

FUNDACIÓN UNIVERSITARIA COLOMBO INTERNACIONAL UNICOLOMBO

FACULTAD DE INGENIERÍA

ESTUDIANTE:

Melany Olivera Atencio

DOCENTE:

Jhon Carlos Arrieta Arrieta

ASIGNATURA:

Bases de Datos

NOMBRE:

Taller 2 de Bases de Datos

CARTAGENA

2025-1

*Tabla de Contenido*

1. Presentación
2. Introducción
3. Objetivos
4. Justificación
5. Desarrollo
6. Bibliografía
7. Herramientas utilizadas
8. Sitios web y documentación
9. Presentación

Este documento presenta el desarrollo de un sistema de gestión de evaluaciones académicas para la asignatura de Computación II en la Universidad de Chile. El trabajo incluye el análisis, modelado y diseño de la base de datos que permitirá almacenar y gestionar los controles escritos, prácticas y exámenes de los estudiantes, así como la asignación de profesores a las distintas evaluaciones.

1. Introducción

El presente trabajo tiene como propósito diseñar un sistema de gestión de evaluaciones académicas que permita almacenar y consultar los resultados de los controles escritos y prácticas realizados por los estudiantes de Computación II. Se ha considerado la necesidad de registrar las notas de cada estudiante en diferentes tipos de evaluaciones, incluyendo controles escritos y prácticas individuales o grupales, con el fin de mejorar la gestión académica.

Para ello, se ha utilizado un modelo de base de datos relacional, asegurando que todas las consultas requeridas puedan ser ejecutadas de manera eficiente. Se incluyen los aspectos clave del diseño, como entidades, relaciones y restricciones, con el objetivo de garantizar la integridad y coherencia de la información.

1. Objetivos

* Diseñar un modelo de base de datos que permita gestionar los resultados de evaluaciones académicas.
* Implementar relaciones entre alumnos, profesores, asignaturas y evaluaciones.
* Garantizar la correcta organización de los datos para facilitar consultas eficientes sobre los resultados de los estudiantes.
* Permitir la gestión de prácticas individuales y grupales.
* Asegurar que los profesores puedan registrar evaluaciones y colaborar en el diseño de prácticas.

1. Justificación

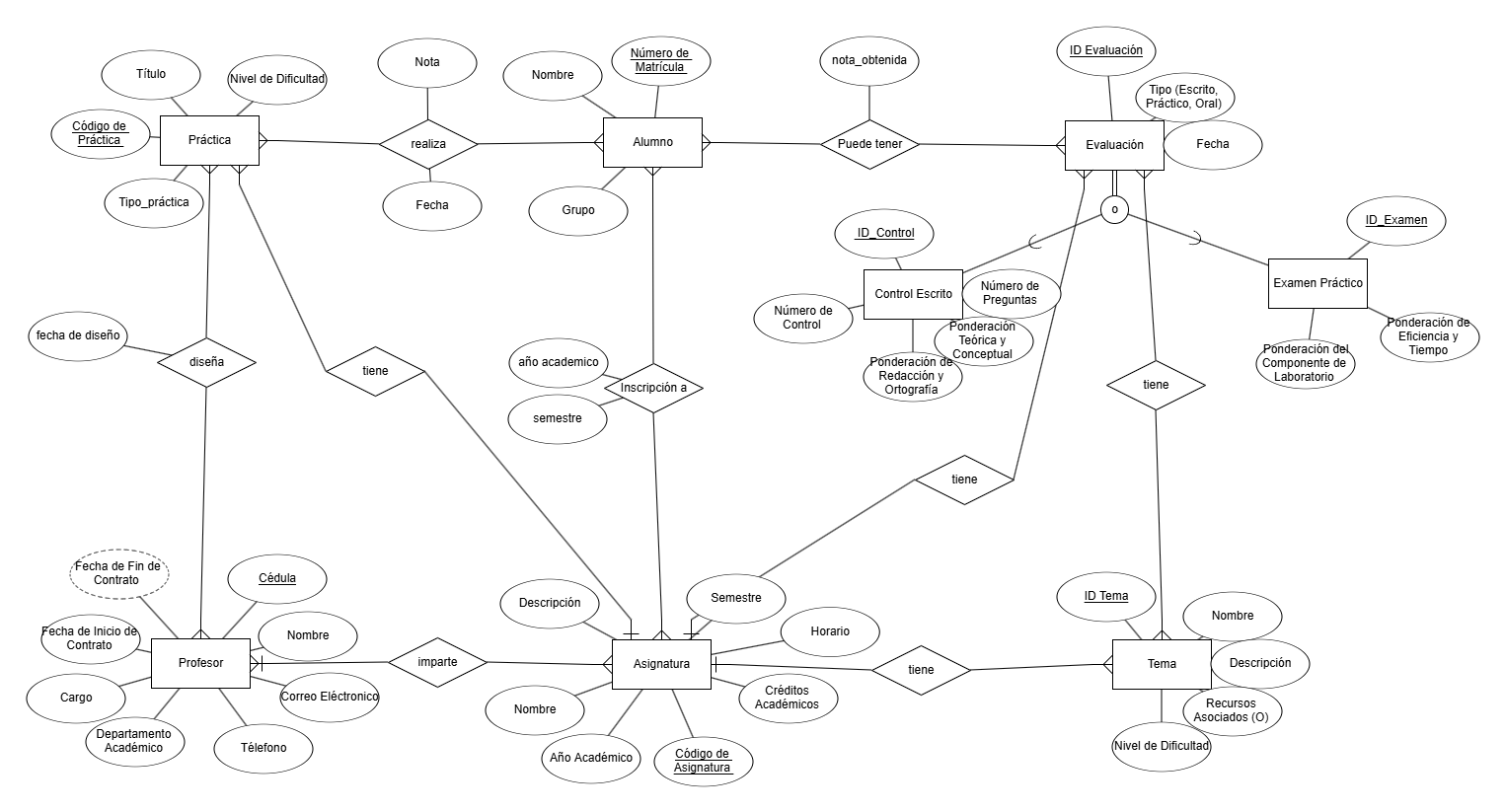
El desarrollo de este sistema es fundamental para mejorar la gestión académica de la asignatura de Computación II, ya que permite automatizar el registro y consulta de evaluaciones. Actualmente, la información de los exámenes y prácticas puede ser difícil de manejar sin una base de datos bien estructurada.

