**Bases de datos II**

**Actividad 2**

Jonny Luna Guerrero

Edwin Marrugo Rodríguez

Alberto Valencia Solis

Universidad de Cartagena

Facultad de Ingeniería

Ingeniería de Software

Bases de Datos II

Profesor Mg. John Arrieta Arrieta

Semestre V

Cartagena, abril 24 de 2023

# Tabla de Contenido

[1 Tabla de Contenido 2](#_Toc138104847)

[2 Introducción 5](#_Toc138104848)

[3 Objetivos 6](#_Toc138104849)

[3.1 Objetivo General: 6](#_Toc138104850)

[3.2 Objetivos específicos 6](#_Toc138104851)

[4 Justificación 7](#_Toc138104852)

[5 Desarrollo 8](#_Toc138104853)

[5.1 Qué es y qué problema resuelve el uso del concepto de Vistas SQL en bases de datos Relacionales o SQL 8](#_Toc138104854)

[5.2 Analizar el Modelo Relacional del ejercicio que seleccionaron como Grupo de BD-1, diseñar una consulta multitabla y probarla (en MySQL, OracleSql, SQLServer y PostgreSQL). 9](#_Toc138104855)

[5.3 Usar la consulta anterior y crear con ella una Vista SQL 9](#_Toc138104856)

[5.4 Mostrar la Vista SQL creada anteriormente 10](#_Toc138104857)

[5.5 Ejecutar la Vista SQL creada anteriormente 10](#_Toc138104858)

[5.6 Abrir la Vista SQL y modificarla para cambiar la consulta por otra consulta o modificarla existente. 10](#_Toc138104859)

[5.7 Volver a ejecutar la Vista SQL 11](#_Toc138104860)

[5.8 Las vistas tienen ventajas, desventajas y limitaciones, muchas de estas características dependen mucho del motor de bases de datos que estemos utilizando, en este punto usted debe: 11](#_Toc138104861)

[5.8.1 Describir las ventajas del uso de Vistas SQL para el X motor de bases de datos sobre el cual desarrolló la actividad 12](#_Toc138104862)

[5.8.2 Describir las desventajas del uso de Vistas SQL para el X motor de bases de datos sobre el cual desarrolló la actividad. 12](#_Toc138104863)

[5.8.3 Describir cuales son las limitaciones que presentan las Vistas SQL en el motor de BD que está utilizando, por ejemplo: Cuáles son las instrucciones SQL que no pueden ser utilizadas dentro de una Vista SQL. etc. 13](#_Toc138104864)

[5.9 Para el desarrollo de los siguientes puntos referentes al tema de Transacciones, ustedes como grupo deben realizar las siguientes tareas sobre OracleSql, SQLServer y PostgreSQL y MySQL: 13](#_Toc138104865)

[5.9.1 Crear una base de datos llamada prueba\_transacciones\_grupoXYZ, siendo XYZ el nombre del ejercicio escogido como equipo. 13](#_Toc138104866)

[5.9.2 Crear tablas **Deudores**(**cc**, clave, nombre, apellido, email). email es único, todos son obligatorios 14](#_Toc138104867)

[5.9.3 Crear tablas **Creditos**(id, fecha, valor, cuotas, interes, estado, deudor\_id), todos son obligatorios, estado por defecto es Activo 15](#_Toc138104868)

[5.9.4 Crear tablas **Pagos**(id, fecha, valor, credito\_id). Todos son obligatorios. Use el comando check para evitar que la columna valor solo puede aceptar valor positivos mayores que 0 y la fecha del pago no puede ser mayor que la fecha actual. 16](#_Toc138104869)

[5.9.5 Insertar al menos un Deudor: 18](#_Toc138104870)

[5.9.6 Mostrar todos los Deudores insertados en la BD 18](#_Toc138104871)

[5.9.7 Insertar al menos un Crédito: 19](#_Toc138104872)

[5.9.8 Mostrar todos los Créditos insertados en la BD 19](#_Toc138104873)

[5.9.9 Insertar 5 pagos así (no modifique ningún valor del ejemplo, colocar las fechas correspondientes, el formato de fecha correspondiente para bd): 20](#_Toc138104874)

[5.9.10 Cambiar o actualizar el estado del Crédito con id=1, para que ahora sea Finalizado. 23](#_Toc138104875)

[5.9.11 Mostrar los datos del Crédito con id=1 28](#_Toc138104876)

[5.9.12 Mostrar todos los Pagos insertados al Crédito con ID=1 insertados en la BD 28](#_Toc138104877)

[5.9.13 Totalizar el total de valores pagados para el crédito con id=1, para que aparezca así: TOTAL PAGOS: $140000 29](#_Toc138104878)

[5.9.14 Analice el resultado de cada una de las operaciones anteriores, comente y explique el por que de cada uno. 30](#_Toc138104879)

[5.9.15 Elimine todos los pagos de crédito con ID: 1 30](#_Toc138104880)

[5.9.16 Mostrar todos los pagos que existen para el crédito con id=1 30](#_Toc138104881)

[5.9.17 inicie una TRANSACCION 31](#_Toc138104882)

[5.9.18 dentro de la transacción volver a insertar los mismo pagos anteriores: 33](#_Toc138104883)

[5.9.19 Cambiar o actualizar el estado del Crédito con id=1, para que ahora sea Finalizado 34](#_Toc138104884)

[5.9.20 Mostrar los datos del Crédito con id=1 36](#_Toc138104885)

[5.9.21 Mostrar todos los Pagos insertados al Crédito con ID=1 insertados en la BD 36](#_Toc138104886)

[5.9.22 Totalizar el total de valores pagados para el crédito con id=1, para que aparezca así: TOTAL PAGOS: $140000 36](#_Toc138104887)

[5.9.23 Cancelar la transaccion (rollblack) 37](#_Toc138104888)

[5.9.24 Mostrar todos los Pagos insertados al Crédito con ID=1 insertados en la BD 37](#_Toc138104889)

[5.9.25 Totalizar el total de valores pagados para el crédito con id=1, para que aparezca así: TOTAL PAGOS: $140000 38](#_Toc138104890)

[5.9.26 Analice el resultado de cada una de las operaciones anteriores, comente y explique el ¿por qué? de cada uno. 38](#_Toc138104891)

[5.10 Diligenciar la siguiente tabla con el mismo ejemplo por línea para los motores de bd: 39](#_Toc138104892)

[5.10.1 Ejemplo de subconsulta en el Select. 39](#_Toc138104893)

[5.10.2 Subconsulta en el From 40](#_Toc138104894)

[5.10.3 Subconsulta en el Where 41](#_Toc138104895)

[5.10.4 Subconsulta en el Having 42](#_Toc138104896)

[5.10.5 Uso de Check 43](#_Toc138104897)

[5.10.6 Join Inner 44](#_Toc138104898)

[5.10.7 Join Full 44](#_Toc138104899)

[5.10.8 Join Outer 46](#_Toc138104900)

[5.10.9 Join Lef 46](#_Toc138104901)

[5.10.10 Join Rigth 47](#_Toc138104902)

[5.10.11 Unión 47](#_Toc138104903)

[5.10.12 Intersect 48](#_Toc138104904)

[5.10.13 Crear una Vista SQL 49](#_Toc138104905)

[5.10.14 Usar una Vista SQL 49](#_Toc138104906)

[5.10.15 Editar una Vista SQL 50](#_Toc138104907)

[5.10.16 Mostrar el contenido de una Vista SQL 50](#_Toc138104908)

[5.10.17 Mostrar las Vistas SQL 51](#_Toc138104909)

[5.10.18 Eliminar una Vista SQL 51](#_Toc138104910)

[5.10.19 Crear una Transacción SQL 52](#_Toc138104911)

[5.10.20 Uso de Try para detectar cuando falla una transacción SQL 53](#_Toc138104912)

[6 Bibliografía 57](#_Toc138104913)

# Introducción

El uso de las transacciones es fundamental en las bases de datos relacionales debido a que nos ayudan a garantizar la integridad de los datos y a mejorar la eficiencia de las operaciones.

Una Transacción es una secuencia lógica de operaciones de base de datos que se ejecutan como una unidad indivisible. Estas operaciones pueden ser: consultas, inserciones, actualizaciones o eliminaciones. El objetivo principal de una transacción es garantizar que todas las operaciones se completen con éxito o no se realicen al ocurrir un error.

Las Transacciones se caracterizan por cumplir con los conceptos de Atomicidad, Consistencia, Aislamiento y Durabilidad.

Un procedimiento almacenado es un bloque de código SQL que se guarda en la base de datos y que se puede ejecutar todas las veces que se desee, es decir se puede reutilizar, pueden aceptar parámetros de entrada y devolver resultados.

Las ventajas que ofrecen los procedimientos almacenados pueden ser: Reutilización de código, Mejoramiento en el rendimiento, seguridad y además pueden incluir transacciones.

# Objetivos

## **Objetivo General**:

Aplicar los conceptos de Transacciones y procedimientos almacenados en las bases de datos relacionales.

## Objetivos específicos

Adquirir destreza en el uso de las Transacciones en MySQL.

Utilizar correctamente los procedimientos almacenado en MySQL.

# Justificación

El uso de transacciones y de procedimientos almacenados es muy útil en las bases de datos relacionales debido a la capacidad de garantizar la integridad de los datos, mantener su consistencia, mejorar el rendimiento, facilitar la reutilización de código además de brindar seguridad a las bases de datos.

En MySQL es mu útil utilizar tanto las transacciones como los procedimientos almacenados.

# Desarrollo

De acuerdo a lo estipulado en tutorías sobre lo concerniente a las actividades académicas evaluativas, a continuación se describe el desarrollo de la primera actividad, donde ustedes como equipo deben tomar como base el mismo ejercicio desarrollado en BD-1 (el caso práctico elegido para desarrollar las actividades propuestas en la asignatura BD-1), continuar con el desarrollo de la temática de la asignatura BD-2, por lo que para esta primera actividad (Unidad 1) deben resolver los siguiente puntos sobre Validación de Restricción de inconsistencia de entrada de datos en la BD (Comando Check), Vistas, Transacciones y su aplicación en los motores de bases de datos relacionales OracleSQL. PostgreSQL, SQLServer y MySQL.

## Qué es y qué problema resuelve el uso del concepto de Vistas SQL en bases de datos Relacionales o SQL

En SQL una vista no es más que una tabla que resulta de una consulta de una, dos o más tablas relacionadas entre si, las vistas se pueden considerar tablas virtuales ya que no son tablas como tal por no tener datos físicos almacenados aunque su estructura es la misma al tener filas y columnas ( (Microsoft, 2023).

Crear vistas en SQL tiene sus ventajas entre ellas tenemos: el control de acceso, la mejora del rendimiento, pruebas seguras, reusabilidad de consultas y mantenimiento de la integridad (Mediacloud, 2023).

## Analizar el Modelo Relacional del ejercicio que seleccionaron como Grupo de BD-1, diseñar una consulta multitabla y probarla (en MySQL, OracleSql, SQLServer y PostgreSQL).

