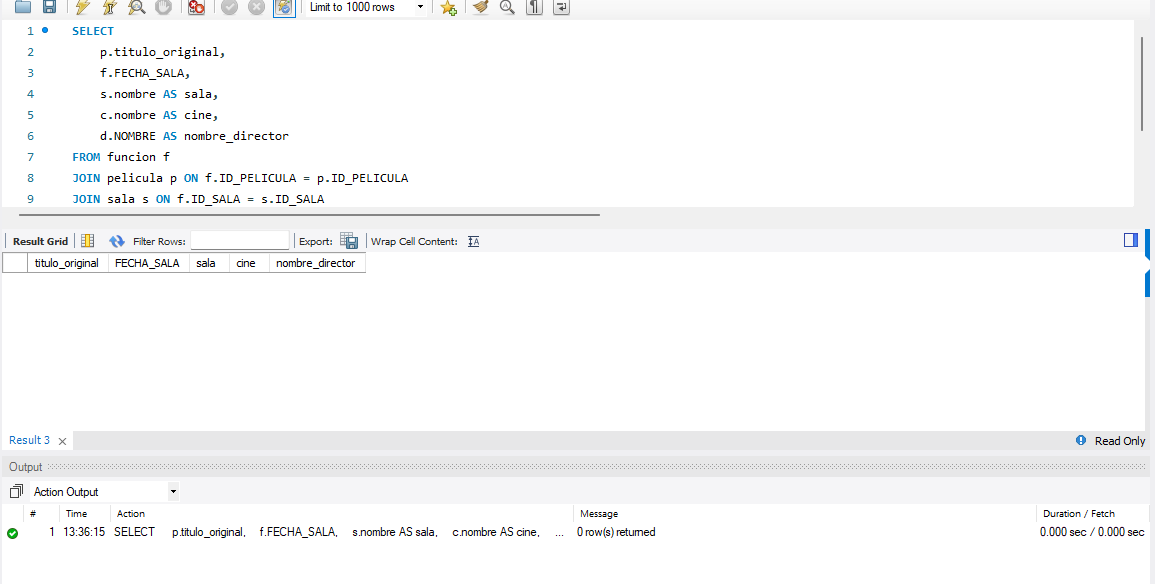
### **1. ¿Qué es una Vista SQL y qué problema resuelve?**

Una **Vista SQL** es una consulta almacenada que se comporta como una tabla virtual. Te permite:

* Reutilizar consultas complejas sin reescribirlas.
* Simplificar el acceso a múltiples tablas.
* Mejorar la seguridad restringiendo columnas visibles.
* Facilitar el mantenimiento y la comprensión del sistema.

**2. Consulta multitabla**

****

SELECT

p.titulo\_original,

f.FECHA\_SALA,

s.nombre AS sala,

c.nombre AS cine,

d.NOMBRE AS nombre\_director

FROM funcion f

JOIN pelicula p ON f.ID\_PELICULA = p.ID\_PELICULA

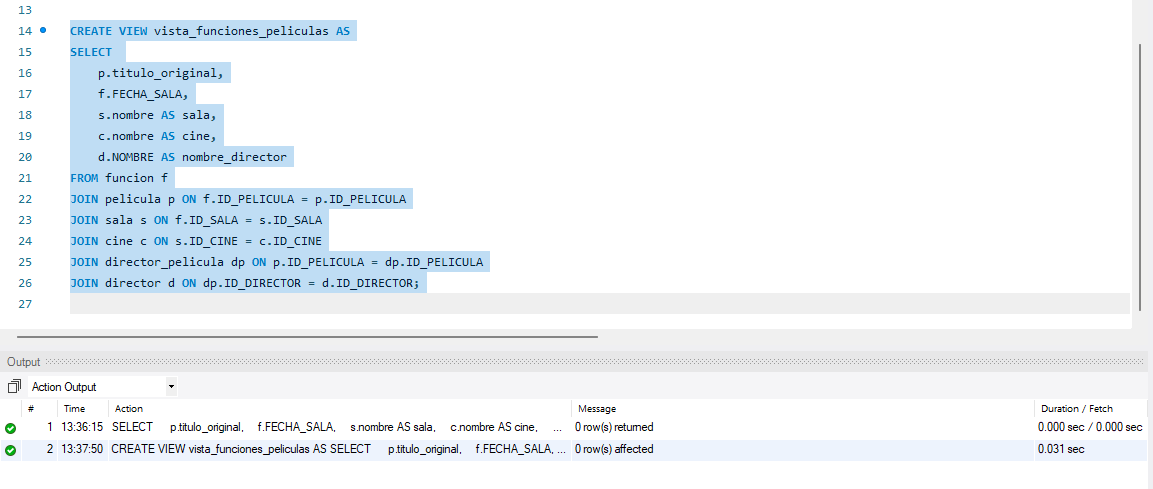
JOIN sala s ON f.ID\_SALA = s.ID\_SALA

JOIN cine c ON s.ID\_CINE = c.ID\_CINE

JOIN director\_pelicula dp ON p.ID\_PELICULA = dp.ID\_PELICULA

JOIN director d ON dp.ID\_DIRECTOR = d.ID\_DIRECTOR;