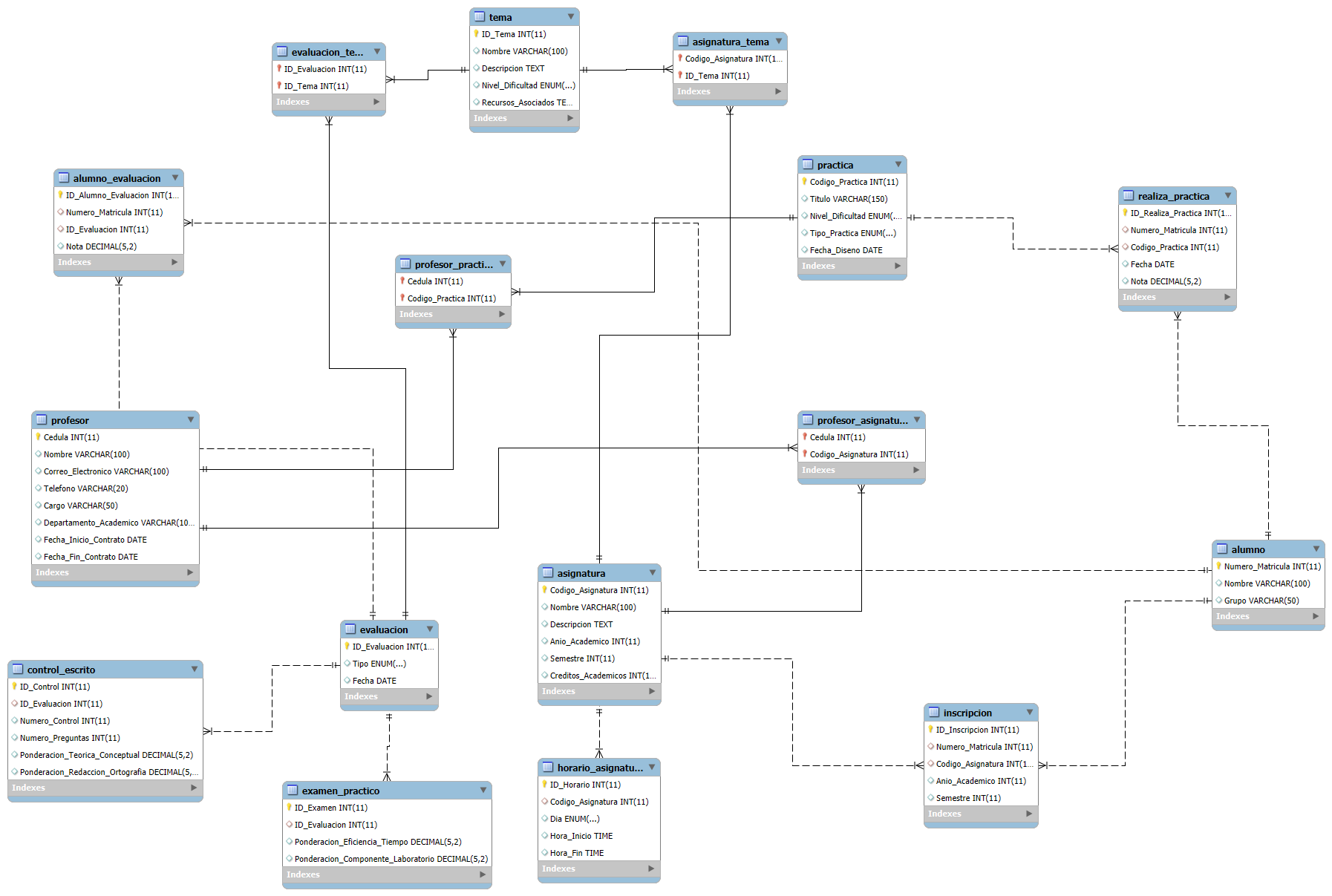
Además, este sistema permitirá a los docentes realizar un seguimiento más preciso del rendimiento de los estudiantes, identificar patrones de aprendizaje y optimizar los procesos de evaluación. Asimismo, facilitará la asignación de profesores a distintas prácticas y exámenes, promoviendo la colaboración en el diseño de actividades evaluativas.