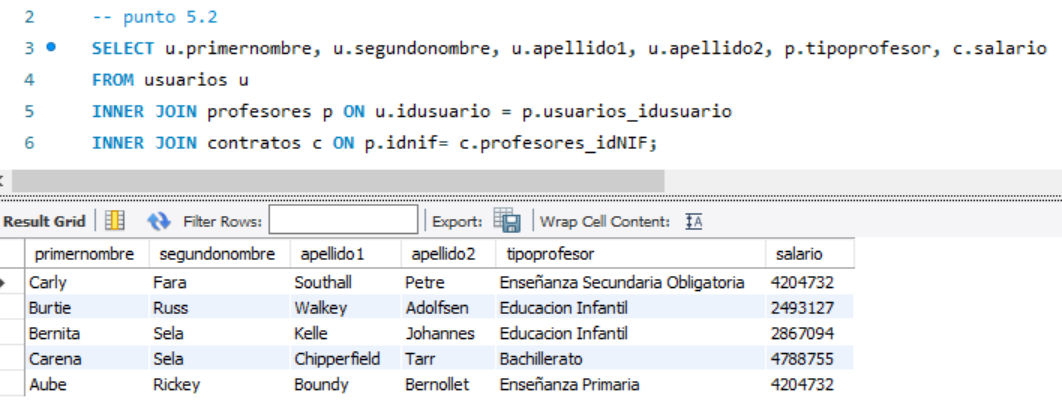
Sentencia SQL

SELECT u.primernombre, u.segundonombre, u.apellido1, u.apellido2, p.tipoprofesor, c.salario

FROM usuarios u

INNER JOIN profesores p ON u.idusuario = p.usuarios\_idusuario

INNER JOIN contratos c ON p.idnif= c.profesores\_idNIF;



## Usar la consulta anterior y crear con ella una Vista SQL

Sentencia SQL

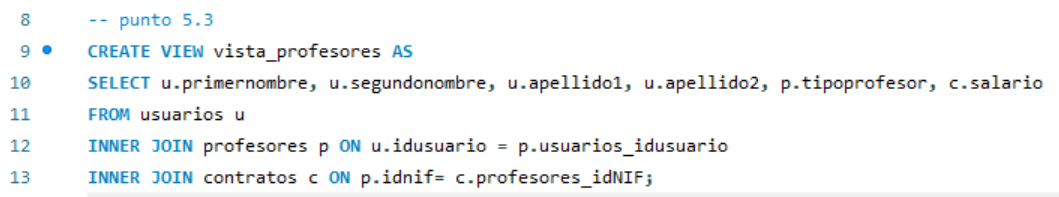
CREATE VIEW vista\_profesores AS

SELECT u.primernombre, u.segundonombre, u.apellido1, u.apellido2, p.tipoprofesor, c.salario

FROM usuarios u

INNER JOIN profesores p ON u.idusuario = p.usuarios\_idusuario

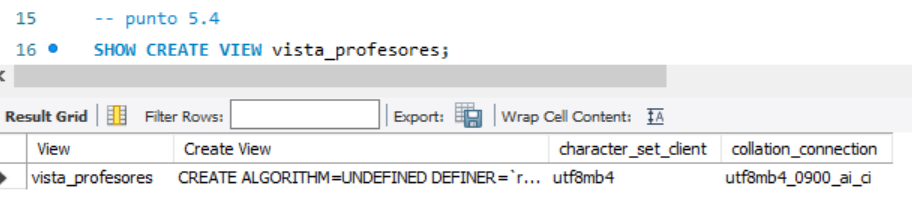
INNER JOIN contratos c ON p.idnif= c.profesores\_idNIF;



## Mostrar la Vista SQL creada anteriormente

Sentencia SQL

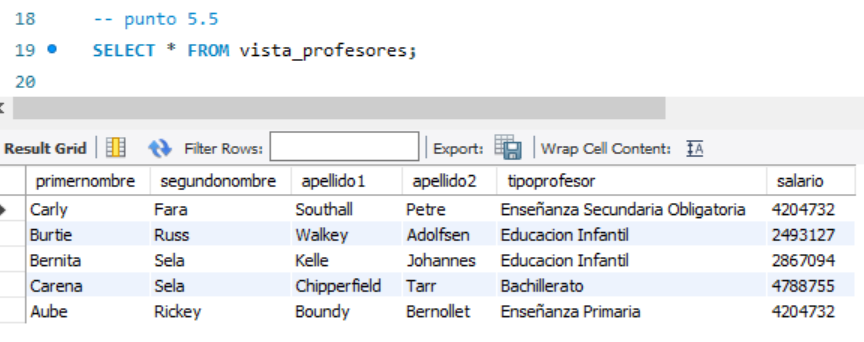
SHOW CREATE VIEW vista\_profesores;



## Ejecutar la Vista SQL creada anteriormente

Sentencia SQL

SELECT \* FROM vista\_profesores;



## Abrir la Vista SQL y modificarla para cambiar la consulta por otra consulta o modificarla existente.

Sentencia SQL

ALTER VIEW vista\_profesores AS

SELECT p. idnif, u.apellido1, u.apellido2, u.primernombre, u.segundonombre, p.tipoprofesor, c.salario

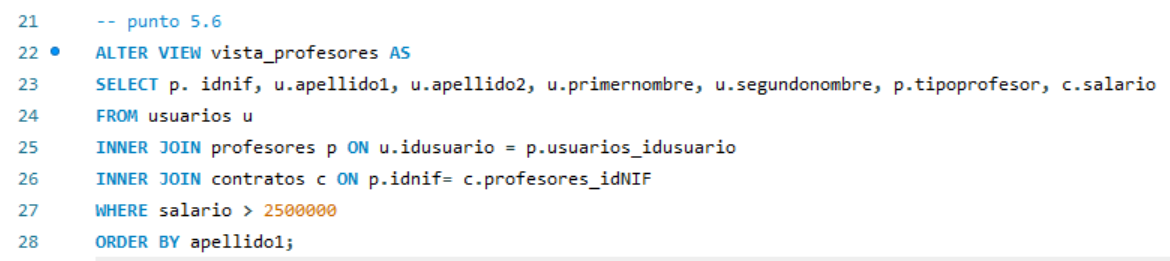
FROM usuarios u

INNER JOIN profesores p ON u.idusuario = p.usuarios\_idusuario

INNER JOIN contratos c ON p.idnif= c.profesores\_idNIF

WHERE salario > 2500000

ORDER BY apellido1;



## Volver a ejecutar la Vista SQL

Sentencia SQL

SELECT \* FROM vista\_profesores;



## Las vistas tienen ventajas, desventajas y limitaciones, muchas de estas características dependen mucho del motor de bases de datos que estemos utilizando, en este punto usted debe:

Las Vistas en bases de datos proporcionan ventajas muy interesantes debido a que ayudan a simplificar los datos, además brindan seguridad y control de acceso, simplifican las consultas frecuentes, reutilizan las consultas, simplifican la estructura de la base de datos y mejoran el rendimiento de la misma.

### Describir las ventajas del uso de Vistas SQL para el X motor de bases de datos sobre el cual desarrolló la actividad

Las ventajas en el uso de vistas en MySQL tenemos:

* Simplificación de consultas complejas: las vistas permiten simplificar las consultas complejas al encapsular la lógica de la consulta en la vista.
* Seguridad y control de acceso: pueden ser utilizadas para restringir acceso a ciertas columnas o filas de una tabla, esto mejora la seguridad y el control de acceso a los datos.
* Reutilización de consultas: Las vistas pueden ser utilizadas por múltiples consultas evitando que se escriba varias veces la misma consulta.
* Rendimiento: MySQL optimiza el rendimiento de las consultas en vista utilizando la ejecución diferida de la consulta, esto puede mejorar el rendimiento si lo comparamos con consultas complejas.

### Describir las desventajas del uso de Vistas SQL para el X motor de bases de datos sobre el cual desarrolló la actividad.

Dentro de las desventajas que se han observado en las vistas tenemos: Sobrecarga de rendimiento, esto se da cuando se utiliza un gran volumen de datos o se utilizan operaciones complejas el resultado puede ser tiempos de respuestas mas largos; Otra desventaja es la Dependencia de la estructura de la base de datos ya que si a esta se le modifica alguna de sus tablas agregando o quitando una o unas columnas la vista se podría ver afectada o incluso ser invalidada y tocaría actualizarla; Restricciones e operaciones complejas, es posible que se requiera de algunas operaciones que incluyan actualizaciones, inserciones o eliminaciones con el uso de múltiples tablas, es posible que no se permitan algunas de estas operaciones debido a que las vistas mas que todo se basan en consultas.

### Describir cuales son las limitaciones que presentan las Vistas SQL en el motor de BD que está utilizando, por ejemplo: Cuáles son las instrucciones SQL que no pueden ser utilizadas dentro de una Vista SQL. etc.

Las Limitaciones más comunes son: No se pueden indexar todas las vistas, restricciones en la modificación de datos y complejidad de mantenimiento.

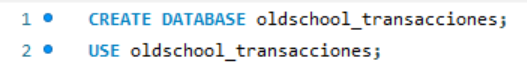
## Para el desarrollo de los siguientes puntos referentes al tema de Transacciones, ustedes como grupo deben realizar las siguientes tareas sobre OracleSql, SQLServer y PostgreSQL y MySQL:

### Crear una base de datos llamada prueba\_transacciones\_grupoXYZ, siendo XYZ el nombre del ejercicio escogido como equipo.

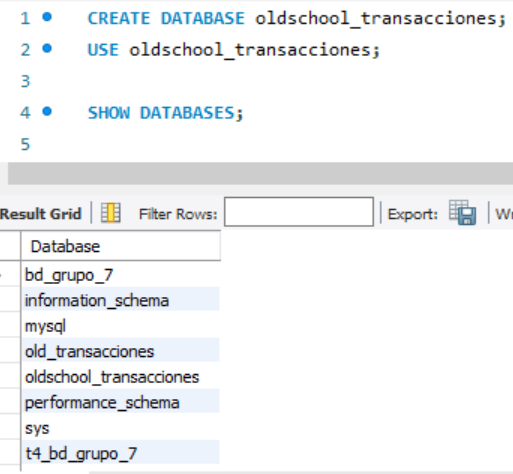
Sentencia SQL

CREATE DATABASE oldschool\_transacciones;

USE oldschool\_transacciones;



SHOW DATABASES;



### Crear tablas **Deudores**(**cc**, clave, nombre, apellido, email). email es único, todos son obligatorios

Sentencia SQL

CREATE TABLE deudores (

cc INT PRIMARY KEY NOT NULL,

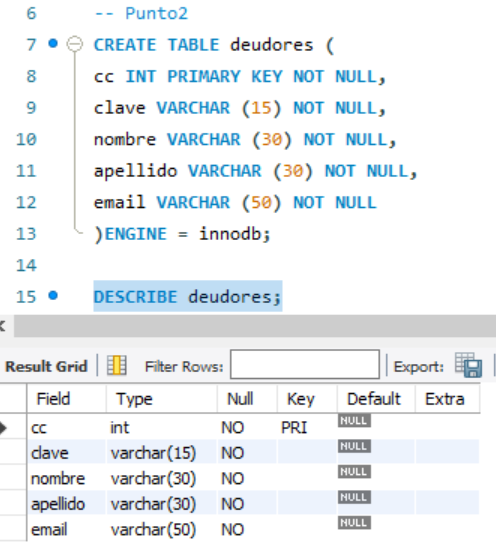
clave VARCHAR (15) NOT NULL,

nombre VARCHAR (30) NOT NULL,

apellido VARCHAR (30) NOT NULL,

email VARCHAR (50) NOT NULL

)ENGINE = innodb;



### Crear tablas **Creditos**(id, fecha, valor, cuotas, interes, estado, deudor\_id), todos son obligatorios, estado por defecto es Activo

Sentencia SQL

CREATE TABLE creditos(

id INT AUTO\_INCREMENT PRIMARY KEY NOT NULL,

fecha DATE NOT NULL,

valor FLOAT NOT NULL,

cuotas INT NOT NULL,

interes\_mes FLOAT NOT NULL,

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

deudor\_id INT NOT NULL,

CONSTRAINT chk\_interesmes

CHECK (interes\_mes>=0 AND interes\_mes<=1),

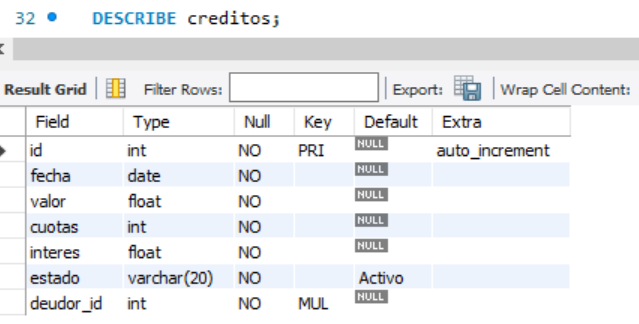
FOREIGN KEY (deudor\_id) REFERENCES deudores (cc)

ON DELETE CASCADE ON UPDATE CASCADE

)ENGINE = innodb;



DESCRIBE creditos;



### Crear tablas **Pagos**(id, fecha, valor, credito\_id). Todos son obligatorios. Use el comando check para evitar que la columna valor solo puede aceptar valor positivos mayores que 0 y la fecha del pago no puede ser mayor que la fecha actual.

