIN proyecto AS proy ON dep.id_proy = proy.id_proy;
```

	Nombre_Completo	nombre	Nombre_Proyecto	Proyecto
1	nombre1 apellido1	La Paz	provincia1	ProyectoA - Educacion
2	nombre2 apellido2	El Alto	provincia2	ProyectoB - Forestacion
3	nombre3 apellido3	Santa Cruz	provincia3	ProyectoC - Cultura
4	nombre4 apellido4	Oruro	provincia4	ProyectoD - Deportivo
5	nombre5 apellido5	El Alto	provincia5	ProyectoE - Forestacion



**Gracias por su  
Atencion**