El diseño de la base de datos también asegura la posibilidad de escalar el sistema en el futuro, integrando nuevas funcionalidades como reportes estadísticos y análisis de desempeño estudiantil.

1. Desarrollo

**Modelo Entidad Relación**



**Modelo Relacional** 

¿Qué es una vista SQL en una base de datos relacional y qué problema resuelve?

Una Vista (VIEW) es como una ventana o una tabla virtual que muestra datos de una o varias tablas, según una consulta que tú diseñas.

¿Para qué sirve o qué problemas resuelve?

* Evita repetir consultas complejas: Si siempre necesitas hacer una consulta complicada con muchas tablas, puedes crear una vista y solo haces: SELECT \* FROM mi\_vista;
* Mejora la seguridad: Puedes mostrar solo las columnas que quieras, así proteges información sensible (como contraseñas, sueldos, etc.).
* Mantiene la lógica del negocio: Si necesitas mostrar datos de forma específica, puedes dejar esa lógica en la vista y no escribirla una y otra vez.
* Organiza mejor tu base de datos: Te ayuda a mantener todo más claro y separado: las tablas con datos, y las vistas con lo que los usuarios necesitan ver.

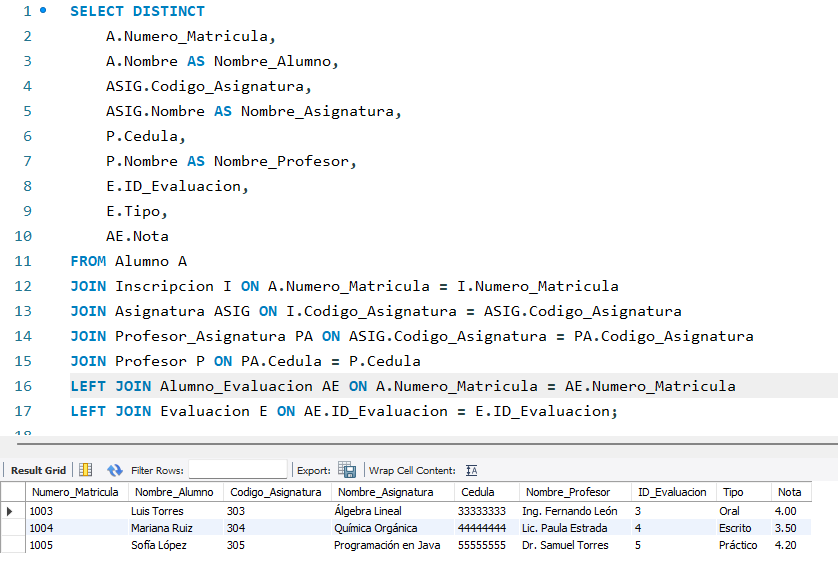
En Oracle o SQL Server:

CREATE OR REPLACE VIEW nombre\_vista AS ...

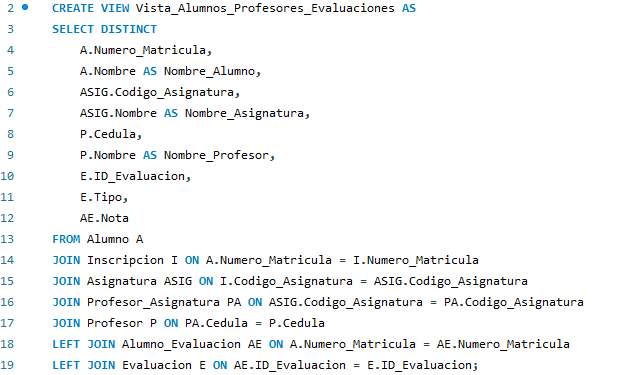
En **PostgreSQL** o **MySQL**

CREATE VIEW nombre\_vista AS ...

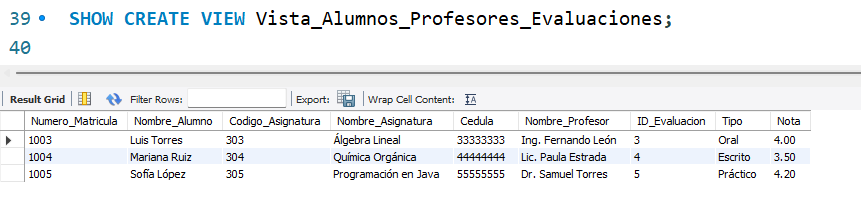
1. Diseñar una consulta multitabla y probarla

****

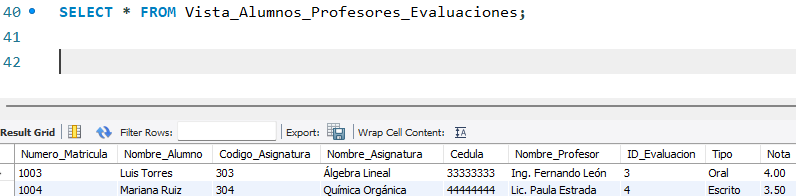
1. Usar la consulta anterior y crear con ella una Vista SQL