Sentencia SQL

CREATE TABLE pagos(

id INT PRIMARY KEY AUTO\_INCREMENT NOT NULL,

fecha DATE NOT NULL,

valor FLOAT NOT NULL ,

credito\_id INT NOT NULL,

CONSTRAINT chk\_valor\_positivo CHECK(valor>0),

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

ON DELETE CASCADE ON UPDATE CASCADE

)ENGINE = innodb;



DELIMITER $$

CREATE TRIGGER verificar\_fecha BEFORE INSERT ON pagos

FOR EACH ROW

BEGIN

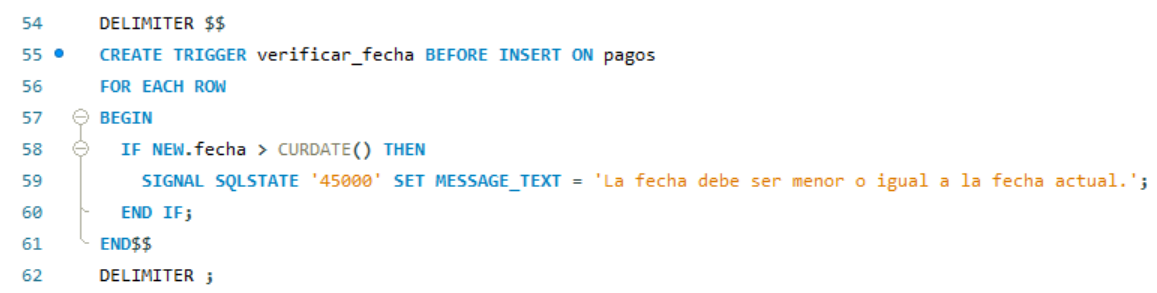
IF NEW.fecha > CURDATE() THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'La fecha debe ser menor o igual a la fecha actual.';

END IF;

END$$

DELIMITER ;



En este caso fue necesario colocar un trigger para restringir a que las fechas ingresadas no sean menores a la fecha actual.

### Insertar al menos un Deudor:

El primero con cc=123, calve=Abc, nombre = Fulanito, apellido=De tal, email=[fulanito1@gmail.com](mailto:fulanito1@gmail.com)}

Sentencia SQL

INSERT INTO deudores

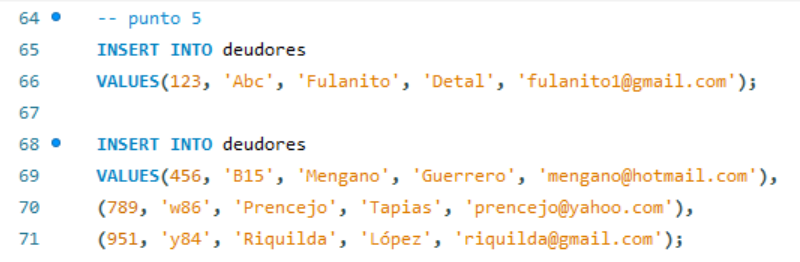
VALUES(123, 'Abc', 'Fulanito', 'Detal', 'fulanito1@gmail.com');

INSERT INTO deudores

VALUES(456, 'B15', 'Mengano', 'Guerrero', 'mengano@hotmail.com'),

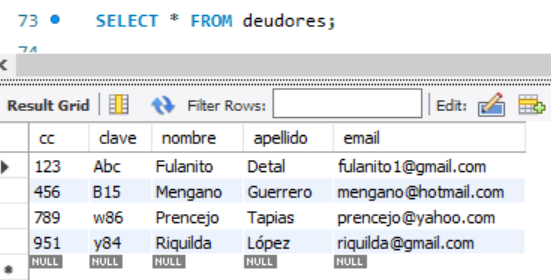
(789, 'w86', 'Prencejo', 'Tapias', 'prencejo@yahoo.com'),

(951, 'y84', 'Riquilda', 'López', 'riquilda@gmail.com');



### Mostrar todos los Deudores insertados en la BD

SELECT \* FROM deudores;



### Insertar al menos un Crédito:

id=1, fecha=hace un año, valor=100000, cuotas=5, interes\_mes=40%, deudor\_id=123

Sentencia SQL

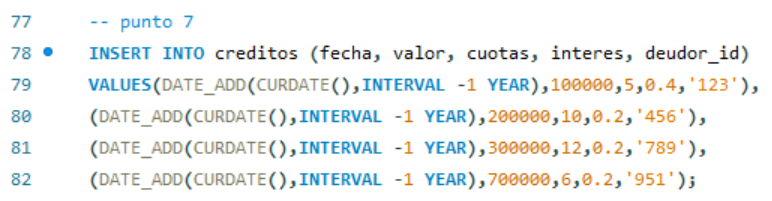
INSERT INTO creditos (fecha, valor, cuotas, interes, deudor\_id)

VALUES(DATE\_ADD(CURDATE(),INTERVAL -1 YEAR),100000,5,0.4,'123'),

(DATE\_ADD(CURDATE(),INTERVAL -1 YEAR),200000,10,0.2,'456'),

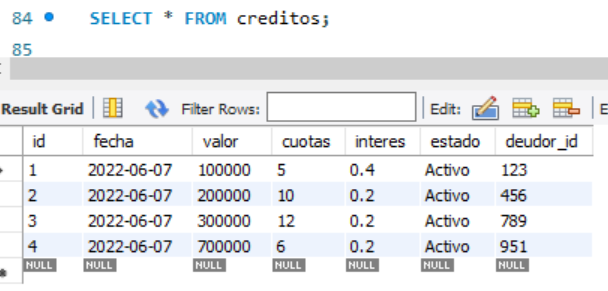
(DATE\_ADD(CURDATE(),INTERVAL -1 YEAR),300000,12,0.2,'789'),

(DATE\_ADD(CURDATE(),INTERVAL -1 YEAR),700000,6,0.2,'951');



### Mostrar todos los Créditos insertados en la BD

SELECT \* FROM creditos;



### Insertar 5 pagos así (no modifique ningún valor del ejemplo, colocar las fechas correspondientes, el formato de fecha correspondiente para bd):

Para hacer este punto realizamos un procedimiento almacenado que afecte la tabla pagos y en la tabla créditos vaya descontando del valor del crédito con la siguiente Sentencia SQL

DELIMITER //

CREATE PROCEDURE InsertarPago

(pago\_id INT,

pago\_fecha DATE,

pago\_valor FLOAT,

credito\_id INT)

BEGIN

-- Se decalra el error

DECLARE error\_obtenido BOOLEAN DEFAULT FALSE;

-- Insertar el pago

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

VALUES (pago\_id, pago\_fecha, pago\_valor, credito\_id);

IF ROW\_COUNT() = 1 THEN

SELECT CONCAT("El pago con Id ", pago\_id,

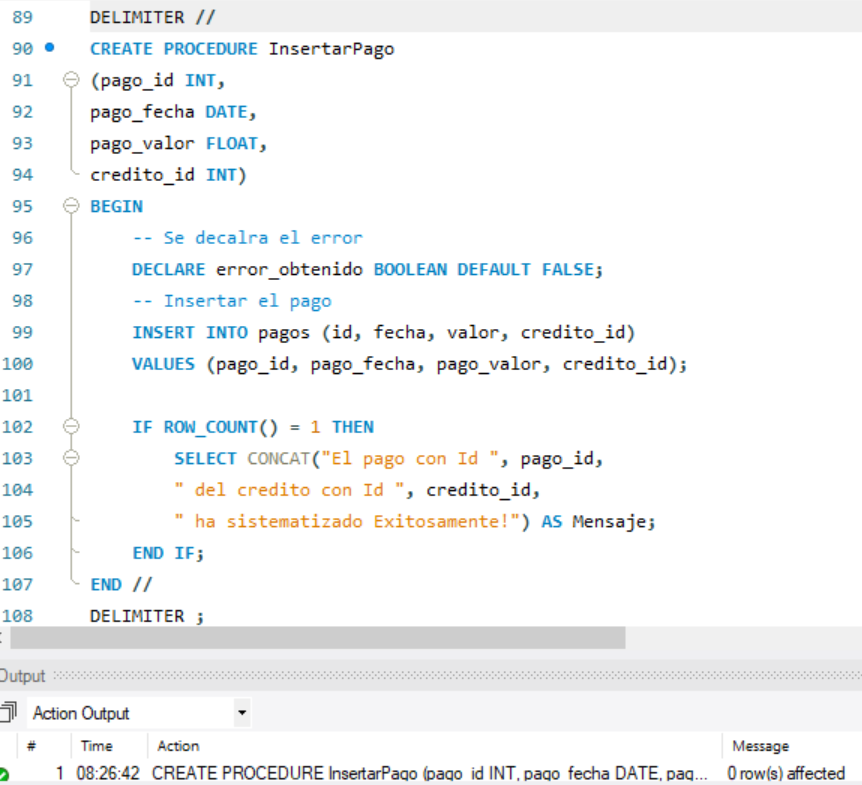
" del credito con Id ", credito\_id,

" ha sistematizado Exitosamente!") AS Mensaje;

END IF;

END //

DELIMITER ;

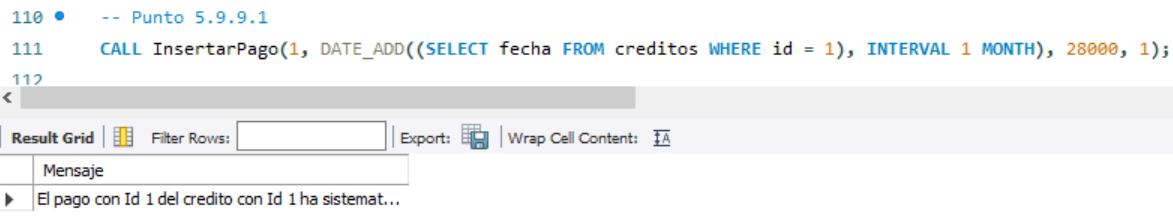


#### id=1, fecha=**un mes después del crédito**, valor=28000, credito\_id=1

Se llama el procedimiento **InsertarPago** para realizar el pago.

Sentencia SQL

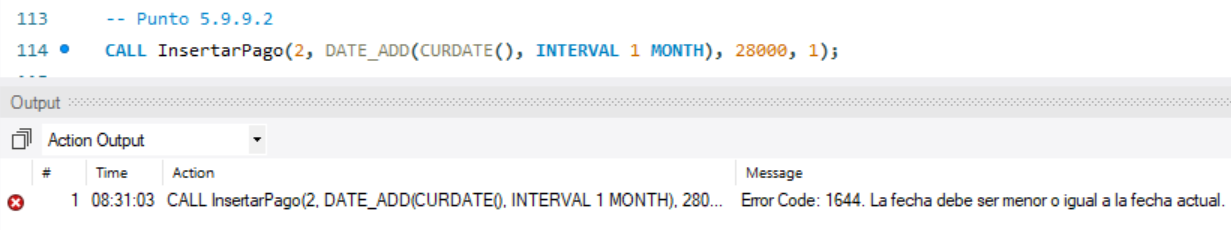
CALL InsertarPago(1, DATE\_ADD((SELECT fecha FROM creditos WHERE id = 1), INTERVAL 1 MONTH), 28000, 1);



#### id=2, fecha=**un mes después de la fecha actual,** valor=28000, credito\_id=1

Sentencia SQL

CALL InsertarPago(2, DATE\_ADD(CURDATE(), INTERVAL 1 MONTH), 28000, 1);

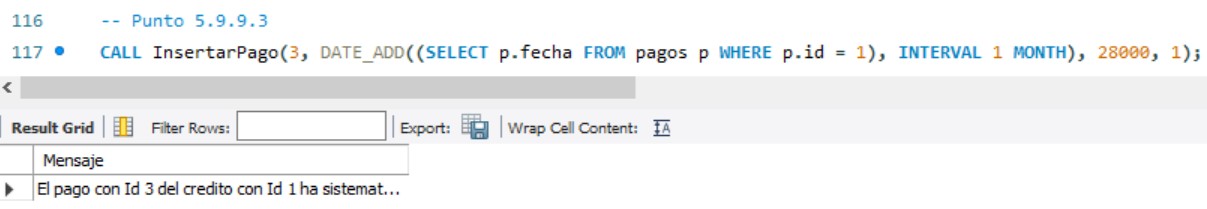


Se observa que al intentar introducir un pago con fecha mayor a la fecha actual da un error esto nos indica que las restricción colocada en un ejercicio anterior es correcta.