3. Crear la Vista SQL



CREATE VIEW vista\_funciones\_peliculas AS

SELECT

p.titulo\_original,

f.FECHA\_SALA,

s.nombre AS sala,

c.nombre AS cine,

d.NOMBRE AS nombre\_director

FROM funcion f

JOIN pelicula p ON f.ID\_PELICULA = p.ID\_PELICULA

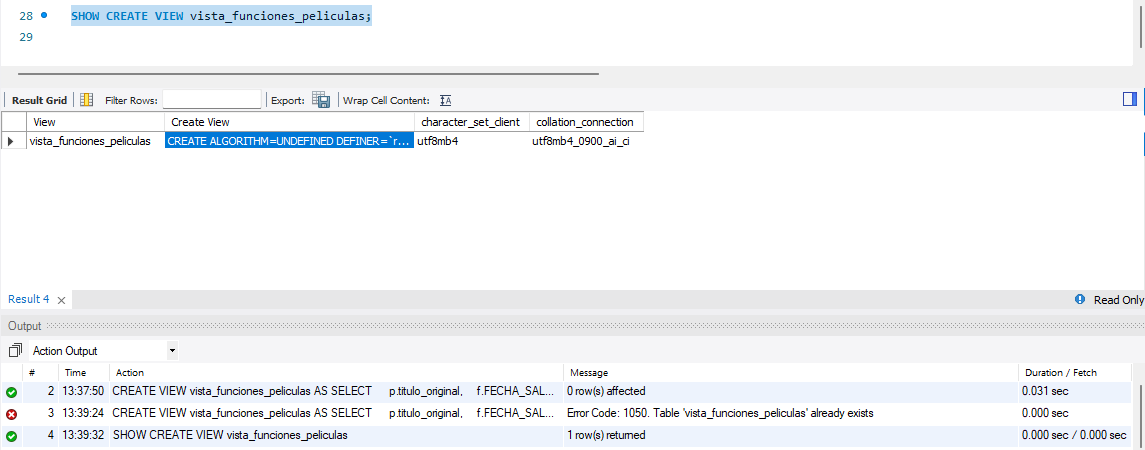
JOIN sala s ON f.ID\_SALA = s.ID\_SALA

JOIN cine c ON s.ID\_CINE = c.ID\_CINE

JOIN director\_pelicula dp ON p.ID\_PELICULA = dp.ID\_PELICULA

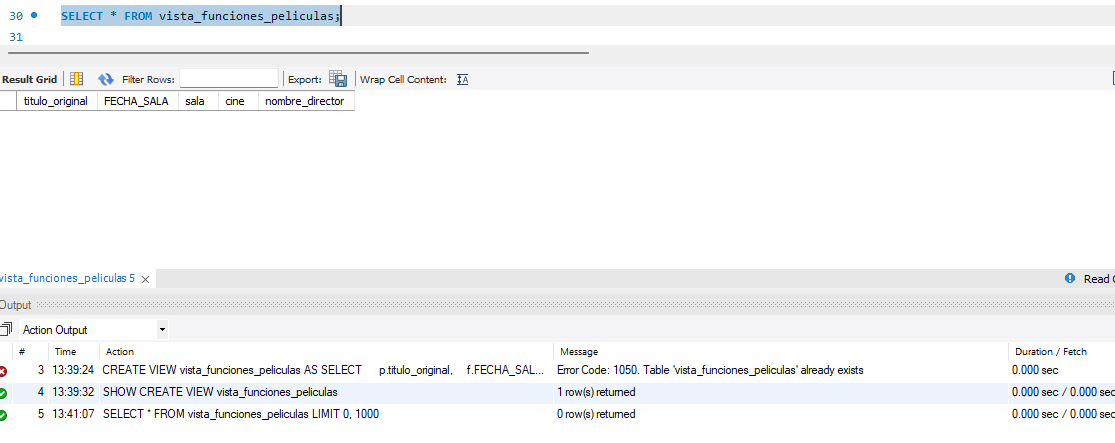
JOIN director d ON dp.ID\_DIRECTOR = d.ID\_DIRECTOR;

4. Mostrar la Vista SQL creada



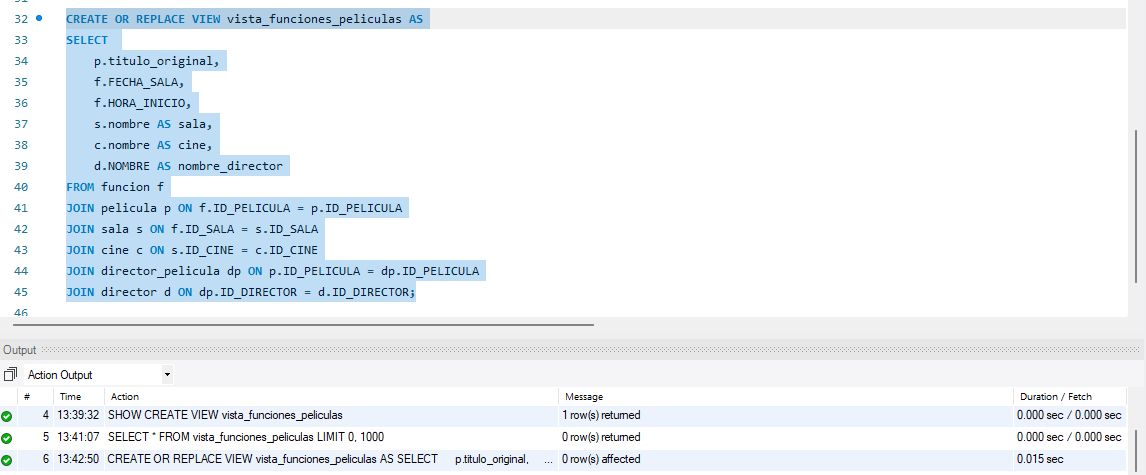
SHOW CREATE VIEW vista\_funciones\_peliculas;

5. Ejecutar la Vista SQL



SELECT \* FROM vista\_funciones\_peliculas;

6. Modificar la Vista SQL



CREATE OR REPLACE VIEW vista\_funciones\_peliculas AS

SELECT

p.titulo\_original,

f.FECHA\_SALA,

f.HORA\_INICIO,

s.nombre AS sala,

c.nombre AS cine,

d.NOMBRE AS nombre\_director

FROM funcion f

JOIN pelicula p ON f.ID\_PELICULA = p.ID\_PELICULA

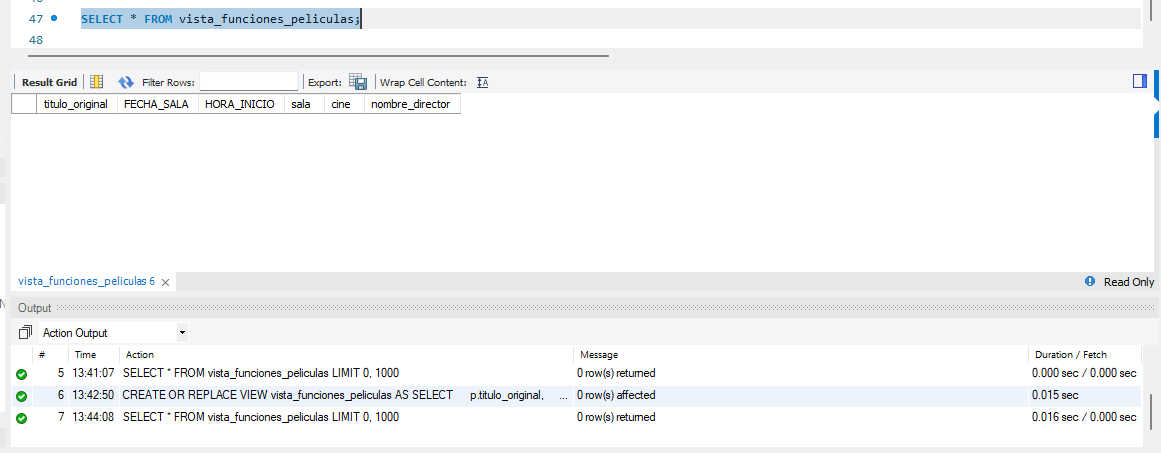
JOIN sala s ON f.ID\_SALA = s.ID\_SALA

JOIN cine c ON s.ID\_CINE = c.ID\_CINE

JOIN director\_pelicula dp ON p.ID\_PELICULA = dp.ID\_PELICULA

JOIN director d ON dp.ID\_DIRECTOR = d.ID\_DIRECTOR;

7. Volver a ejecutar la vista modificada



SELECT \* FROM vista\_funciones\_peliculas;

### **8. Ventajas de las Vistas SQL en MySQL**

* **Reutilización** de consultas complejas.
* **Seguridad**: se puede restringir acceso a ciertas columnas.
* **Facilidad de mantenimiento**: si cambias la lógica, solo cambias la vista.
* Mejora la **legibilidad del código**.