****

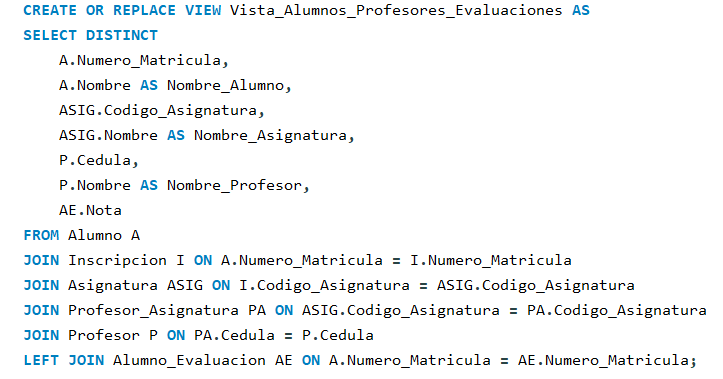
1. Mostrar la Vista SQL creada anteriormente

****

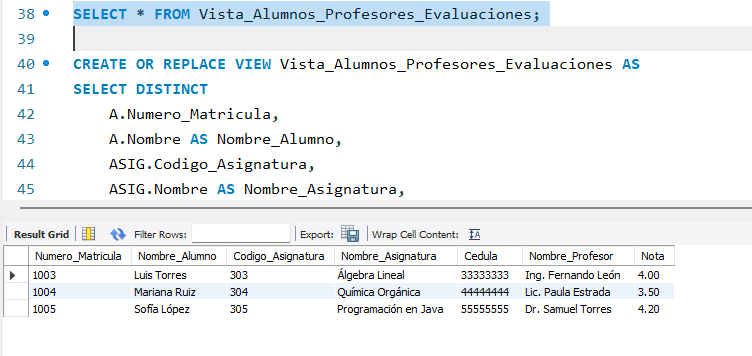
1. Ejecutar la Vista SQL creada anteriormente



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



1. Volver a ejecutar la Vista SQL



1. 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:

a. Ventajas de usar vistas en MySQL

* **Facilitan el trabajo con consultas complejas:** Si una consulta es muy larga o complicada, en lugar de escribirla una y otra vez, podemos crear una vista y consultarla como si fuera una tabla normal.
* **Mejoran la seguridad y el control de acceso:** Podemos crear vistas que solo muestren ciertas columnas o filas, ocultando información sensible a usuarios que no deben acceder a todos los datos.
* **Hacen más fácil el mantenimiento**: Si en algún momento necesitamos modificar una consulta utilizada en varios lugares, solo hay que actualizar la vista en vez de cambiar múltiples consultas manualmente.
* **Ayudan a estructurar mejor los datos:** Nos permiten presentar la información de manera más organizada sin modificar la estructura original de las tablas.

b. Desventajas del uso de Vistas SQL en MySQL:

* **No mejoran el rendimiento por sí solas:**  
  MySQL no almacena los resultados de una vista, así que cada vez que la consultamos, el sistema vuelve a ejecutar la consulta original, lo que puede hacerla lenta si maneja muchos datos.
* **Tienen limitaciones para modificar datos:**  
  No siempre se pueden hacer cambios (INSERT, UPDATE, DELETE) directamente sobre una vista, especialmente si usa varias tablas, funciones agregadas (SUM, AVG, COUNT), o DISTINCT.
* **No permiten ordenar resultados dentro de su definición:**  
  No se puede usar ORDER BY dentro de una vista a menos que también se use LIMIT. Si queremos ordenar los datos, debemos hacerlo cuando consultemos la vista.
* **No existen vistas materializadas en MySQL:**  
  A diferencia de otros motores de bases de datos, MySQL no permite guardar el resultado de una vista para consultas más rápidas. Siempre se recalculan los datos en cada consulta.

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.

 **No se pueden modificar algunas vistas:**  
Si una vista tiene JOIN, GROUP BY, HAVING, o usa funciones como SUM o AVG, no podremos actualizar ni eliminar datos desde la vista.

 **No se pueden usar ciertas instrucciones dentro de una vista:**  
Algunas funciones como TRUNCATE no están permitidas dentro de una vista.

 **Si cambia la estructura de las tablas, puede afectar la vista:**  
Si eliminamos o cambiamos columnas en una tabla usada en la vista, esta puede dejar de funcionar o devolver errores.

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

Sentencia SQL:

CREATE DATABASE prueba\_transacciones;

USE prueba\_transacciones;



* 1. Crear tablas **Deudores**(**cc**, clave, nombre, apellido, email). email es único, todos son obligatorios
  2. Crear tablas **Creditos**(id, fecha, valor, cuotas, interes, estado, deudor\_id), todos son obligatorios, estado por defecto es Activo
  3. 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 Deudores (

cedula INT PRIMARY KEY, -- Cédula como clave primaria

clave VARCHAR(255) NOT NULL,

nombre VARCHAR(100) NOT NULL,

apellido VARCHAR(100) NOT NULL,

email VARCHAR(255) UNIQUE NOT NULL

);

CREATE TABLE Creditos (

id INT AUTO\_INCREMENT PRIMARY KEY, -- Identificador único

fecha DATE NOT NULL,

valor DECIMAL(10,2) NOT NULL,

cuotas INT NOT NULL,

interes DECIMAL(5,2) NOT NULL,

estado ENUM('Activo', 'Pagado', 'Cancelado') DEFAULT 'Activo',

deudor\_id INT NOT NULL,

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

);