#### id=3, fecha=**un mes después del primer pago,** valor=28000, credito\_id=1

Sentencia SQL

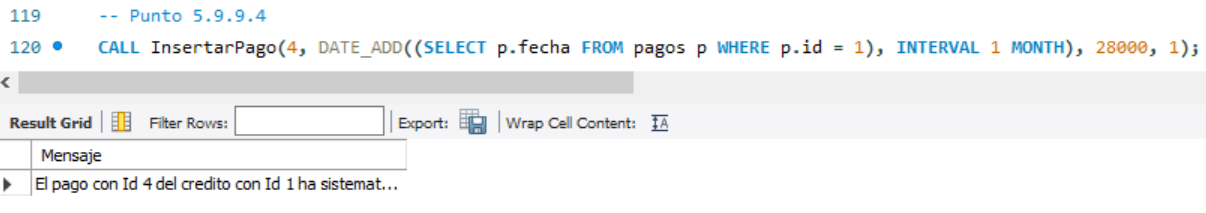
CALL InsertarPago(3, DATE\_ADD((SELECT p.fecha FROM pagos p WHERE p.id = 1), INTERVAL 1 MONTH), 28000, 1);



#### id=4, fecha=**un mes después del primer pago,** valor=28000, credito\_id=1

Sentencia SQL

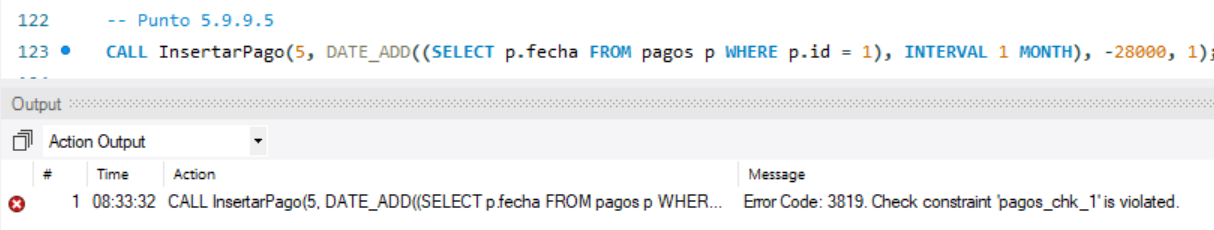
CALL InsertarPago(4, DATE\_ADD((SELECT p.fecha FROM pagos p WHERE p.id = 1), INTERVAL 1 MONTH), 28000, 1);



#### id=5, fecha=**un mes después del primer pago,** valor=**-**28000, credito\_id=1

Sentencia SQL

CALL InsertarPago(5, DATE\_ADD((SELECT p.fecha FROM pagos p WHERE p.id = 1), INTERVAL 1 MONTH), -28000, 1);



Se observa en este punto que al intentar introducir un valor negativo en el pago se viola la restricción CHECK hecha en esa columna donde no se permiten valores negativos para el pago.

### Cambiar o actualizar el estado del Crédito con id=1, para que ahora sea Finalizado.

Para que un crédito tenga el estado finalizado debe haberse cancelado en su totalidad, por lo tanto realizamos un procedimiento con el nombre EstadoCredito () que valide esta condición. Un crédito finalizado debe cumplir con la condición de que el valor del crédito más los intereses generados sea igual al valor pagado es decir a la suma de los valores pagados en las cuotas.

DELIMITER //

CREATE PROCEDURE EstadoCredito(IN credito\_id INT)

BEGIN

DECLARE valor\_credito FLOAT;

DECLARE tasa\_interes FLOAT;

DECLARE valor\_deuda FLOAT;

DECLARE valor\_saldo FLOAT;

DECLARE valor\_pagado FLOAT;

DECLARE estado\_actual VARCHAR(20);

DECLARE estado\_final VARCHAR(20);

-- Obtenemos el valor del crédito y la tasa de interés

SELECT valor, interes\_mes, estado INTO valor\_credito, tasa\_interes, estado\_actual

FROM creditos

WHERE id = credito\_id;

-- Calculamos el valor de la deuda

SET valor\_deuda = valor\_credito \*(1+ tasa\_interes);

-- Calculamos el valor total pagado

SELECT SUM(valor) INTO valor\_pagado

FROM pagos

WHERE credito\_id = credito\_id;

-- calculamos el saldo

SET valor\_saldo = valor\_deuda - valor\_pagado;

-- Determinar el estado del crédito

IF valor\_pagado >= valor\_deuda THEN

SET estado\_final = 'Finalizado';

ELSE

SET estado\_final = estado\_actual;

END IF;

-- Actualizar el estado del crédito en la tabla Creditos

UPDATE creditos

SET estado = estado\_final

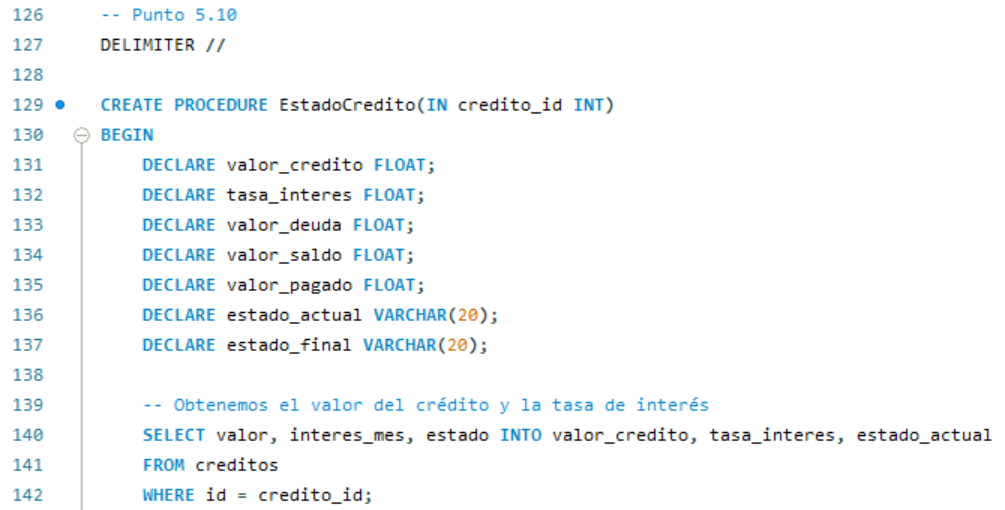
WHERE id = credito\_id;

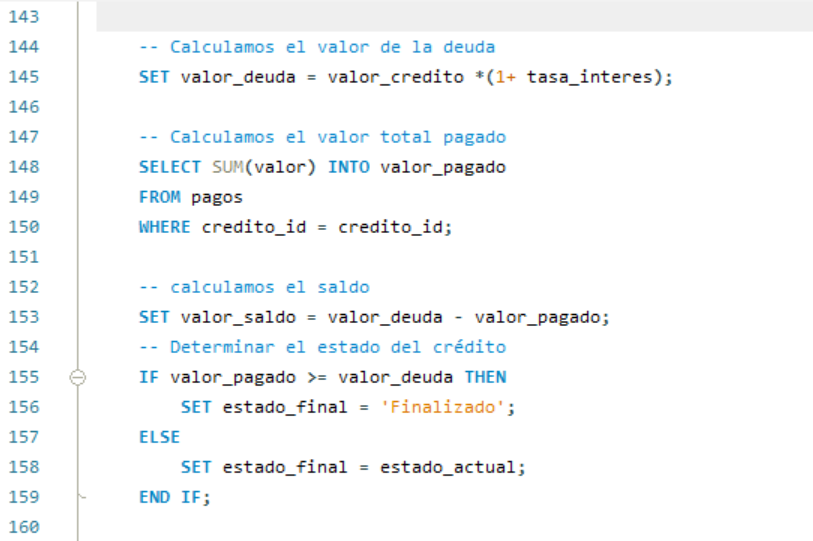
-- enviamos un mensaje con el estado final del credito

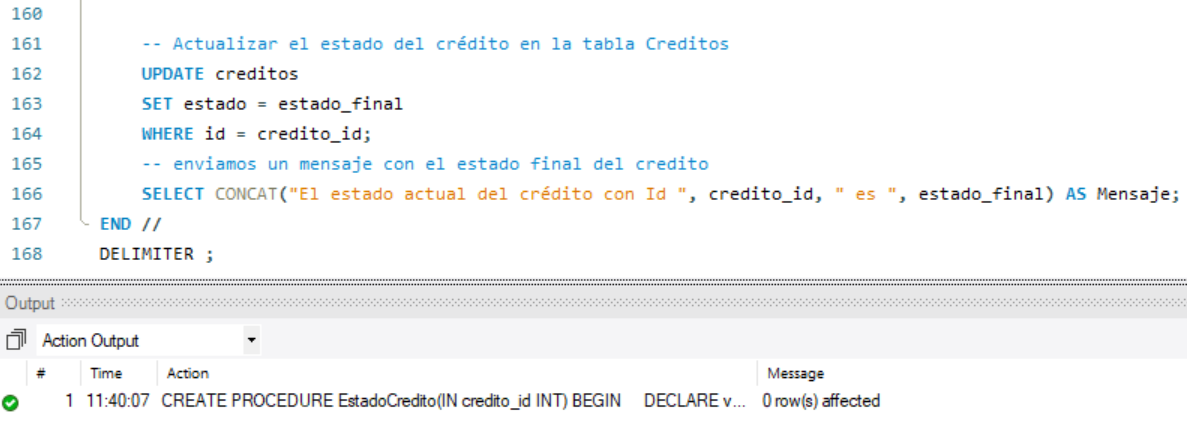
SELECT CONCAT("El estado actual del crédito con Id ", credito\_id, " es ", estado\_final) AS Mensaje;

END //

DELIMITER ;

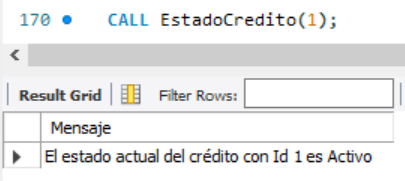






Ahora llamamos el procedimiento EstadoCredito() con id = 1

CALL EstadoCredito(1);

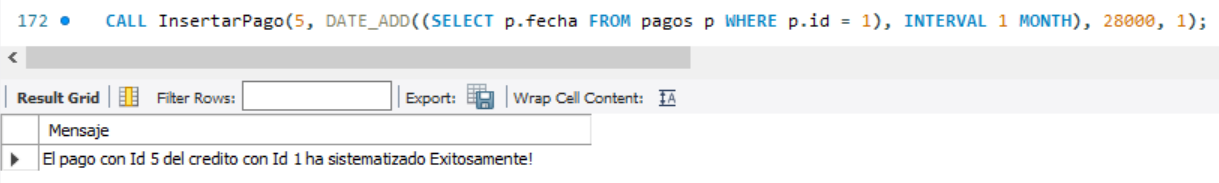


Se observa que como solo se han hecho 3 pagos efectivos en el crédito con id = 1 el estado es activo.

Ahora procedemos a realizar dos pagos adicionales para verificar la funcionalidad del procedimiento almacenado y que nos de como resultado lo solicitado es decir el estado = “Finalizado”.

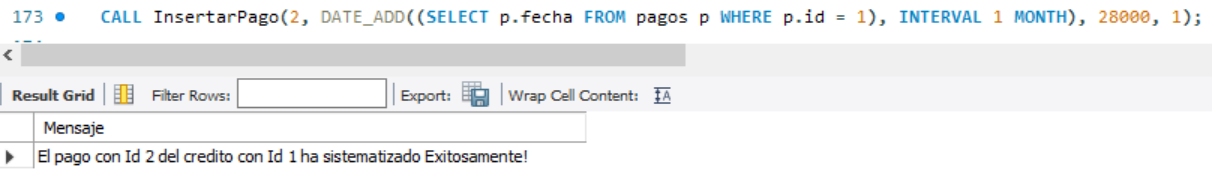
Corregimos el pago con el id 5 el cual se hacia con un pago de valor negativo.

CALL InsertarPago(5, DATE\_ADD((SELECT p.fecha FROM pagos p WHERE p.id = 1), INTERVAL 1 MONTH), 28000, 1);



También corregimos el pago con id = 2 el cual tenia como fecha de pago un mes posterior a la fecha actual.

CALL InsertarPago(2, DATE\_ADD((SELECT p.fecha FROM pagos p WHERE p.id = 1), INTERVAL 1 MONTH), 28000, 1);



Finalmente llamamos nuevamente al procedimiento CALL EstadoCredito() para ver el nuevo estado del crédito con id = 1.