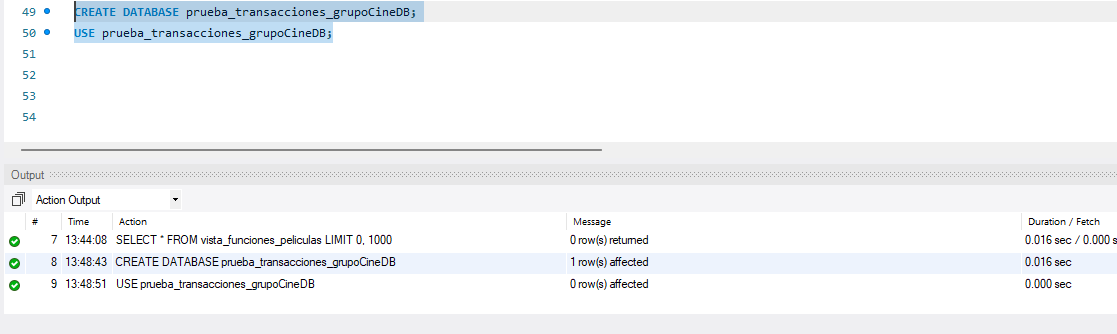
### **9. Desventajas de las Vistas SQL en MySQL**

* No se pueden **indexar** directamente.
* No puedes hacer INSERT, UPDATE o DELETE sobre vistas complejas.
* Las vistas no siempre mejoran el rendimiento, a veces lo empeoran.

### **10. Limitaciones de las Vistas en MySQL**

* No se pueden usar instrucciones como:  
  + ORDER BY (salvo en consultas que usen LIMIT)
  + LIMIT, OFFSET (excepto en subconsultas específicas)
* No se pueden incluir variables ni procedimientos almacenados dentro de una vista.
* Algunas funciones agregadas no están permitidas en vistas actualizables.

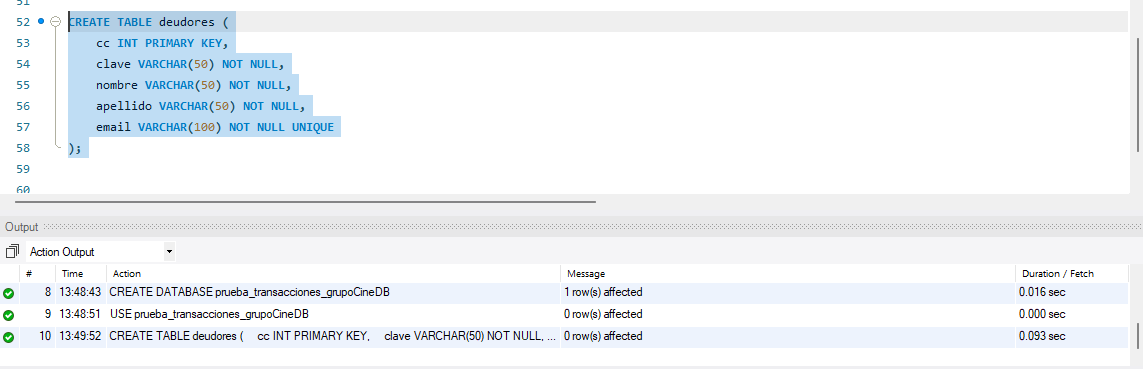
1. Crear la base de datos

****

**CREATE DATABASE prueba\_transacciones\_grupoCineDB;**

**USE prueba\_transacciones\_grupoCineDB;**

**2. Crear tabla deudores**

****

CREATE TABLE deudores (

cc INT PRIMARY KEY,

clave VARCHAR(50) NOT NULL,

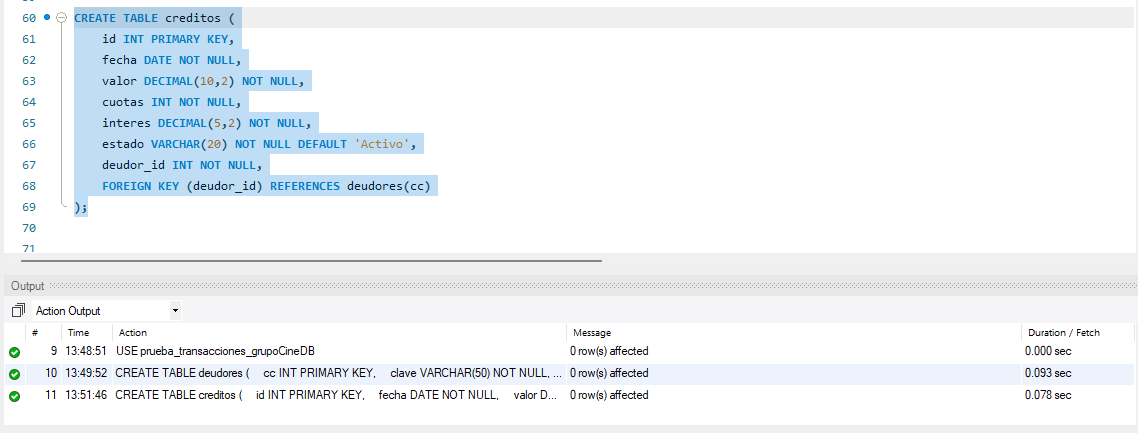
nombre VARCHAR(50) NOT NULL,

apellido VARCHAR(50) NOT NULL,

email VARCHAR(100) NOT NULL UNIQUE

);

