

# Trabajo Práctico 1

## Base de datos II

### Fundamentos, Integridad y Concurrency

#### Ejercicio 1:

En el siguiente ejemplo si queremos eliminar un alumno de la tabla “Alumnos” y este pertenece a una Asignatura, generaría que estas tengan un id ya no existente. Para evitar esto usamos “ON DELETE RESTRICT” para que no se pueda eliminar un alumno que tenga asignaturas asociadas. También podríamos usar “ON DELETE CASCADE” para que también se elimine la asignatura referente al alumno eliminado.

##### Schema SQL

```
1 CREATE TABLE Alumnos (  
2   id INT AUTO_INCREMENT PRIMARY KEY,  
3   nombre VARCHAR(50),  
4   edad INT  
5 );  
6  
7 CREATE TABLE Asignaturas (  
8   id INT AUTO_INCREMENT PRIMARY KEY,  
9   id_alumno INT,  
10  asignatura VARCHAR(50),  
11  FOREIGN KEY (id_alumno) REFERENCES Alumnos(id)  
12  ON DELETE RESTRICT  
13 );  
14
```

Text to DDL

##### Query SQL

```
1 INSERT INTO Alumnos (nombre, edad) VALUES  
2 ('Ramiro', 26),  
3 ('Lucía', 23),  
4 ('Carlos', 24);  
5  
6 INSERT INTO Asignaturas (id_alumno, asignatura) VALUES  
7 (1, 'Matemática'),  
8 (1, 'Programación'),  
9 (2, 'Física'),  
10 (3, 'Historia');  
11  
12 DELETE FROM Alumnos WHERE id = 1;
```

##### Results

Query Error: Cannot delete or update a parent row: a foreign key constraint fails (`test`.`Asignaturas`, CONSTRAINT `Asignaturas\_ibfk\_1` FOREIGN KEY (`id\_alumno`) REFERENCES `Alumnos` (`id`))

#### Ejercicio 2:

En el siguiente ejemplo agregamos la Matrícula la cual asigna una asignatura a un alumno existente. En este caso tratamos de agregar un alumno inexistente (99), a la matrícula (2) que es Historia, esto resulta en un error gracias a la restricción de la clave foránea.

Error: Query Error: Cannot add or update a child row: a foreign key constraint fails (`test`.`Matricula`, CONSTRAINT `Matricula\_ibfk\_1` FOREIGN KEY (`id\_alumno`) REFERENCES `Alumnos` (`id`) ON DELETE CASCADE)

Schema SQL

```
1 CREATE TABLE Alumnos (  
2   id INT AUTO_INCREMENT PRIMARY KEY,  
3   nombre VARCHAR(50),  
4   edad INT  
5 );  
6  
7 CREATE TABLE Asignaturas (  
8   id INT AUTO_INCREMENT PRIMARY KEY,  
9   asignatura VARCHAR(50)  
10 );  
11  
12 CREATE TABLE Matricula (  
13   id INT AUTO_INCREMENT PRIMARY KEY,  
14   id_alumno INT,  
15   id_asignatura INT,  
16   fecha DATE,  
17   FOREIGN KEY (id_alumno) REFERENCES Alumnos(id) ON DELETE CASCADE,  
18   FOREIGN KEY (id_asignatura) REFERENCES Asignaturas(id) ON DELETE CASCADE  
19 );
```

Text to DDL

Query SQL

```
1 INSERT INTO Alumnos (nombre, edad) VALUES  
2 ('Ramiro', 26),  
3 ('Lucía', 23),  
4 ('Carlos', 24);  
5  
6  
7 INSERT INTO Asignaturas (asignatura) VALUES  
8 ('Matemática'),  
9 ('Historia'),  
10 ('Programación');  
11  
12 INSERT INTO Matricula (id_alumno, id_asignatura) VALUES  
13 (1, 1),  
14 (1, 3),  
15 (2, 2),  
16 (3, 1);  
17  
18 INSERT INTO Matricula (id_alumno, id_asignatura) VALUES (99, 1);
```

Results

Query Error: Cannot add or update a child row: a foreign key constraint fails ('test`.`Matricula`, CONSTRAINT `Matricula\_ibfk\_1` FOREIGN KEY (`id\_alumno`) REFERENCES `Alumnos` (`id`) ON DELETE CASCADE)

## Ejercicio 3:

En el siguiente ejemplo podemos observar 2 transacciones que ocurrirían al mismo tiempo, una está mostrando el saldo de una cuenta, mientras que la otra lo está modificando. Si usamos el nivel de aislamiento READ COMMITTED, la primera transacción podría leer el valor 1000, y si vuelve a hacer el mismo SELECT más tarde en la misma transacción, podría obtener el valor 500 si la segunda transacción ya hizo commit.