CALL EstadoCredito(1);

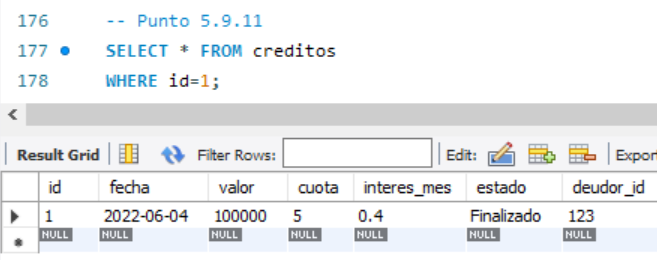


### Mostrar los datos del Crédito con id=1

Sentencia SQL

SELECT \* FROM creditos

WHERE id=1;

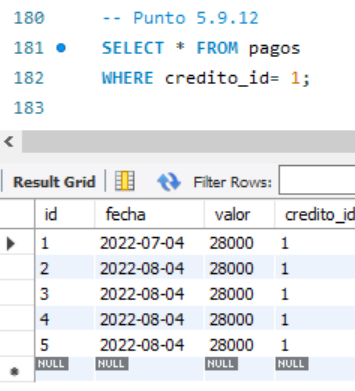


### Mostrar todos los Pagos insertados al Crédito con ID=1 insertados en la BD

Sentencia SQL

SELECT \* FROM pagos

WHERE credito\_id= 1;



### Totalizar el total de valores pagados para el crédito con id=1, para que aparezca así: TOTAL PAGOS: $140000

Sentencia SQL

SELECT credito\_id, SUM(valor) AS 'TOTAL PAGOS'

FROM pagos

WHERE credito\_id = 1;



### Analice el resultado de cada una de las operaciones anteriores, comente y explique el por que de cada uno.

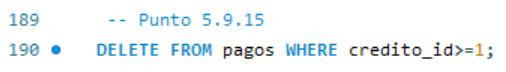
Se observa que en el crédito con id = 1 el cual corresponde a un valor de $ 100.000 con una tasa de interés del 40% mensual y cuotas a pagar 5 por un valor de $28.000 cada una. Al realizar los 5 pagos se observó que el pago con id = 2 no fue posible porque violaba la restricción hecha de que no se puede ingresar pagos con fecha posterior a la que se hace el mismo, el pago con id = 5 tampoco pudo hacerse porque viola la restricción del valor negativo.

Por tales razones al totalizar los pagos se observa que solo fueron pagados 3 cuotas y debido a eso el TOTAL PAGOS fue de $ 84.000 lo que indica que aún hay dos cuotas pendientes por pagar y el crédito no puede estar en estado = ‘Finalizado’, .

### Elimine todos los pagos de crédito con ID: 1

Sentencia SQL

DELETE FROM pagos WHERE credito\_id=1;

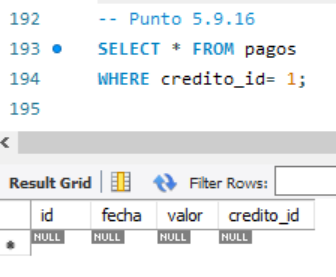


### Mostrar todos los pagos que existen para el crédito con id=1

Sentencia SQL

SELECT \* FROM pagos

WHERE credito\_id= 1;



### inicie una TRANSACCION

Para realizar este punto realizamos un procedimiento almacenado nuevamente esta vez incluimos dentro del mismo la transacción para que capturara el error si se da.

Sentencia SQL

DELIMITER //

CREATE PROCEDURE insertar\_pago\_transaccion

(

p\_id INT,

p\_fecha DATE,

p\_valor FLOAT,

p\_credito\_id INT

)

BEGIN

DECLARE EXIT HANDLER FOR SQLEXCEPTION

BEGIN

SHOW ERRORS LIMIT 1;

RESIGNAL;

ROLLBACK;

END;

START TRANSACTION;

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

VALUES(p\_id, p\_fecha, p\_valor, p\_credito\_id);

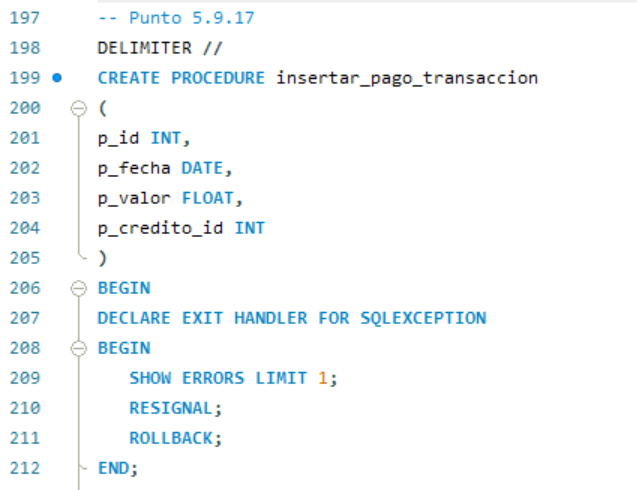
COMMIT;

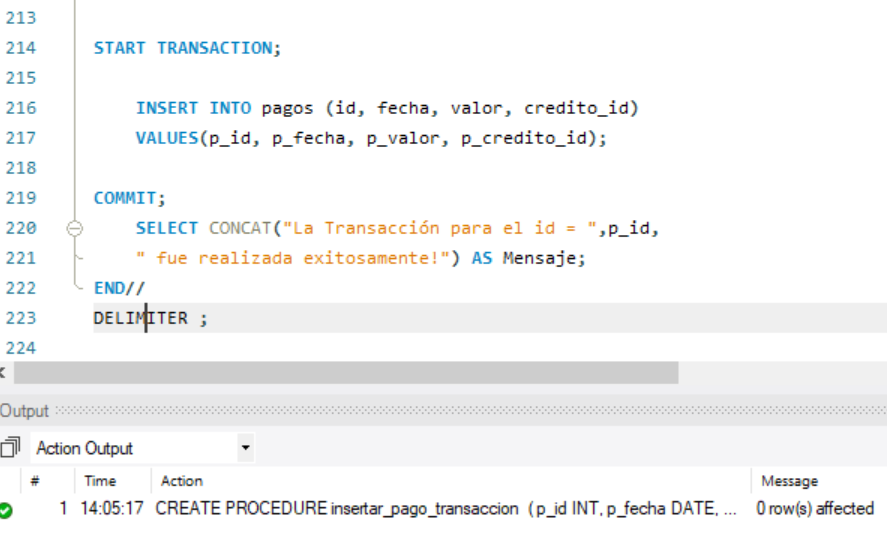
SELECT CONCAT("La Transacción para el id = ",p\_id,

" fue realizada exitosamente!") AS Mensaje;

END//

DELIMITER ;



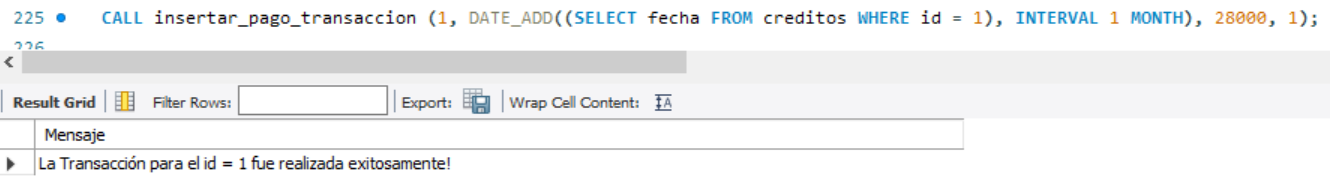


### dentro de la transacción volver a insertar los mismo pagos anteriores:

Insertamos los pagos

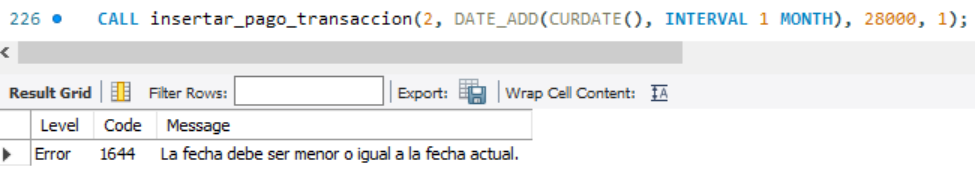
#### id=1, fecha=**un mes después del crédito**, valor=28000, credito\_id=1

CALL insertar\_pago\_transaccion (1, DATE\_ADD((SELECT fecha FROM creditos WHERE id = 1), INTERVAL 1 MONTH), 28000, 1);



#### id=2, fecha=**un mes después de la fecha actual,** valor=28000, credito\_id=1

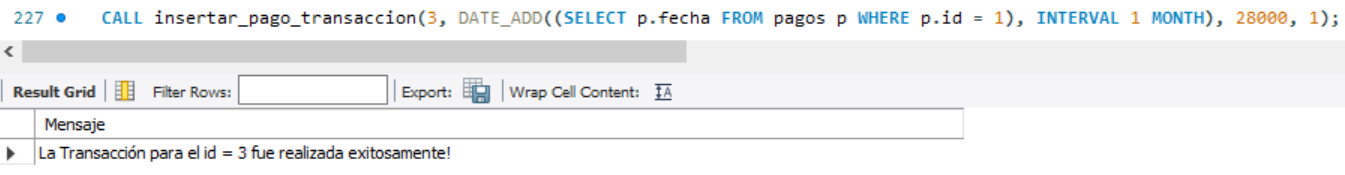
CALL insertar\_pago\_transaccion(2, DATE\_ADD(CURDATE(), INTERVAL 1 MONTH), 28000, 1);



En este caso se captura el error y no se ejecuta la transacción.

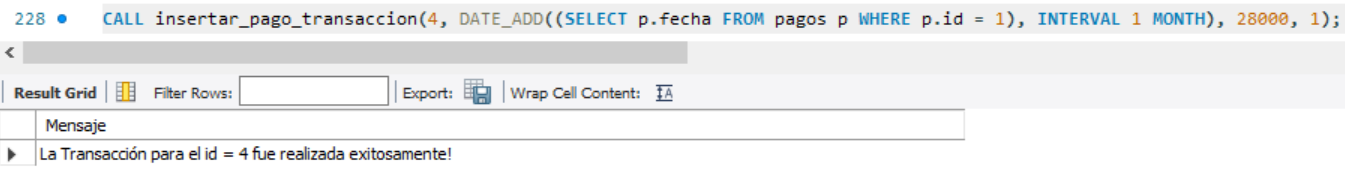
#### id=3, fecha=**un mes después del primer pago,** valor=28000, credito\_id=1

CALL insertar\_pago\_transaccion(3, DATE\_ADD((SELECT p.fecha FROM pagos p WHERE p.id = 1), INTERVAL 1 MONTH), 28000, 1);



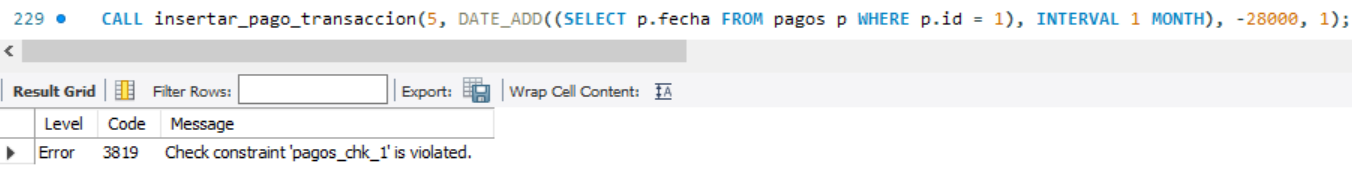
#### id=4, fecha=**un mes después del primer pago,** valor=28000, credito\_id=1

CALL insertar\_pago\_transaccion(4, DATE\_ADD((SELECT p.fecha FROM pagos p WHERE p.id = 1), INTERVAL 1 MONTH), 28000, 1);



#### id=5, fecha=**un mes después del primer pago,** valor=**-**28000, credito\_id=1

CALL insertar\_pago\_transaccion(5, DATE\_ADD((SELECT p.fecha FROM pagos p WHERE p.id = 1), INTERVAL 1 MONTH), -28000, 1);