CREATE TABLE Pagos (

id INT AUTO\_INCREMENT PRIMARY KEY,

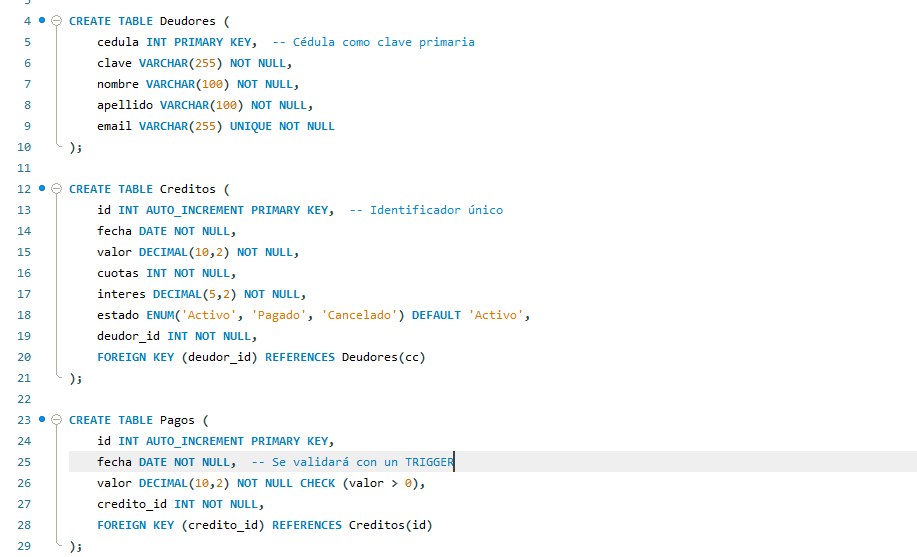
fecha DATE NOT NULL, -- Se validará con un TRIGGER

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

credito\_id INT NOT NULL,

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

);



DELIMITER $$

CREATE TRIGGER before\_insert\_pagos

BEFORE INSERT ON Pagos

FOR EACH ROW

BEGIN

IF NEW.fecha > CURDATE() THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Error: La fecha del pago no puede ser mayor a la fecha actual.';

END IF;

END $$

DELIMITER ;

DELIMITER $$

CREATE TRIGGER before\_update\_pagos

BEFORE UPDATE ON Pagos

FOR EACH ROW

BEGIN

IF NEW.fecha > CURDATE() THEN

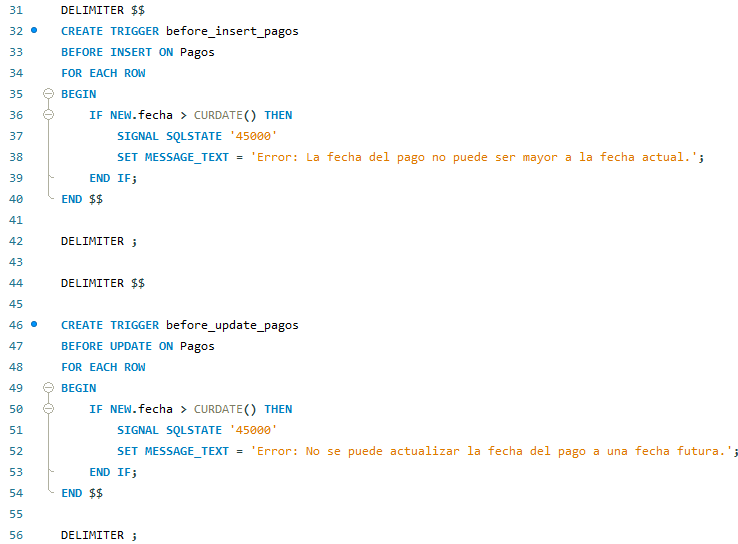
SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Error: No se puede actualizar la fecha del pago a una fecha futura.';

END IF;

END $$

DELIMITER ;



* 1. Insertar un al menor un Deudor:

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

* 1. Mostrar todos los Deudores insertados en la BD
  2. Insertar al menos un Crédito:

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

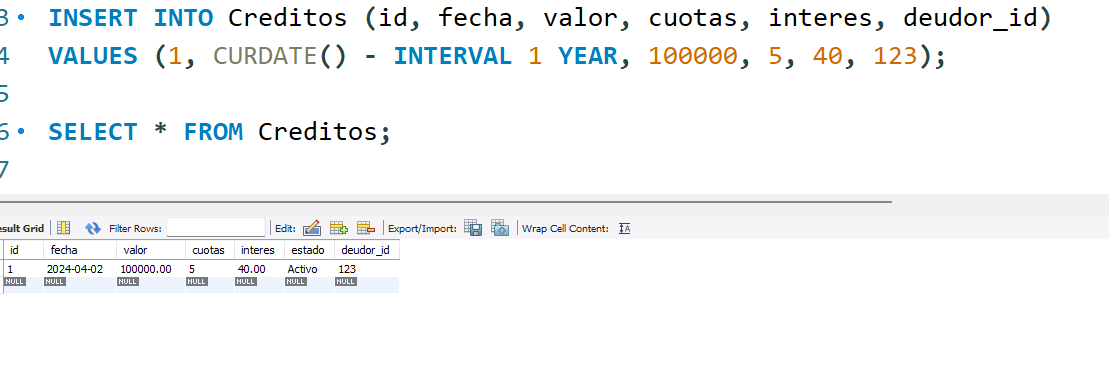
* 1. Mostrar todos los Créditos insertados en la BD