Schema SQL

```
1 CREATE TABLE cuentaBancaria (  
2   id INT AUTO_INCREMENT PRIMARY KEY,  
3   nombreCuenta VARCHAR(50),  
4   saldo DECIMAL(10,2)  
5 );  
6
```

Text to DDL

Query SQL

```
1 -- Sesion 1  
2  
3 SET TRANSACTION ISOLATION LEVEL READ COMMITTED;  
4  
5 START TRANSACTION;  
6  
7 Select saldo FROM cuentaBancaria WHERE nombre = 'Ramiro'  
8  
9 -- Resultado esperado 1000  
10  
11 -- Sesion 2  
12  
13 SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;  
14  
15 START TRANSACTION;  
16  
17 UPDATE cuentaBancaria SET saldo = saldo - 500 WHERE nombre =  
   'Ramiro';
```

En el siguiente ejemplo podemos observar las mismas transacciones pero esta vez usando el nivel de aislamiento SERIALIZABLE, esto hace que se ejecuten de manera serializada (Una detrás de otra) por lo tanto el UPDATE de la Sesión 2 no podrá continuar mientras la transacción de la sección 1 este abierta. Tiene que esperar a que esta realice COMMIT.

Este nivel es el más estricto ya que las transacciones se ejecutan de manera aislada, evitando lecturas fantasma.

#### Schema SQL ●

```
1 CREATE TABLE cuentaBancaria (  
2   id INT AUTO_INCREMENT PRIMARY KEY,  
3   nombreCuenta VARCHAR(50),  
4   saldo DECIMAL(10,2)  
5 );  
6
```

#### Query SQL ●

```
1 -- Sesion 1  
2 SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
3  
4 START TRANSACTION;  
5  
6 Select saldo FROM cuentaBancaria WHERE nombre = 'Ramiro'  
7  
8 -- Resultado: 1000  
9  
10 -- Sesion 2  
11 COMMIT;  
12  
13 SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
14  
15 START TRANSACTION;  
16  
17 UPDATE cuentaBancaria SET saldo = saldo - 500 WHERE nombre =  
   'Ramiro';  
18 -- Espera a que Sesion 1 termine
```

## Ejercicio 4:

En este ejercicio lo primero que hacemos es crear una base de datos “Personas” e le insertamos varios registros:

```
Query 1 x  
Limit to 1000 rows  
1 create table personas(  
2     nombre varchar(50),  
3     edad int,  
4     deporte varchar(50)  
5 );
```

Luego realizamos una consulta donde quiero que me devuelva las personas con más de 25 años, primero realizamos esta consulta sin índice, después creamos el índice y realizamos nuevamente la misma consulta consulta:

✓	58	21:34:04	select * from personas where edad > 25 LIMIT 0, 1000	15 row(s) returned	0.00057 sec / 0.000007...
✓	59	21:34:11	create index idx_edad on personas(edad)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.133 sec
✓	60	21:34:19	select * from personas where edad > 25 LIMIT 0, 1000	15 row(s) returned	0.00050 sec / 0.000009...

Podemos notar que la consulta que realizamos con índice es minimamente más rápida que la consulta sin índice.

## Ejercicio 5:

En este ejercicio vamos crear primero una base de datos para guardar juegos: sus nombre, género, valoraciones, y fecha de salida:

```
SQL File 6* x
1 • create table Juegos(
2     nombre varchar(50),
3     genero varchar(50),
4     valoracion float,
5     fechaSalida date
6 );
```

Le insertamos a la tabla varios registros para luego realizar una consulta, donde básicamente quiero que me devuelva los nombres de los juegos que sean del género "Aventura" y que su valoración sea mayor a 9.0.