3. Crear tabla creditos



CREATE TABLE creditos (

id INT PRIMARY KEY,

fecha DATE NOT NULL,

valor DECIMAL(10,2) NOT NULL,

cuotas INT NOT NULL,

interes DECIMAL(5,2) NOT NULL,

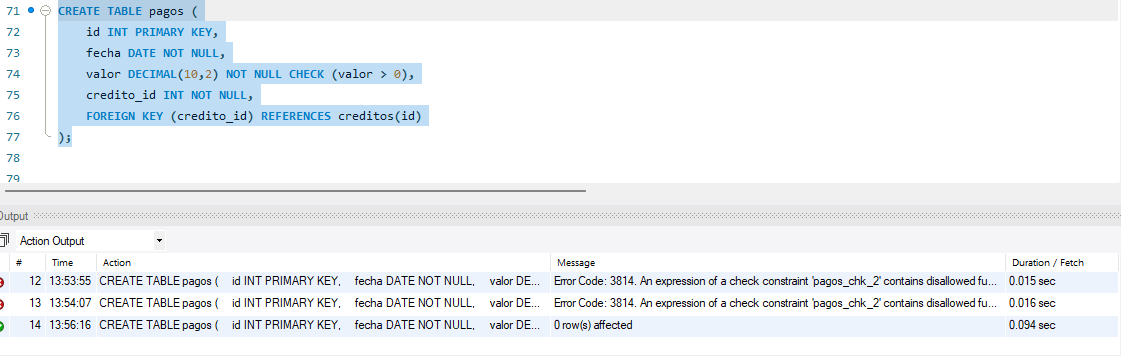
estado VARCHAR(20) NOT NULL DEFAULT 'Activo',

deudor\_id INT NOT NULL,

FOREIGN KEY (deudor\_id) REFERENCES deudores(cc)

);

4. Crear tabla pagos



CREATE TABLE pagos (

id INT PRIMARY KEY,

fecha DATE NOT NULL,

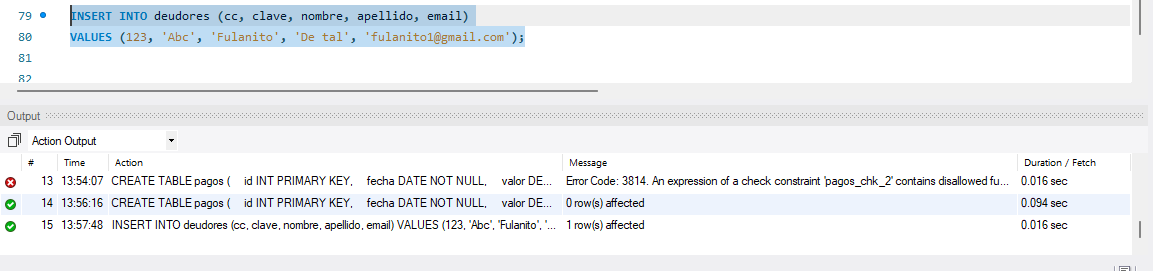
valor DECIMAL(10,2) NOT NULL CHECK (valor > 0),

credito\_id INT NOT NULL,

FOREIGN KEY (credito\_id) REFERENCES creditos(id)

);

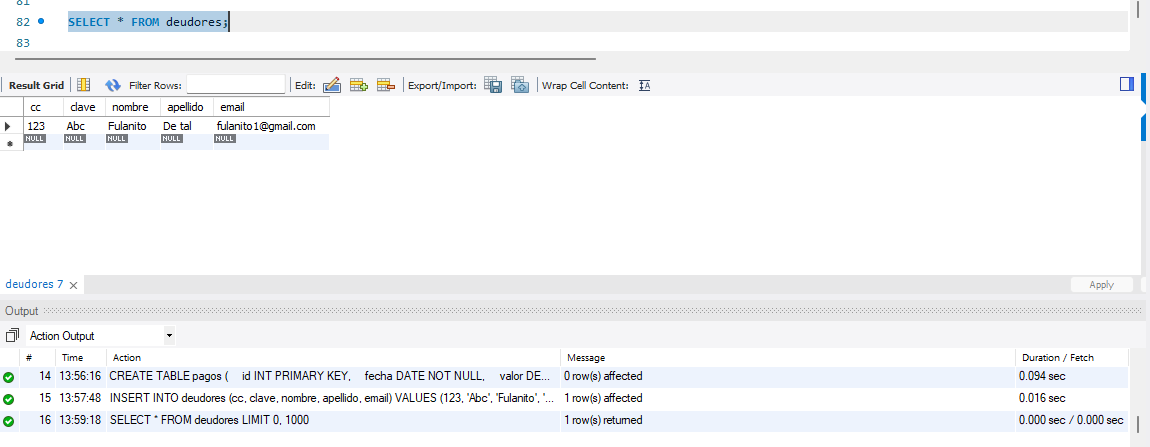
5. Insertar un Deudor



INSERT INTO deudores (cc, clave, nombre, apellido, email)

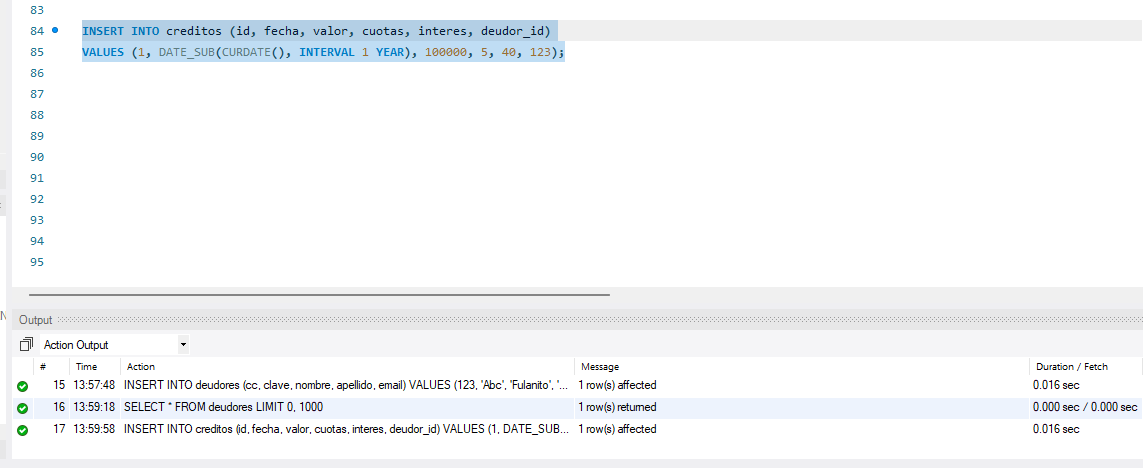
VALUES (123, 'Abc', 'Fulanito', 'De tal', '[fulanito1@gmail.com](mailto:fulanito1@gmail.com)');