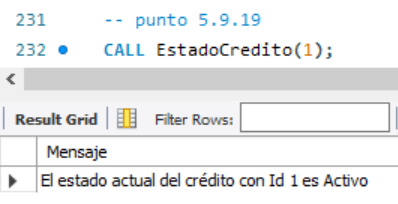


### Cambiar o actualizar el estado del Crédito con id=1, para que ahora sea Finalizado

En este punto utilizamos el procedimiento realizado anteriormente **EstadoCredito**

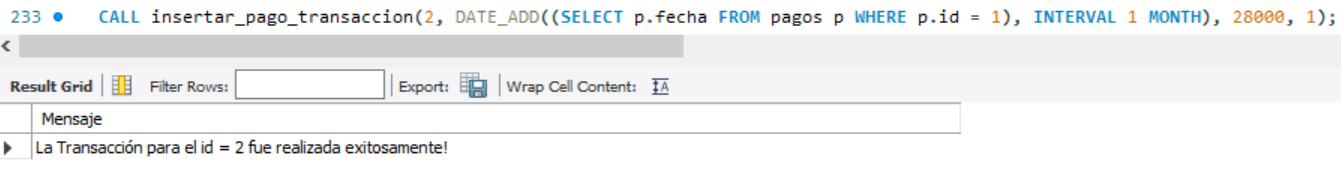
Sentencia SQL

CALL EstadoCredito(1);

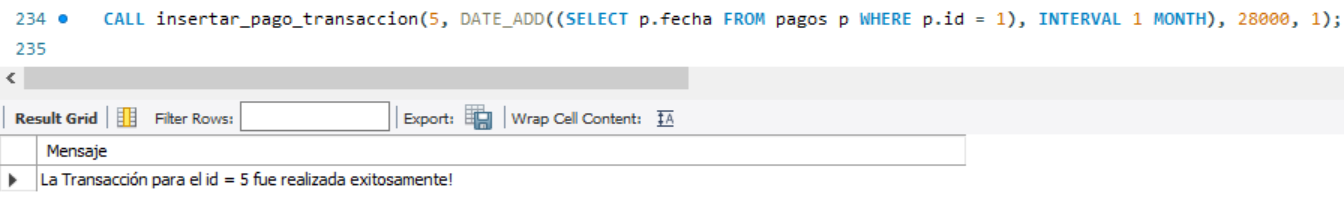


Se observa que el crédito está activo debido a que los pagos con id = 2 e id = 5 fueron rechazados por violar las restricciones de fecha y valor negativo para el pago respectivamente. Procedemos a corregirlos para que el crédito se finalice.

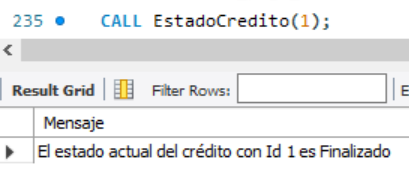
CALL insertar\_pago\_transaccion(2, DATE\_ADD((SELECT p.fecha FROM pagos p WHERE p.id = 1), INTERVAL 1 MONTH), 28000, 1);



CALL insertar\_pago\_transaccion(5, DATE\_ADD((SELECT p.fecha FROM pagos p WHERE p.id = 1), INTERVAL 1 MONTH), 28000, 1);



Nuevamente miramos el estado del crédito

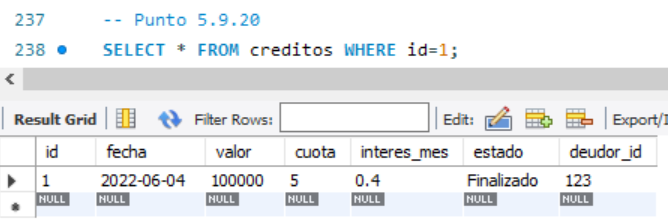


Observamos que ahora si está finalizado el crédito

### Mostrar los datos del Crédito con id=1

Sentencia SQL

SELECT \* FROM creditos WHERE id= 1;

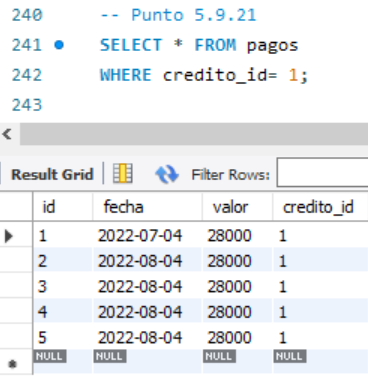


### Mostrar todos los Pagos insertados al Crédito con ID=1 insertados en la BD

Sentencia SQL

SELECT \* FROM pagos

WHERE credito\_id= 1;



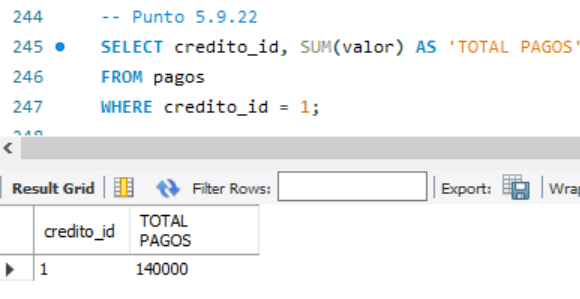
### Totalizar el total de valores pagados para el crédito con id=1, para que aparezca así: TOTAL PAGOS: $140000

Sentencia SQL

SELECT credito\_id, SUM(valor) AS 'TOTAL PAGOS'

FROM pagos

WHERE credito\_id = 1;



### Cancelar la transaccion (rollblack)

ROLLBACK;

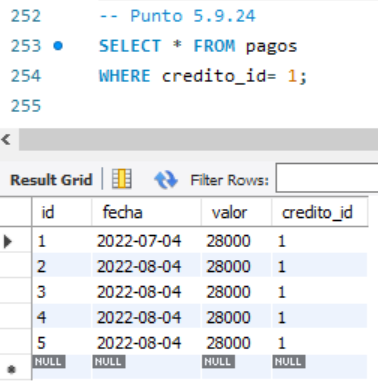


### Mostrar todos los Pagos insertados al Crédito con ID=1 insertados en la BD

Sentencia SQL

SELECT \* FROM pagos

WHERE credito\_id= 1;



Una vez hecho el COMMIT la instrucción ROLLBACK no permite que se elimine la transacción por eso se observa que los pagos realizados quedaron en la tabla pagos.

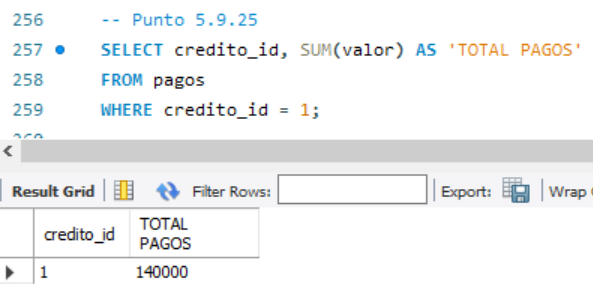
### Totalizar el total de valores pagados para el crédito con id=1, para que aparezca así: TOTAL PAGOS: $140000

Sentencia SQL

SELECT credito\_id, SUM(valor) AS 'TOTAL PAGOS'

FROM pagos

WHERE credito\_id = 1;



### Analice el resultado de cada una de las operaciones anteriores, comente y explique el ¿por qué? de cada uno.

En este punto se observa que al hacer los pagos en la transacciones los pagos con id = 2 e id = 5 dieron errores en la transacción y no fue posible su registro, debido a que el segundo pago genera un error por violar la restricción de fecha y el quinto pago viola la restricción del valor por ser menor que cero. Para establecer el estado del crédito en finalizado es necesario corregir estos errores y hacer los pagos respectivos. La sentencia ROLLBACK No revierte las operaciones que se encuentran dentro de la transacción debido a que estas ya fueron ejecutadas con un COMMIT, por ese motivo al ejecutar esta instrucción se observa que la tabla pagos no tiene cambios al final.

## Diligenciar la siguiente tabla con el mismo ejemplo por línea para los motores de bd:

MySQL

A continuación, resolvemos los puntos de la tabla.

### Ejemplo de subconsulta en el Select.

Se desea saber el nombre de los deudores con el total de créditos activos tenemos la tabla deudores y la tabla creditos.

Tabla deudores

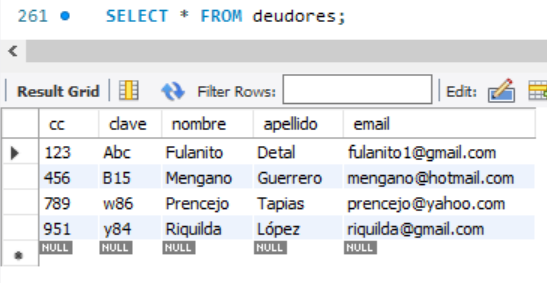
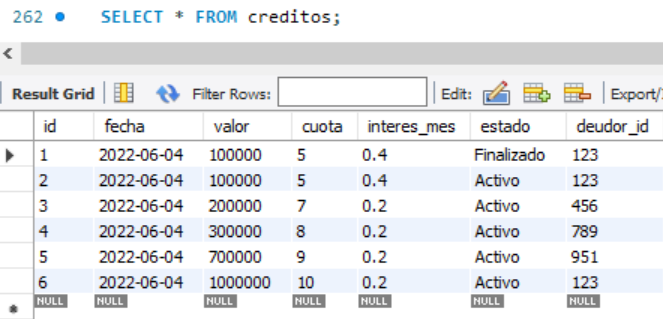


Tabla créditos



Sentencia SQL de la subconsulta en el FROM

SELECT d.nombre, (

SELECT COUNT(\*)

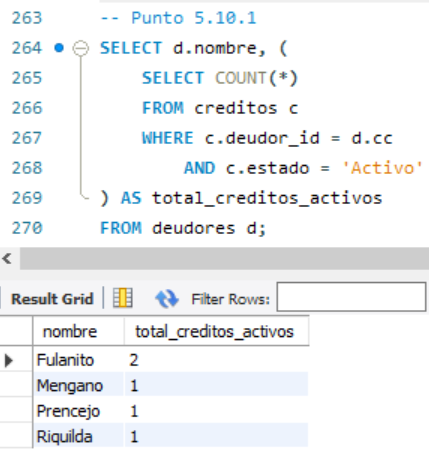
FROM creditos c

WHERE c.deudor\_id = d.cc

AND c.estado = 'Activo'

) AS total\_creditos\_activos

FROM deudores d;



### Subconsulta en el From

Se desea obtener el valor total de los créditos de cada deudor.

Sentencia SQL.

SELECT d.nombre, c.valor\_credito

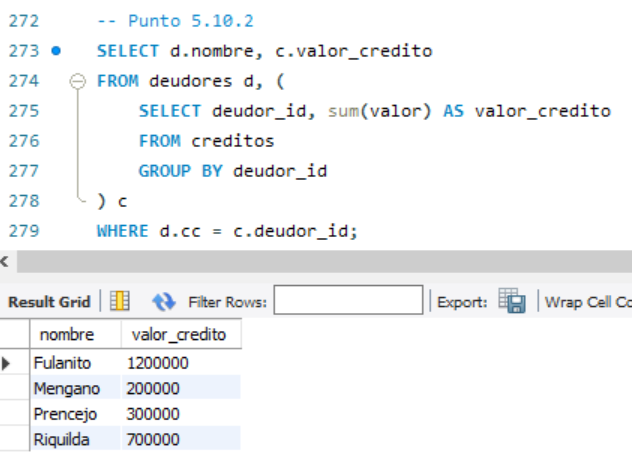
FROM deudores d, (

SELECT deudor\_id, sum(valor) AS valor\_credito

FROM creditos

GROUP BY deudor\_id) c

WHERE d.cc = c.deudor\_id;



### Subconsulta en el Where

Se desea conocer los deudores que tienen crédito activo.

Sentencia SQL

SELECT d.nombre

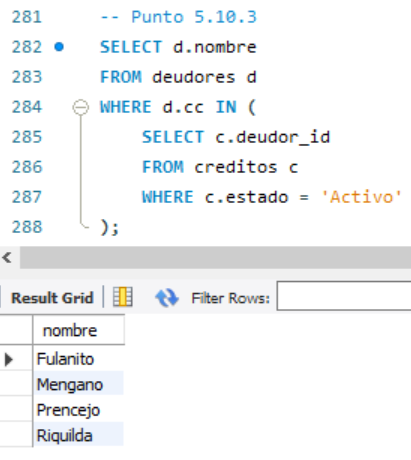
FROM deudores d

WHERE d.cc IN (

SELECT c.deudor\_id

FROM creditos c

WHERE c.estado = 'Activo');



### Subconsulta en el Having

Se desea obtener una lista de deudores junto con el número de créditos activos pero solo mostrar los deudores que tengan mas créditos activos que el promedio de créditos activos de todos los deudores.