Primero vamos a realizar esta consulta sin utilizar índices, para luego crearlos y comparar el rendimiento de ambas consultas

66	22:06:28	select nombre as Juegos from Juegos where genero = 'Aventura' and ...	2 row(s) returned	0.0055 sec / 0.000091 ...
67	22:08:03	create index idx_genero on Juegos(genero)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.273 sec
68	22:08:04	create index idx_valoracion on Juegos(valoracion)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.197 sec
69	22:08:20	select nombre as Juegos from Juegos where genero = 'Aventura' and ...	2 row(s) returned	0.00042 sec / 0.000006...

Como podemos ver la consulta que realizamos con índice es minimamente más rápida que la consulta sin índice.

## Ejercicio 6:

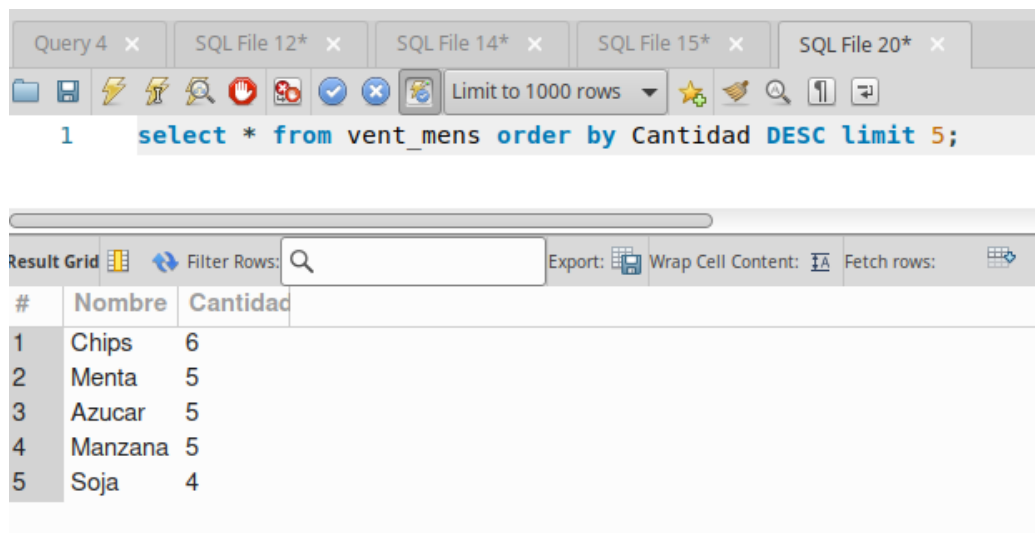
Para este ejercicio, primero creamos dos tablas, una para registrar los "Productos": sus nombre y precios, y otra tabla para registrar las "Ventas" de los productos a partir de su ID:

```
Query 4 x
1 • create table Productos(
2     id int auto_increment primary key,
3     nombre varchar(50),
4     precio float
5 );
6 • create table Ventas(
7     id int auto_increment primary key,
8     id_producto int,
9     foreign key(id_producto) references Productos(id)
10 );
```

Luego creamos una vista para ver las ventas mensuales de cada producto registrado:

```
Query 4 x SQL File 12* x SQL File 14* x SQL File 15* x SQL File 19* x
1 • create view vent_mens as select nombre as Nombre, count(*) as Cantidad from Productos as P inner join Ventas as V on P.id = V.id_producto group by Nombre;
```

Después utilizamos la vista que creamos anteriormente para usarla dentro de una consulta para que nos devuelva los primero 5 productos más vendidos:



The screenshot shows a SQL IDE interface. At the top, there are tabs for 'Query 4', 'SQL File 12\*', 'SQL File 14\*', 'SQL File 15\*', and 'SQL File 20\*'. Below the tabs is a toolbar with various icons, including a 'Limit to 1000 rows' dropdown. The main text area contains the following SQL query:

```
1 select * from vent_mens order by Cantidad DESC limit 5;
```

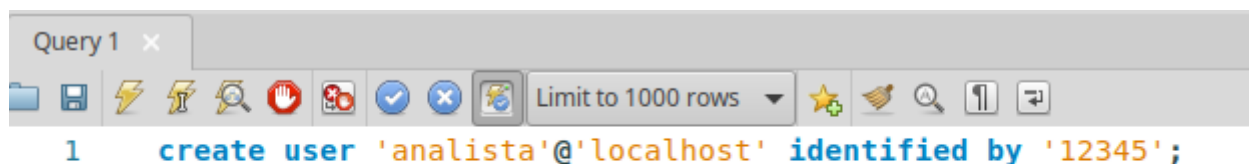
Below the query editor, the 'Result Grid' is displayed. It has a search bar and buttons for 'Export', 'Wrap Cell Content', and 'Fetch rows'. The results are shown in a table with three columns: '#', 'Nombre', and 'Cantidad'.