6. Mostrar todos los deudores



SELECT \* FROM deudores;

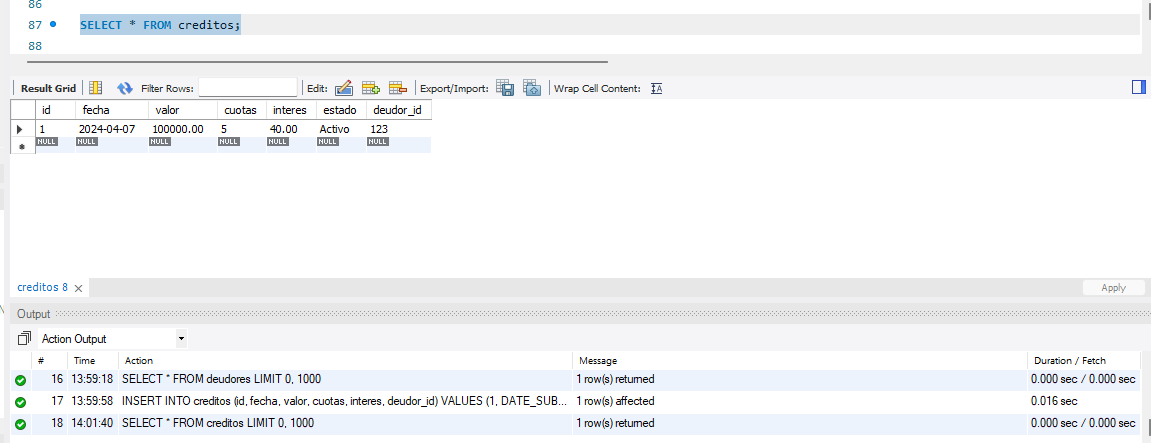
7. Insertar un Crédito



INSERT INTO creditos (id, fecha, valor, cuotas, interes, deudor\_id)

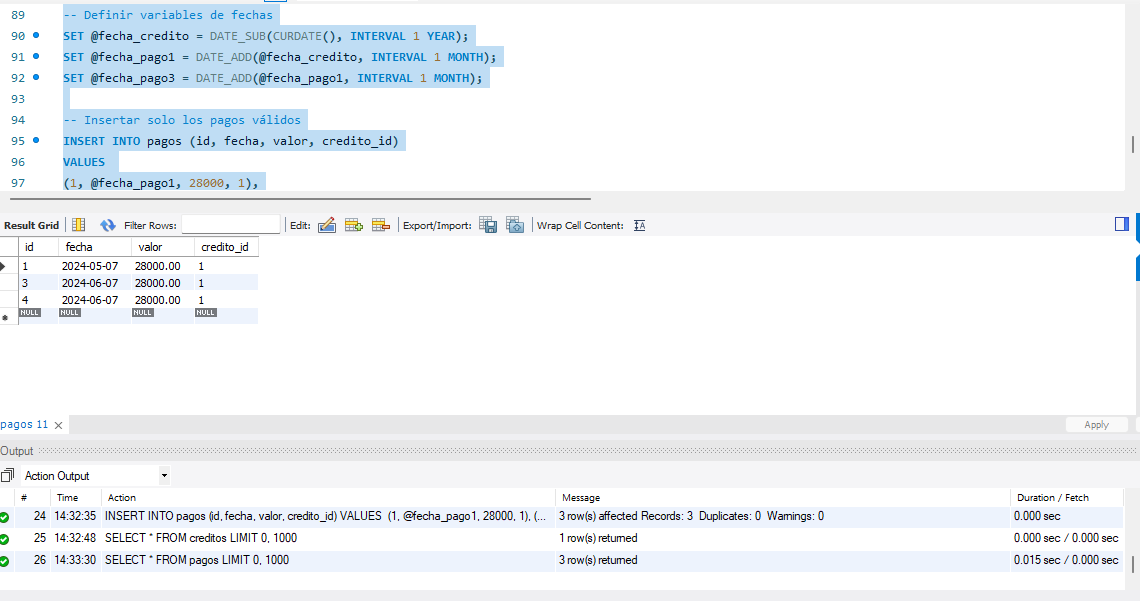
VALUES (1, DATE\_SUB(CURDATE(), INTERVAL 1 YEAR), 100000, 5, 40, 123);

8. Mostrar todos los Créditos



SELECT \* FROM creditos;

9. Insertar 5 pagos (respetando formato de fecha y validaciones)



-- Definir variables de fechas

SET @fecha\_credito = DATE\_SUB(CURDATE(), INTERVAL 1 YEAR);

SET @fecha\_pago1 = DATE\_ADD(@fecha\_credito, INTERVAL 1 MONTH);

SET @fecha\_pago3 = DATE\_ADD(@fecha\_pago1, INTERVAL 1 MONTH);

INSERT INTO pagos (id, fecha, valor, credito\_id)

VALUES

(1, @fecha\_pago1, 28000, 1),

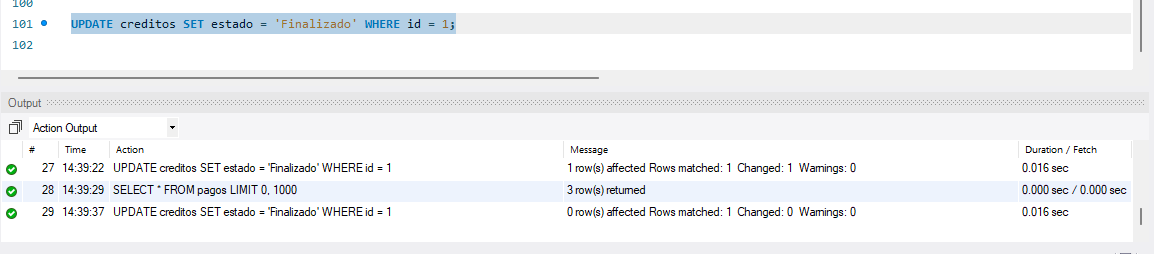
(3, @fecha\_pago3, 28000, 1),

(4, @fecha\_pago3, 28000, 1);

El pago 2 no se puede insertar porque su fecha es a futuro y como la tabla tiene una restricción CHECK que no permite fechas mayores a la actual.