Sentencia SQL

SELECT d.nombre, COUNT(c.id) AS total\_creditos\_activos

FROM deudores d

JOIN creditos c ON d.cc = c.deudor\_id

WHERE c.estado = 'Activo'

GROUP BY d.cc

HAVING COUNT(c.id) > (

SELECT AVG(total\_creditos)

FROM (

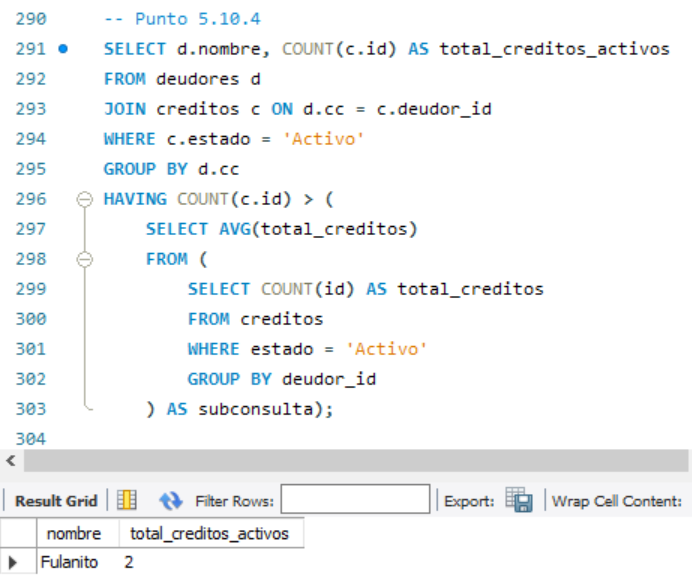
SELECT COUNT(id) AS total\_creditos

FROM creditos

WHERE estado = 'Activo'

GROUP BY deudor\_id

) AS subconsulta);



### Uso de Check

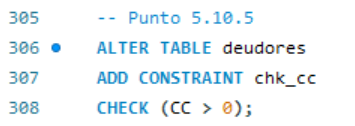
Se desea restringir el ingreso de numero de cedula menor o igual que cero en la tabla deudores en la columna cc.

Sentencia SQL

ALTER TABLE deudores

ADD CONSTRAINT chk\_cc

CHECK (CC > 0);





Se observa que al intentar introducir un deudor con valor de cedula negativa viola la restricción hecha y no permite la inserción del deudor.

### Join Inner

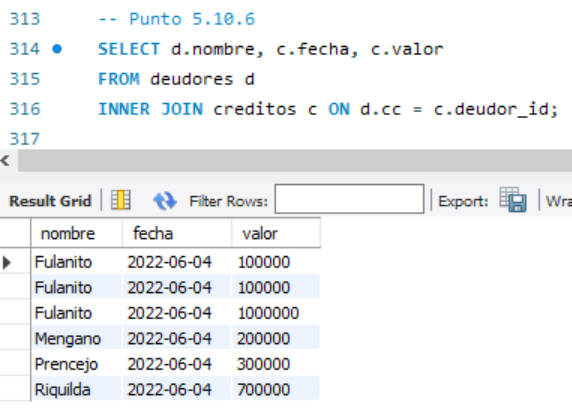
Se desea obtener una consulta que muestre el nombre del deudor en la tabla deudor y unirlo con la fecha y el valor del crédito de la tabla creditos.

Sentencia SQL

SELECT d.nombre, c.fecha, c.valor

FROM deudores d

INNER JOIN creditos c ON d.cc = c.deudor\_id;



### Join Full

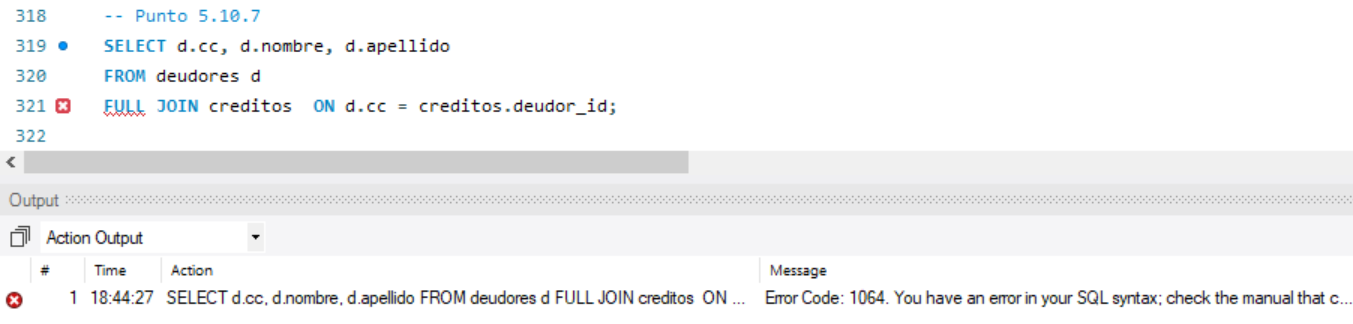
Se desea obtener una consulta que muestre el nombre del deudor en la tabla deudor y unirlo con la fecha y el valor del crédito de la tabla creditos.

Sentencia SQL

SELECT d.cc, d.nombre, d.apellido

FROM deudores d

FULL JOIN creditos ON d.cc = creditos.deudor\_id;



MySQL no admite FULL JOIN, por lo tanto esta consulta se hace con una opción alternativa que es combinar el RIGHT JOIN CON EL LEFT JOIN

Sentencia SQL

SELECT d.nombre, c.fecha, c.valor

FROM deudores d

LEFT JOIN creditos c ON d.cc = c.deudor\_id

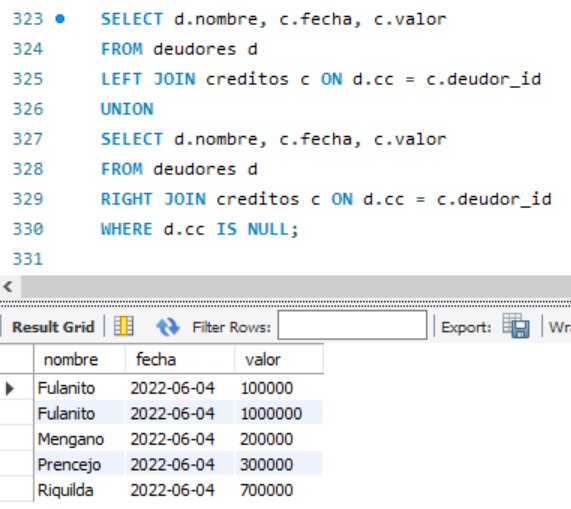
UNION

SELECT d.nombre, c.fecha, c.valor

FROM deudores d

RIGHT JOIN creditos c ON d.cc = c.deudor\_id

WHERE d.cc IS NULL;



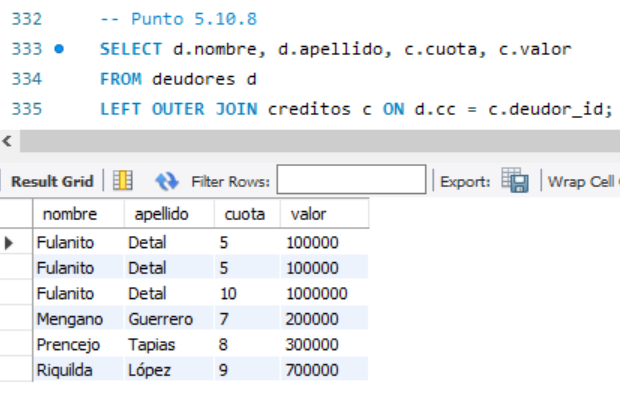
### Join Outer

Sentencia SQL

SELECT d.nombre, d.apellido, c.cuota, c.valor

FROM deudores d

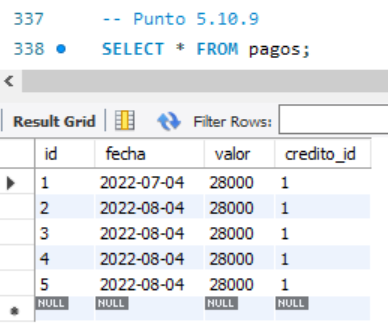
LEFT OUTER JOIN creditos c ON d.cc = c.deudor\_id;



### Join Lef

Para hacer este ejemplo y los siguientes hacemos los pagos de los puntos 5.9.9.1, al 5.9.9.5 corregidos estos se ejecutaron en las líneas correspondientes del script.

Tabla pagos



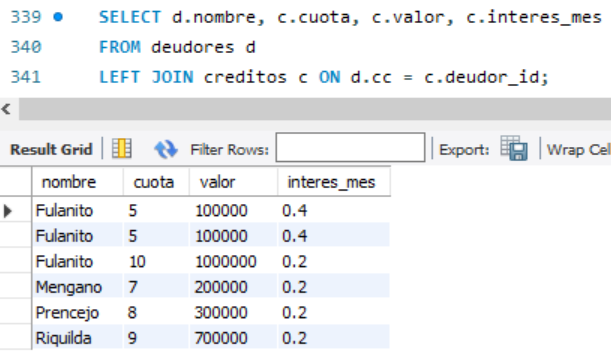
Luego hacemos la sentencia para el LEFT JOIN

Sentencia SQL

SELECT d.nombre, c.cuota, c.valor, c.interes\_mes

FROM deudores d

LEFT JOIN creditos c ON d.cc = c.deudor\_id;



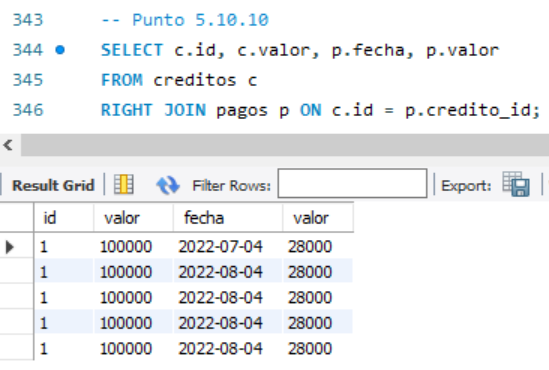
### Join Rigth

Sentencia SQL

SELECT c.id, c.valor, p.fecha, p.valor

FROM creditos c

RIGHT JOIN pagos p ON c.id = p.credito\_id;



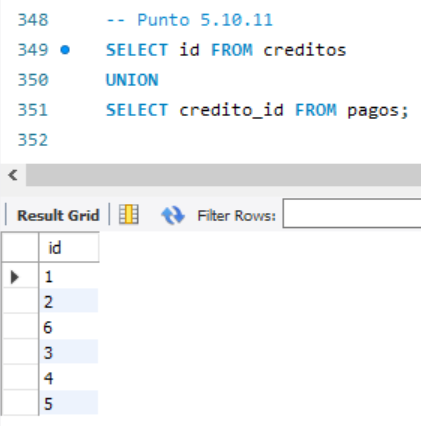
### Unión

Sentencia SQL

SELECT id FROM creditos

UNION

SELECT credito\_id FROM pagos;



### Intersect

INTERSECT no está disponible en MySQL pero se puede obtener un resultado similar con el siguiente ejemplo

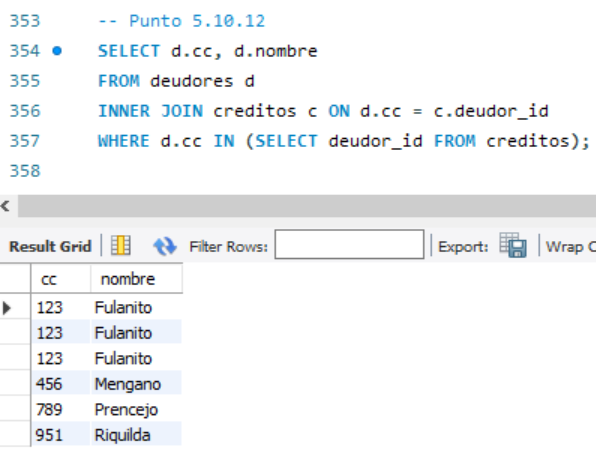
Sentencia SQL

SELECT d.cc, d.nombre

FROM deudores d

INNER JOIN creditos c ON d.cc = c.deudor\_id

WHERE d.cc IN (SELECT deudor\_id FROM creditos);