#	Nombre	Cantidad
1	Chips	6
2	Menta	5
3	Azucar	5
4	Manzana	5
5	Soja	4

### Ejercicio 7:

En este ejercicio, primero vamos a crear un usuario llamado “Analista” dónde vamos darle a través del usuario root, la función de realizar “Select” en la base de datos que queramos.

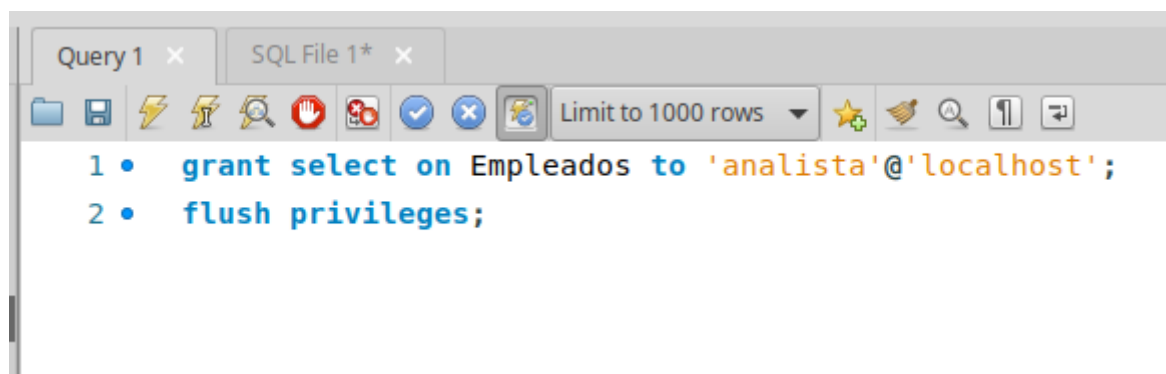
Primero creamos el usuario “Analista”:



The screenshot shows a SQL IDE interface with a tab for 'Query 1'. The main text area contains the following SQL query:

```
1 create user 'analista'@'localhost' identified by '12345';
```

Segundo le damos privilegios (solo la función select en la base de datos llamada “Empleados”) y aplicamos los cambios:



The screenshot shows a SQL IDE interface with tabs for 'Query 1' and 'SQL File 1\*'. The main text area contains the following SQL queries:

```
1 • grant select on Empleados to 'analista'@'localhost';  
2 • flush privileges;
```

y Tercero, a la hora que el usuario analista quiera hacer una función que no tiene permitida sucede esto:

The screenshot shows a database query editor with a toolbar at the top. The query window contains the following SQL statement:

```
1 insert into Empleados(nombre, edad, puesto)
2 value('pepe', 28, 'Marketing');
```

Below the query window is the "Action Output" panel, which displays a table of execution results:

#	Time	Action	Message	Duration
1	15:52:34	use Ora	Error Code: 1044. Access denied for user 'analista'@'localhost'...	0.001236
2	15:52:40	select * from Empleados LIMIT 0, 1000	Error Code: 1046. No database selected Select the default DB to be used by double-clicking its name...	0.00021
3	15:52:53	use Empresa	0 row(s) affected	0.00018
4	15:52:59	select * from Empleados LIMIT 0, 1000	4 row(s) returned	0.00042
5	15:54:47	insert into Empleados(nombre, edad, puesto) value('pepe', 28, 'Mark...	Error Code: 1142. INSERT command denied to user 'analista'...	0.0030

Le tira un error, diciendo que la función de insertar datos en una tabla no la tiene permitida.