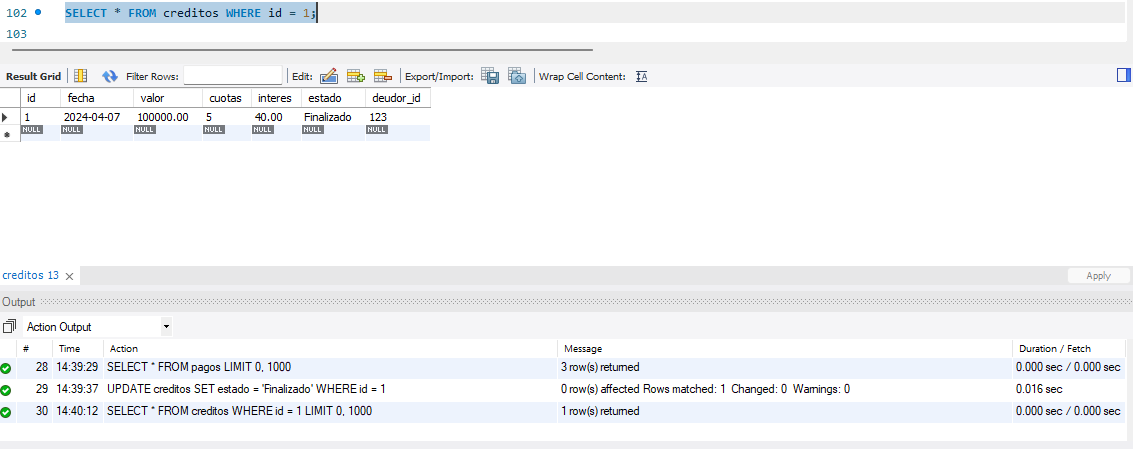
El pago 5 no se puede insertar por su valor negativo y la tabla tiene una restricción CHECK que solo permite número mayores a 0.

10. Cambiar el estado del crédito a Finalizado



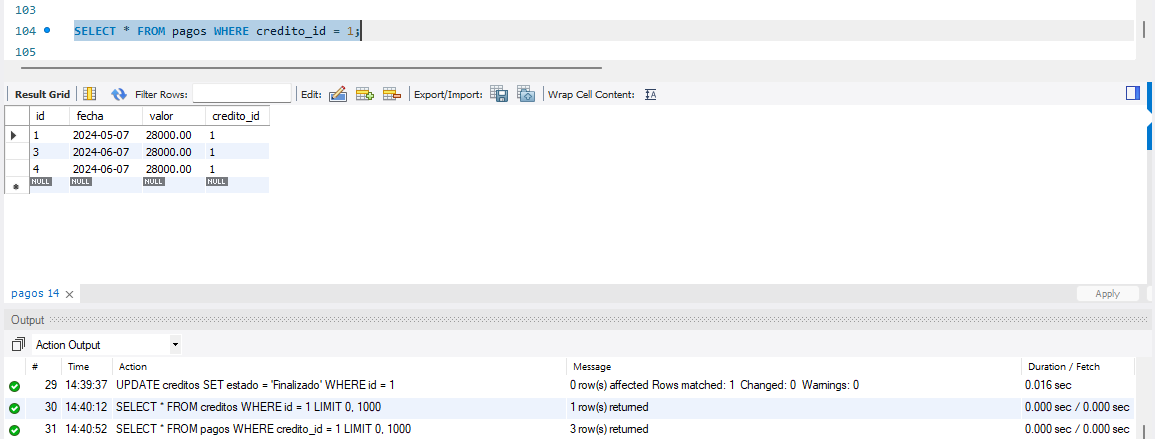
UPDATE creditos SET estado = 'Finalizado' WHERE id = 1;

11. Mostrar datos del crédito con id=1



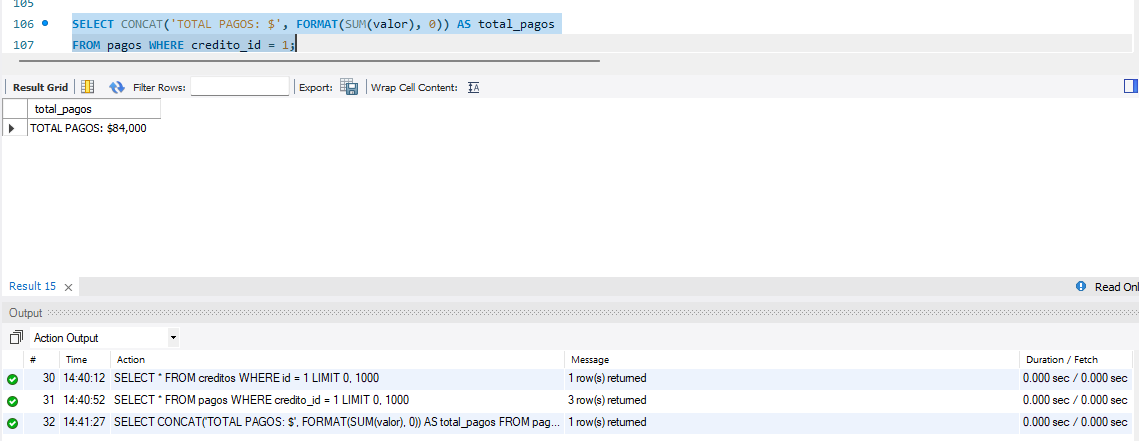
SELECT \* FROM creditos WHERE id = 1;

12. Mostrar todos los pagos para crédito 1



SELECT \* FROM pagos WHERE credito\_id = 1;