### Crear una Vista SQL

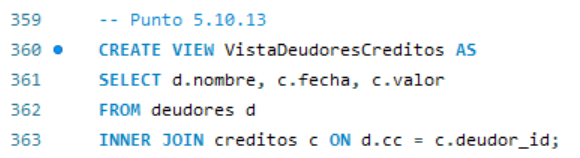
Sentencia SQL

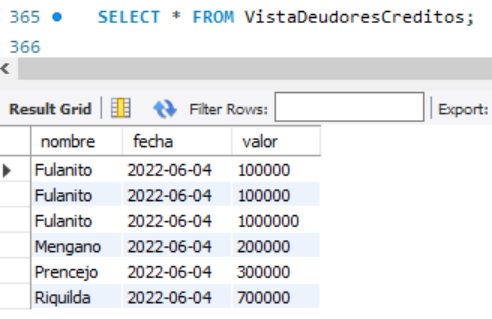
CREATE VIEW VistaDeudoresCreditos AS

SELECT d.nombre, c.fecha, c.valor

FROM deudores d

INNER JOIN creditos c ON d.cc = c.deudor\_id;





Aquí se observa la vista creada

### Usar una Vista SQL

Sentencia SQL

SELECT nombre, fecha, valor

FROM VistaDeudoresCreditos

WHERE valor > 100000;



### Editar una Vista SQL

Sentencia SQL

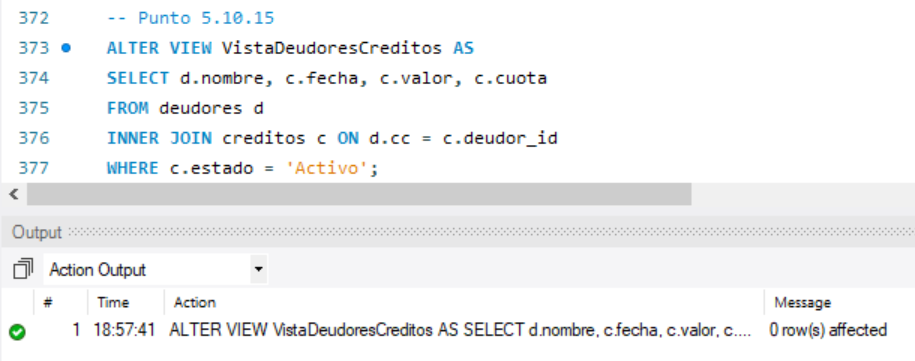
ALTER VIEW VistaDeudoresCreditos AS

SELECT d.nombre, c.fecha, c.valor, c.cuota

FROM deudores d

INNER JOIN creditos c ON d.cc = c.deudor\_id

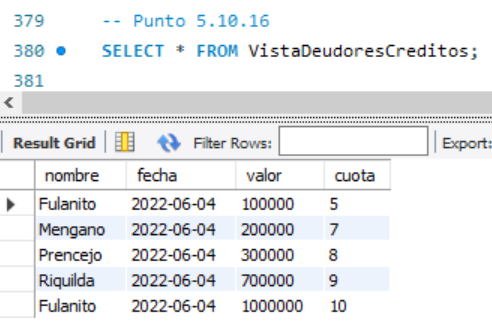
WHERE c.estado = 'Activo';



### Mostrar el contenido de una Vista SQL

Sentencia SQL

SELECT \* FROM VistaDeudoresCreditos;



### Mostrar las Vistas SQL

Para mostrar el contenido de todas las vistas en una BD se ejecuta la sentencia SQL

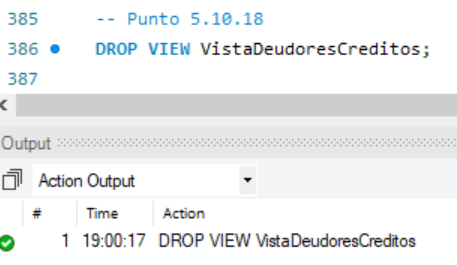
SHOW FULL TABLES WHERE Table\_type='VIEW';



### Eliminar una Vista SQL

Sentencia SQL

DROP VIEW VistaDeudoresCreditos;



### Crear una Transacción SQL

Sentencia SQL

START TRANSACTION;

#Insertar ocho nuevos registros de pago en la tabla "Pagos" al credito con id = 4

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

VALUES (DATE\_ADD((SELECT fecha FROM creditos WHERE id = 1), INTERVAL 1 MONTH), 45000, 4),

(DATE\_ADD((SELECT fecha FROM creditos WHERE id = 1), INTERVAL 2 MONTH), 45000, 4),

(DATE\_ADD((SELECT fecha FROM creditos WHERE id = 1), INTERVAL 3 MONTH), 45000, 4),

(DATE\_ADD((SELECT fecha FROM creditos WHERE id = 1), INTERVAL 4 MONTH), 45000, 4),

(DATE\_ADD((SELECT fecha FROM creditos WHERE id = 1), INTERVAL 5 MONTH), 45000, 4),

(DATE\_ADD((SELECT fecha FROM creditos WHERE id = 1), INTERVAL 6 MONTH), 45000, 4),

(DATE\_ADD((SELECT fecha FROM creditos WHERE id = 1), INTERVAL 7 MONTH), 45000, 4),

(DATE\_ADD((SELECT fecha FROM creditos WHERE id = 1), INTERVAL 8 MONTH), 45000, 4);

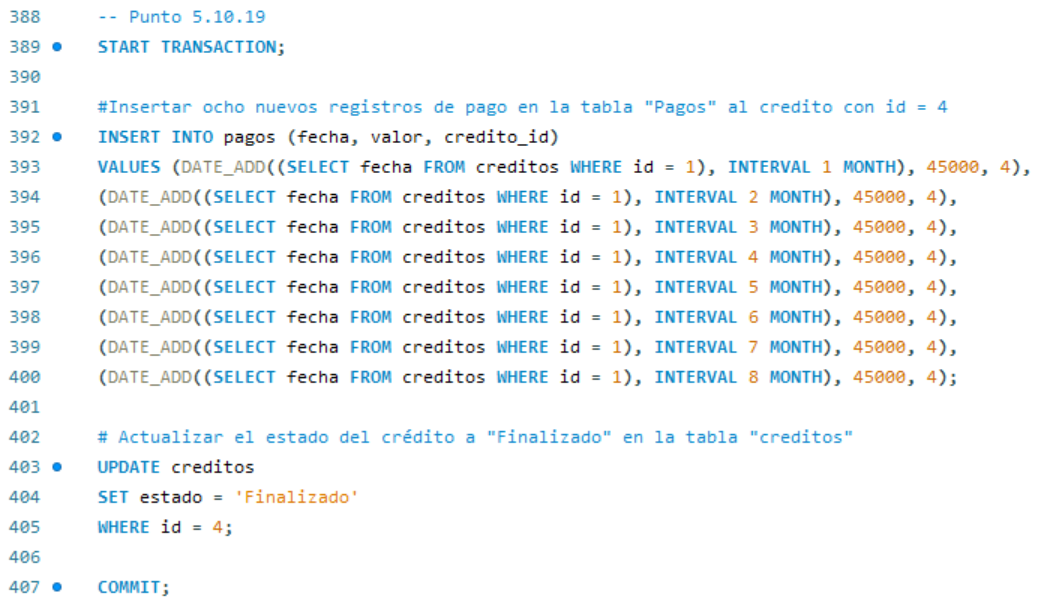
# Actualizar el estado del crédito a "Finalizado" en la tabla "creditos"

UPDATE creditos

SET estado = 'Finalizado'

WHERE id = 4;

COMMIT;



### Uso de Try para detectar cuando falla una transacción SQL

MySQL no admite el bloque TRY CATCH pero se pueden manejar los errores de una forma distinta usando procedimientos almacenados (ChickenCode, 2023).

Sentencia SQL

DELIMITER //

CREATE PROCEDURE insertar\_pago

(

p\_fecha date,

p\_valor float,

p\_credito\_id int

)

BEGIN

DECLARE EXIT HANDLER FOR SQLEXCEPTION

BEGIN

SHOW ERRORS LIMIT 1;

RESIGNAL;

ROLLBACK;

END;

START TRANSACTION;

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

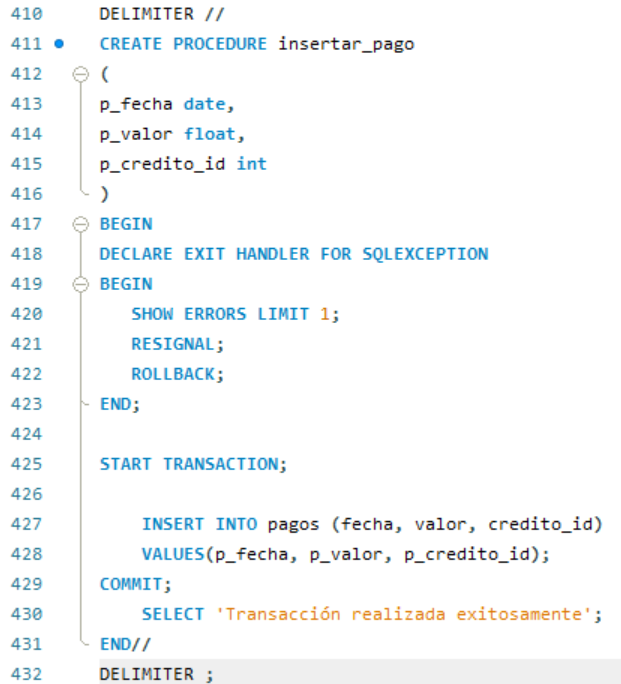
VALUES(p\_fecha, p\_valor, p\_credito\_id);

COMMIT;

SELECT 'Transacción realizada exitosamente';

END//

DELIMITER ;



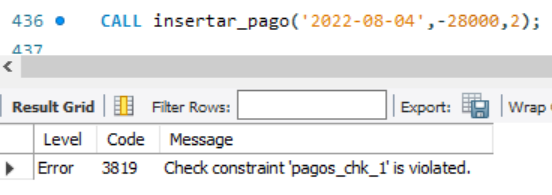
Al ejecutar el procedimiento almacenado con valores permitidos o correctos tenemos:

CALL insertar\_pago('2022-08-04',28000,2);



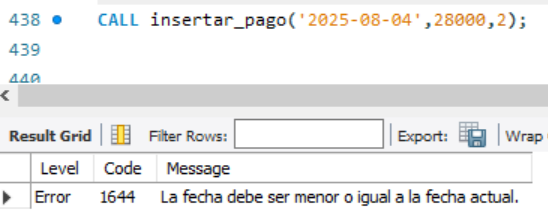
Al ejecutar el procedimiento almacenado con un valor del pago negativo tenemos:

CALL insertar\_pago('2022-08-04',-28000,2);



Al ejecutar el procedimiento almacenado con una fecha posterior a la fecha actual tenemos:

CALL insertar\_pago('2025-08-04',28000,2);



.

# Bibliografía

*ChickenCode*. (12 de 06 de 2023). Obtenido de https://youtu.be/1MhU2-LB\_Z8

Montes, A. (18 de 12 de 2022). *YouTube*. Obtenido de Restaurando backup de base de datos en PostgreSQL en formato .SQL desde consola (CMD).: https://youtu.be/0KvzfH2j9nQ

Nuggets, S. (15 de 12 de 2022). *YouTube*. Obtenido de NO ACTION Foreign Keys in PostgreSQL, learn how the NO ACTION behavior is enforced.: https://youtu.be/L1lpGsg5PbI

PostgreSql. (10 de 11 de 2022). *https://www.postgresql.org/*. Obtenido de https://www.postgresql.org/docs/current/sql-alterrole.html

Silberschatz, A., Korth, H., & Sudarshan, S. (2006). *Fundamentos de bases de datos.* Madrid: McGraw-Hill.

*tutorialesprogramacionya*. (15 de 04 de 2023). Obtenido de https://www.tutorialesprogramacionya.com/postgresqlya/index.php?inicio=50

VM, D. (18 de 12 de 2022). *YouTube*. Obtenido de Denisse VM: https://youtu.be/KIGSxLRPHw4