### Ejercicio 8:

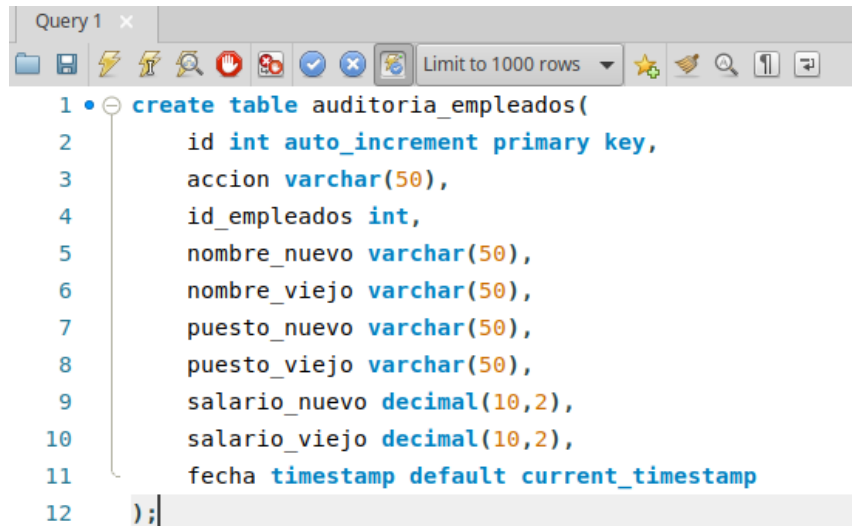
En este ejercicio para poder simular una auditoría con triggers vamos a hacerlo en varios pasos.

Primero creamos una tabla, en este caso una tabla “empleados” donde guardaremos datos de empleados:

The screenshot shows a database query editor with a toolbar at the top. The query window contains the following SQL statement:

```
1 create table Empleados(
2     id int auto_increment primary key,
3     nombre varchar(50),
4     puesto varchar(50),
5     salario decimal(10,2)
6 );
```

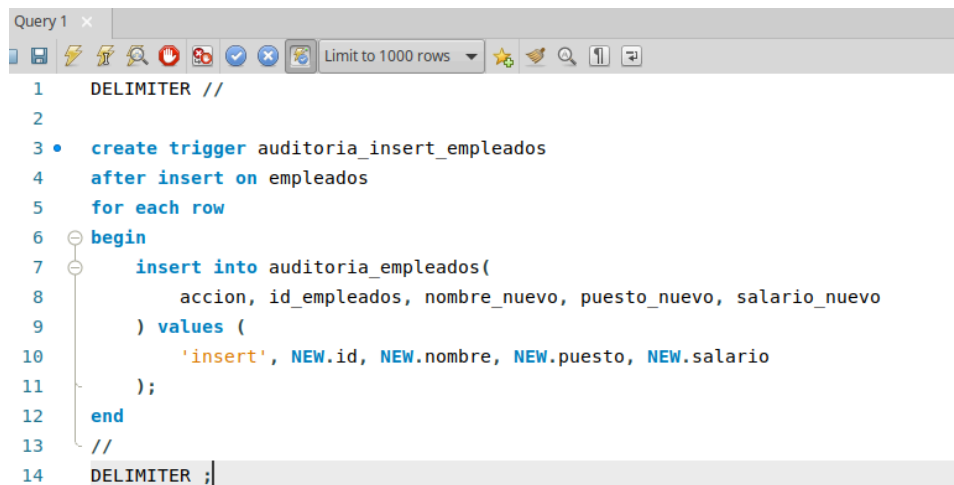
Luego creamos otra tabla llamada “auditoria\_empleados”, donde guardaremos los registro de los cambios que se van a ir haciendo en la tabla empleados:



```
Query 1 x
Limit to 1000 rows
1 • create table auditoria_empleados(
2     id int auto_increment primary key,
3     accion varchar(50),
4     id_empleados int,
5     nombre_nuevo varchar(50),
6     nombre_viejo varchar(50),
7     puesto_nuevo varchar(50),
8     puesto_viejo varchar(50),
9     salario_nuevo decimal(10,2),
10    salario_viejo decimal(10,2),
11    fecha timestamp default current_timestamp
12 );
```

Después creamos tres triggers, que nos van a ayudar a registrar los cambios que suceden en la tabla “empleados”, ya sea agregando nuevos empleados, actualizando empleados o eliminando empleados:

+Agregar empleados:



```
Query 1 x
Limit to 1000 rows
1 DELIMITER //
2
3 • create trigger auditoria_insert_empleados
4 after insert on empleados
5 for each row
6 begin
7     insert into auditoria_empleados(
8         accion, id_empleados, nombre_nuevo, puesto_nuevo, salario_nuevo
9     ) values (
10         'insert', NEW.id, NEW.nombre, NEW.puesto, NEW.salario
11     );
12 end
13 //
14 DELIMITER ;
```

## +Actualizar empleados:

```
Query 1 x
Limit to 1000 rows

1 DELIMITER //
2
3 • create trigger auditoria_update_empleados
4 after update on empleados
5 for each row
6 begin
7     insert into auditoria_empleados(
8         accion, id_empleados,
9         nombre_viejo ,nombre_nuevo,
10        puesto_viejo ,puesto_nuevo,
11        salario_viejo ,salario_nuevo
12    ) values (
13        'update', OLD.id,
14        OLD.nombre,NEW.nombre,
15        OLD.puesto ,NEW.puesto,
16        OLD.salario ,NEW.salario
17    );
18 end
19 //
```

## +Eliminar empleados:

```
Query 1 x
Limit to 1000 rows

1 DELIMITER //
2
3 • create trigger auditoria_delete_empleados
4 after delete on empleados
5 for each row
6 begin
7     insert into auditoria_empleados(
8         accion, id_empleados,
9         nombre_viejo ,puesto_viejo ,salario_viejo
10    ) values (
11        'delete', OLD.id,
12        OLD.nombre, OLD.puesto ,OLD.salario
13    );
14 end
15 //
16 DELIMITER ;
```

Luego de crear las tablas y triggers vamos a probar si funciona.

A la hora de agregar un usuario en la tabla “Empleados”:

```
Query 1 x    SQL File 4* x
Limit to 1000 rows

1 • insert into empleados(nombre, puesto, salario) values
2    ('Mateo', 'Programador', 120654.5);
```



Query 1 x SQL File 4\* x

Limit to 1000 rows

```
1 • select * from auditoria_empleados;
```

Result Grid

#	id	accion	id_empleados	nombre_nuevo	nombre_viejo	puesto_nuevo	puesto_viejo	salario_nuevo	salario_viejo	fecha
1	1	insert	1	Mateo	NULL	Programador	NULL	120654.50	NULL	2025-04-23 21:00:08
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Podemos ver que en la tabla de “auditoria\_empleados” queda registrado el “insert” que hicimos anteriormente.

Cuando actualizamos un empleados en la tabla “Empleados”:

Query 1 x SQL File 4\* x

Limit to 1000 rows

```
1 • update empleados set puesto = 'Analista' where id = 1;
```

Query 1 x SQL File 4\* x

Limit to 1000 rows

```
1 • select * from auditoria_empleados;
```

Result Grid

#	id	accion	id_empleados	nombre_nuevo	nombre_viejo	puesto_nuevo	puesto_viejo	salario_nuevo	salario_viejo	fecha
1	1	insert	1	Mateo	NULL	Programador	NULL	120654.50	NULL	2025-04-23 21:00:08
2	2	update	1	Mateo	Mateo	Analista	Programador	120654.50	120654.50	2025-04-23 21:04:14
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Podemos notar que en la tabla “auditoria\_empleados” queda registrado el “update” que hicimos anteriormente.

Y Cuando queremos eliminar un empleado en la tabla “Empleados”:

Query 1 x SQL File 4\* x SQL File 5\* x

Limit to 1000 rows

```
1 • delete from empleados where id = 1
```

Query 1 x SQL File 4\* x SQL File 5\* x

Limit to 1000 rows

1 • `select * from auditoria_empleados;`

#	id	accion	id_empleado	nombre_nuevo	nombre_viejo	puesto_nuevo	puesto_viejo	salario_nuevo	salario_viejo	fecha
1	1	insert	1	Mateo	NULL	Programador	NULL	120654.50	NULL	2025-04-23 21:00:08
2	2	update	1	Mateo	Mateo	Analista	Programador	120654.50	120654.50	2025-04-23 21:04:14
3	3	delete	1	NULL	Mateo	NULL	Analista	120654.50	120654.50	2025-04-23 21:06:05
*										

Podemos ver que en la tabla “auditoria\_empleados” queda registrado el “delete” que hicimos recién.

## Ejercicio 9:

Video paso a paso cómo hacer un Backup y Restore de una base de datos desde Powershell.

## [Video](#)