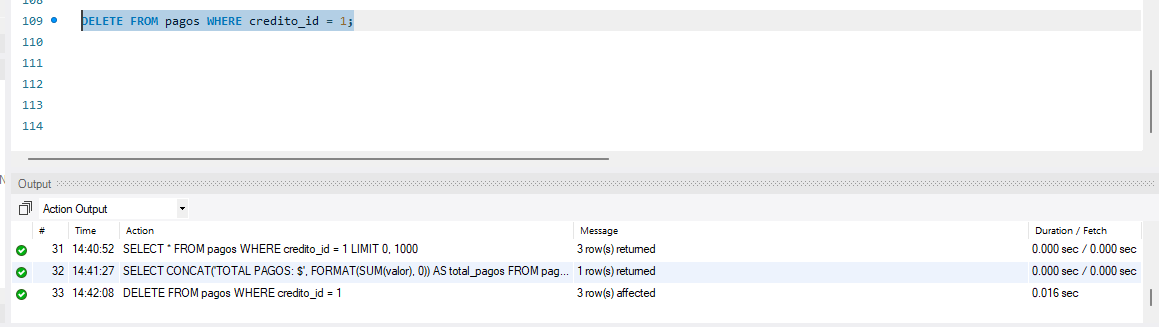
13. Totalizar pagos del crédito 1



SELECT CONCAT('TOTAL PAGOS: $', FORMAT(SUM(valor), 0)) AS total\_pagos

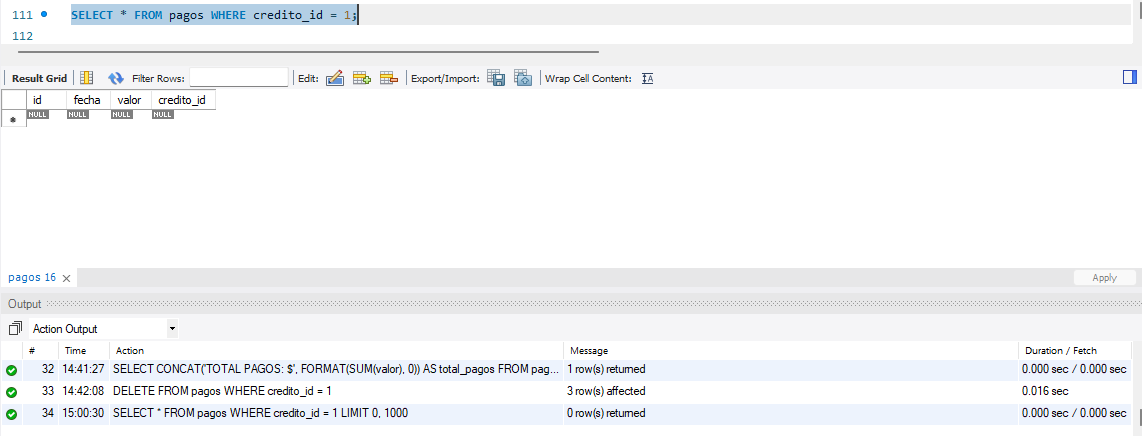
FROM pagos WHERE credito\_id = 1;

14. Eliminar todos los pagos del crédito 1



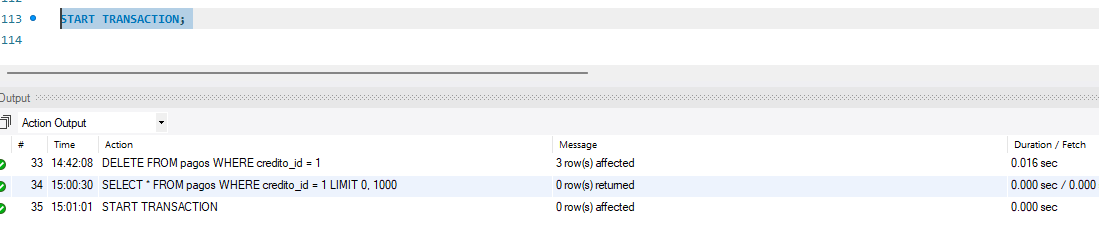
DELETE FROM pagos WHERE credito\_id = 1;

15. Mostrar todos los pagos del crédito 1



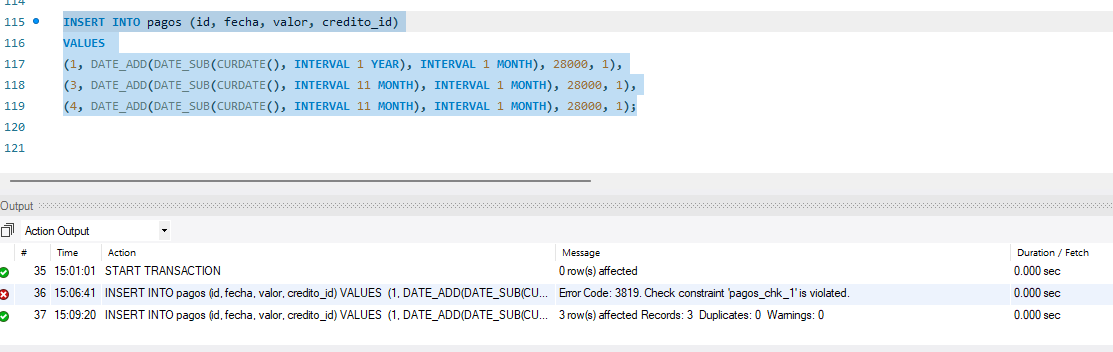
SELECT \* FROM pagos WHERE credito\_id = 1;

16. Iniciar una Transacción



START TRANSACTION;

17. Insertar pagos dentro de la transacción



INSERT INTO pagos (id, fecha, valor, credito\_id)