Sentencia SQL:

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

VALUES (1, CURDATE() - INTERVAL 1 YEAR, 100000, 5, 40, 123);

SELECT \* FROM Creditos;



* 1. Insertar 5 pagos así (no modifique ningún valor del ejemplo, colocar las fechas correspondientes, el formato de fecha correspondiente para bd):
     1. id=1, fecha=**un mes después del crédito**, valor=28000, credito\_id=1
     2. id=2, fecha=**un mes después de la fecha actual,** valor=28000, credito\_id=1
     3. id=3, fecha=**un mes después del primer pago,** valor=28000, credito\_id=1
     4. id=4, fecha=**un mes después del primer pago,** valor=28000, credito\_id=1
     5. id=5, fecha=**un mes después del primer pago,** valor=**-**28000, credito\_id=1

Sentencia SQL:

-- Pago 1: Un mes después del crédito

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

VALUES (1, DATE\_ADD((SELECT fecha FROM Creditos WHERE id = 1), INTERVAL 1 MONTH), 28000, 1);

-- Pago 2: Un mes después de la fecha actual

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

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

SET @fecha\_pago1 = (SELECT fecha FROM Pagos WHERE id = 1);

-- Pago 3: Un mes después del primer pago

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

VALUES (3, DATE\_ADD(@fecha\_pago1, INTERVAL 1 MONTH), 28000, 1);

-- Pago 4: Un mes después del primer pago

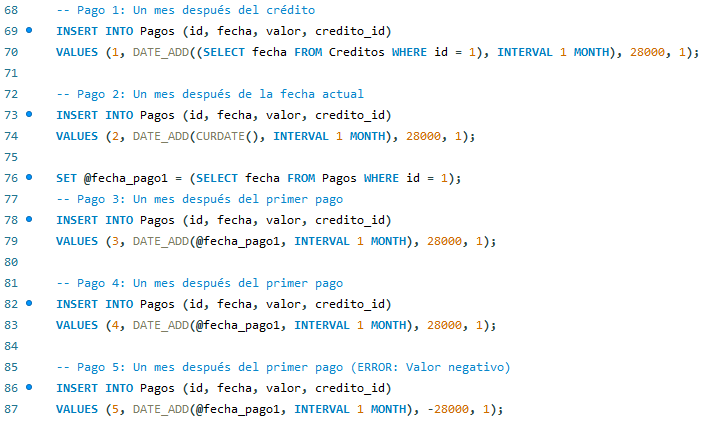
INSERT INTO Pagos (id, fecha, valor, credito\_id)

VALUES (4, DATE\_ADD(@fecha\_pago1, INTERVAL 1 MONTH), 28000, 1);

-- Pago 5: Un mes después del primer pago (ERROR: Valor negativo)

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

VALUES (5, DATE\_ADD(@fecha\_pago1, INTERVAL 1 MONTH), -28000, 1);



**Pago 2:** Dará error porque el TRIGGER before\_insert\_pagos bloquea cualquier inserción con una fecha futura. Como el pago se intenta registrar un mes después de la fecha actual (CURDATE()), el SIGNAL SQLSTATE '45000' lanzará un error y no se insertará.

**Pago 5:** Dará error porque el CHECK de la columna valor no permite valores negativos. Como el pago es de -28000, se violará la restricción y no se insertará.

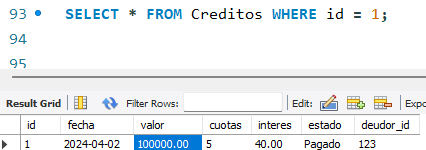
* 1. Cambiar o actualizar el estado del Crédito con id=1, para que ahora sea Finalizado

UPDATE Creditos SET estado = 'Pagado' WHERE id = 1;



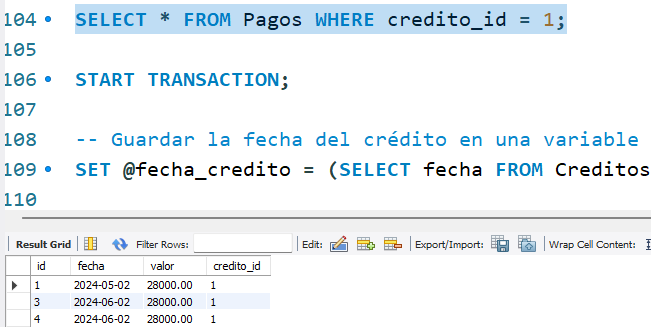
* 1. Mostrar los datos del Crédito con id=1

SELECT \* FROM Creditos WHERE id = 1;



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

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

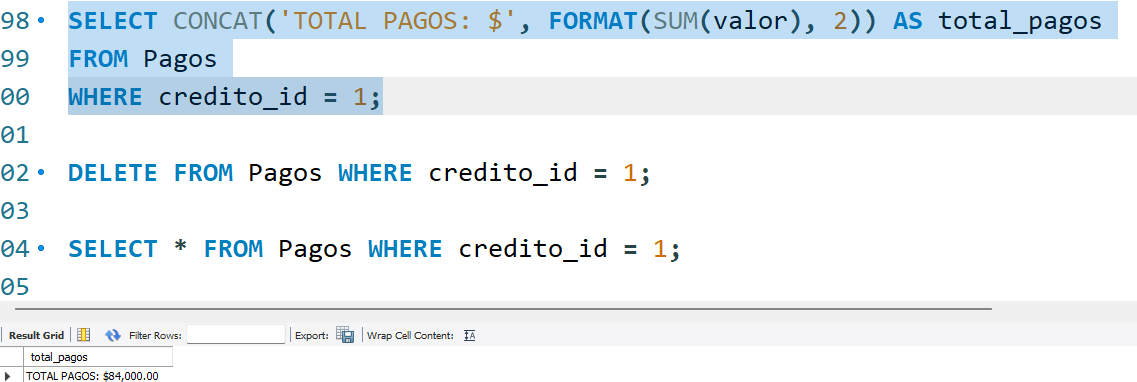


* 1. Totalizar el total de valores pagados para el crédito con id=1, para que aparezca así: TOTAL PAGOS: $

SENTENCIA SQL:

SELECT CONCAT('TOTAL PAGOS: $', FORMAT(SUM(valor), 2)) AS total\_pagos

FROM Pagos WHERE credito\_id = 1;



* 1. Elimine todos los pagos de crédito con ID: 1

SENTENCIA SQL:

DELETE FROM Pagos WHERE credito\_id = 1;

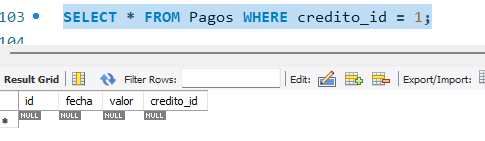




* 1. Mostrar todos los pagos que existen para el crédito con id=1

SENTENCIA SQL:

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



* 1. Inicie una TRANSACCION



Iniciamos una transacción, lo que nos permite agrupar varias operaciones.

* 1. Dentro de la transacción volver a insertar los mismos pagos anteriores:
     1. id=1, fecha=**un mes después del crédito**, valor=28000, credito\_id=1
     2. id=2, fecha=**un mes después de la fecha actual,** valor=28000, credito\_id=1
     3. id=3, fecha=**un mes después del primer pago,** valor=28000, credito\_id=1
     4. id=4, fecha=**un mes después del primer pago,** valor=28000, credito\_id=1
     5. id=5, fecha=**un mes después del primer pago,** valor=**-**28000, credito\_id=1

SENTENCIA SQL:

-- Guardar la fecha del crédito en una variable

SET @fecha\_credito = (SELECT fecha FROM Creditos WHERE id = 1);

-- Pago 1: Un mes después del crédito

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

VALUES (1, DATE\_ADD(@fecha\_credito, INTERVAL 1 MONTH), 28000, 1);

-- Pago 2: Un mes después de la fecha actual

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

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

-- Obtener la fecha del primer pago para los siguientes pagos

SET @fecha\_pago1 = (SELECT fecha FROM Pagos WHERE id = 1);

-- Pago 3: Un mes después del primer pago

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

VALUES (3, DATE\_ADD(@fecha\_pago1, INTERVAL 1 MONTH), 28000, 1);

-- Pago 4: Un mes después del primer pago

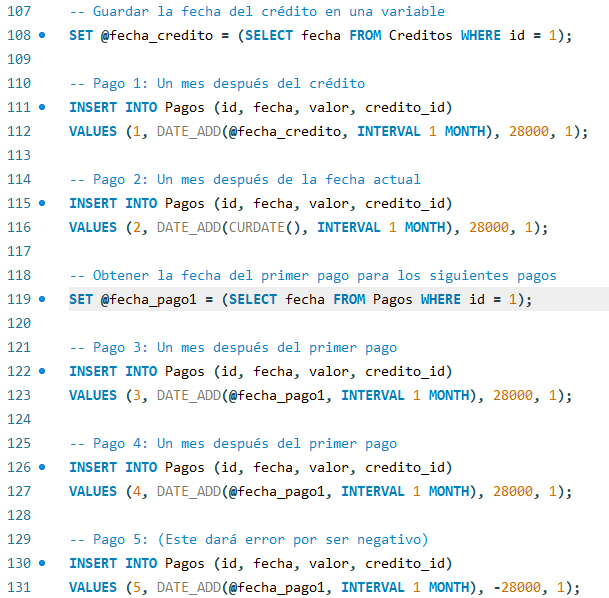
INSERT INTO Pagos (id, fecha, valor, credito\_id)

VALUES (4, DATE\_ADD(@fecha\_pago1, INTERVAL 1 MONTH), 28000, 1);

-- Pago 5: (Este dará error por ser negativo)

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

VALUES (5, DATE\_ADD(@fecha\_pago1, INTERVAL 1 MONTH), -28000, 1);



Intentamos insertar los pagos. El pago 2 y 5 fallaron.

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

SENTENCIA SQL:

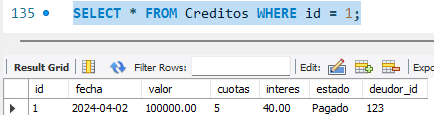
UPDATE Creditos SET estado = 'Pagado' WHERE id = 1;



Cambiamos el estado del crédito.