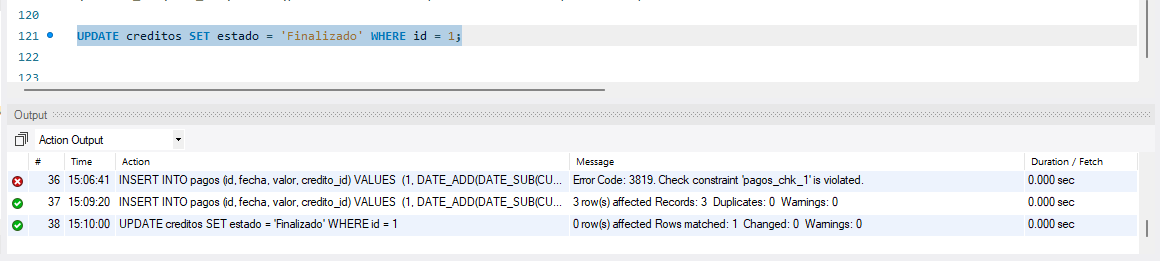
VALUES

(1, DATE\_ADD(DATE\_SUB(CURDATE(), INTERVAL 1 YEAR), INTERVAL 1 MONTH), 28000, 1),

(3, DATE\_ADD(DATE\_SUB(CURDATE(), INTERVAL 11 MONTH), INTERVAL 1 MONTH), 28000, 1),

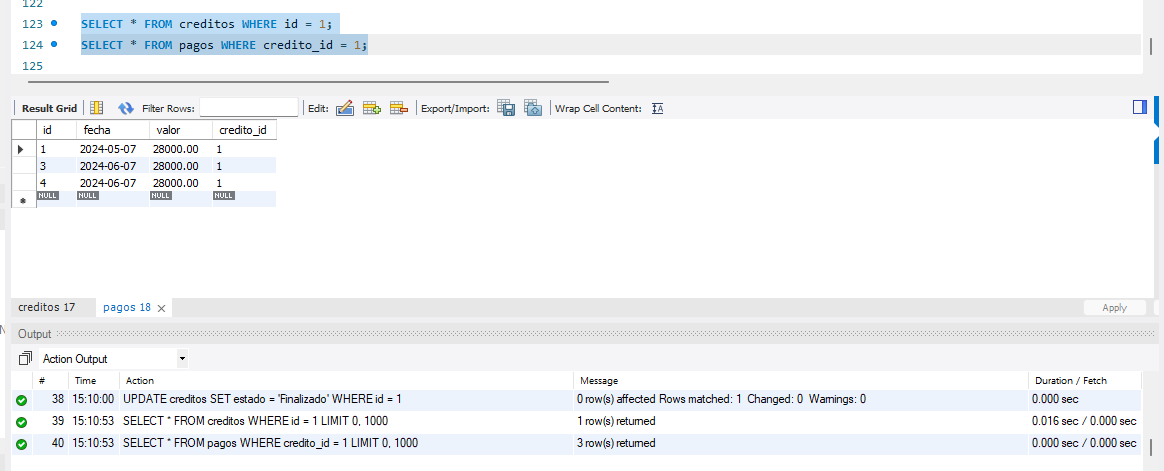
(4, DATE\_ADD(DATE\_SUB(CURDATE(), INTERVAL 11 MONTH), INTERVAL 1 MONTH), 28000, 1);

18. Actualizar el crédito a Finalizado



UPDATE creditos SET estado = 'Finalizado' WHERE id = 1;

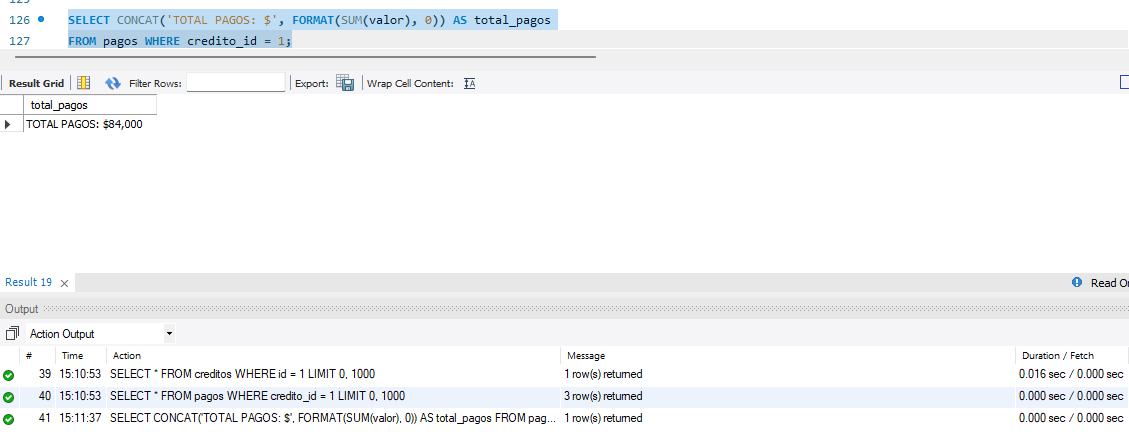
19. Mostrar crédito y pagos nuevamente



SELECT \* FROM creditos WHERE id = 1;

SELECT \* FROM pagos WHERE credito\_id = 1;

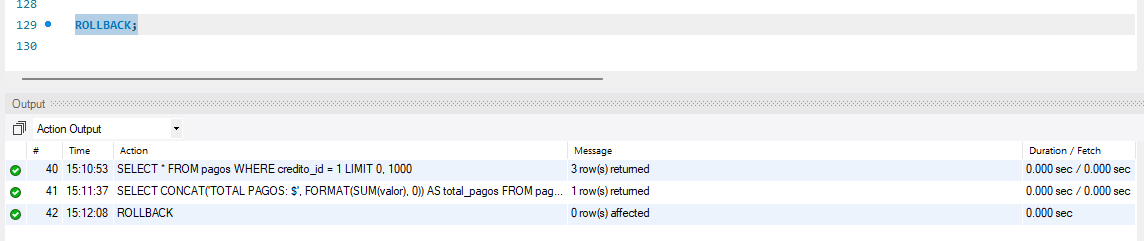
20. Totalizar pagos



SELECT CONCAT('TOTAL PAGOS: $', FORMAT(SUM(valor), 0)) AS total\_pagos

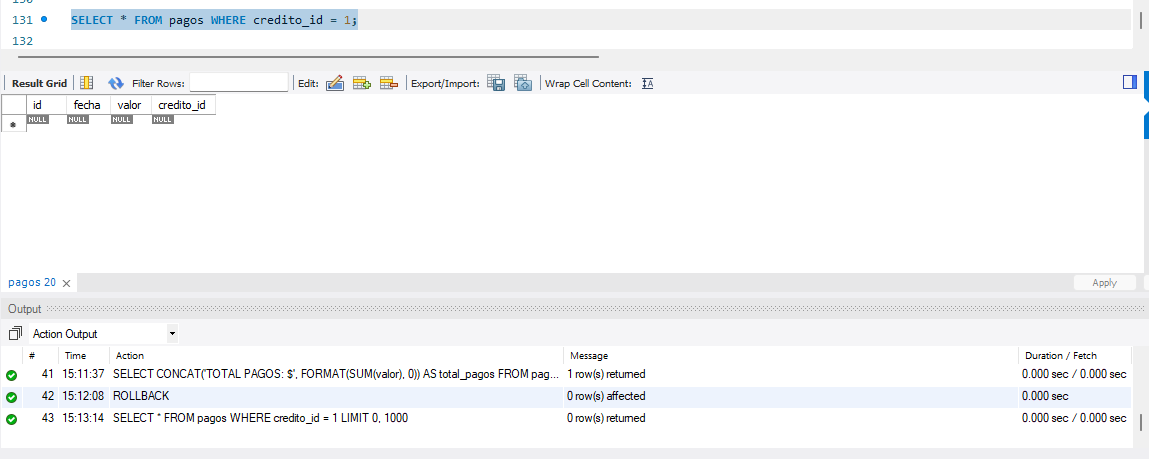
FROM pagos WHERE credito\_id = 1;

21. Cancelar transacción



ROLLBACK;

22. Mostrar pagos después de rollback



SELECT \* FROM pagos WHERE credito\_id = 1;

### **23. Comentario y análisis**

**Análisis**:

* Las **restricciones** (CHECK, NOT NULL, UNIQUE, etc.) protegen la integridad de los datos.
* La **transacción** permite hacer múltiples cambios atómicamente: todos o ninguno.
* El uso de **ROLLBACK** revierte todo si hay un error, lo cual es clave en sistemas financieros.