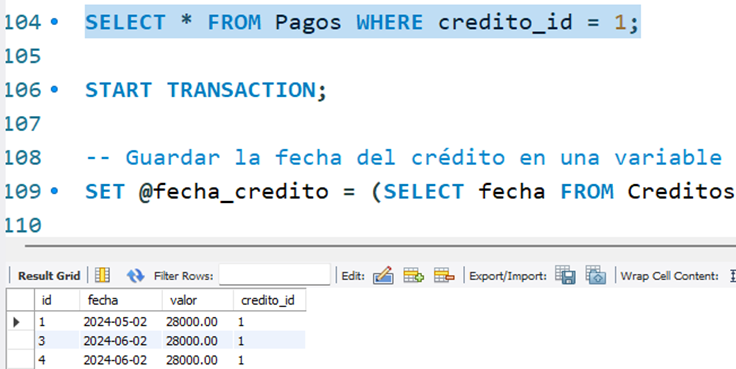


* 1. Mostrar los datos del Crédito con id=1



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

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

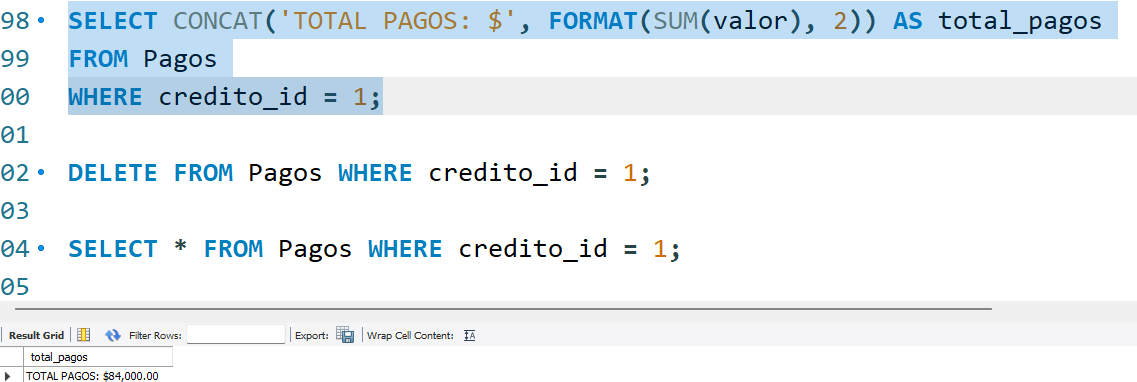
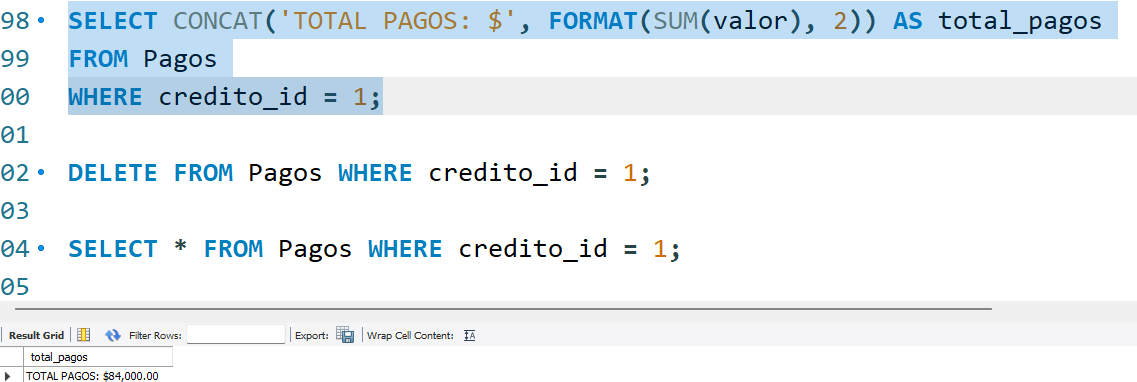


* 1. Totalizar el total de valores pagados para el crédito con id=1, para que aparezca así: TOTAL PAGOS: $

SELECT CONCAT('TOTAL PAGOS: $', FORMAT(SUM(valor), 2)) AS total\_pagos

FROM Pagos

WHERE credito\_id = 1;



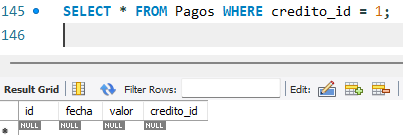
* 1. Cancelar la transaccion (rollblack)

ROLLBACK;



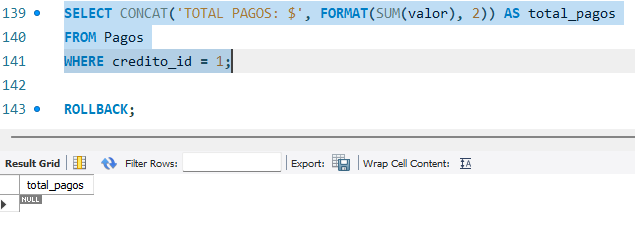
* 1. Mostrar todos los Pagos insertados al Crédito con ID=1 insertados en la BD

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



Aquí no debe aparecer ningún pago porque se deshicieron los cambios con ROLLBACK.

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



**El resultado debe ser** $0 **porque eliminamos todos los pagos con** ROLLBACK**.**