Nombre del taller:

PARCIAL FINAL DB

Estudiante:

DEIVID DANIEL PATIÑO BLANCO

Programa:

TECNOLOGIA EN DESARROLLO DE SISTEMAS DE INFORMACION Y DE SOFTWARE

Asignatura:

BASES DE DATOS 5TO SEMESTRE

Docente:

JHON CARLOS ARRIETA ARRIETA



CARTAGENA DE INDIAS D.T. Y C.

**Tabla de contenido**

1. Presentación
2. Tabla de contenido
3. Introducción
4. Objetivos
5. Justificación
6. Desarrollo  
    6.1 Consultas Multitabla

6.2 Vistas

6.3 Transacciones

6.4 Enlace del repositorio GitHub

1. Presentación

Este documento presenta el desarrollo del taller de la Unidad 2 de la asignatura Bases de Datos II, orientado al uso de consultas multitabla, vistas y transacciones en el sistema de lavadero de autos. La práctica se realizó utilizando el motor de base de datos MySQL y parte del modelo previamente diseñado en Bases de Datos .

**3. Introducción**

El presente trabajo busca aplicar los conocimientos adquiridos en la asignatura Bases de Datos, mediante la implementación de consultas multitabla, vistas, restricciones y transacciones. La actividad práctica se desarrolla sobre una base de datos previamente construida, que simula el funcionamiento de un sistema para la gestión de un lavadero de autos.

4. Objetivos

Objetivo general:

Desarrollar habilidades en la implementación de operaciones avanzadas en bases de datos relacionales utilizando MySQL.

Objetivos específicos:

* Aplicar consultas multitabla mediante JOIN y subconsultas.
* Crear vistas que simplifiquen el acceso a la información.
* Definir restricciones mediante la cláusula CHECK.
* Ejecutar transacciones que aseguren la integridad de los datos.

5. Justificación

En el contexto actual, el manejo de información distribuida entre múltiples tablas y el uso de vistas y transacciones es esencial para la administración eficiente de sistemas de información. Este taller permite fortalecer esas habilidades en un entorno práctico y cercano a la realidad, utilizando MySQL, uno de los motores más usados en la industria.

6. Desarrollo

6.1 Consultas Multitabla

Se realizaron consultas multitabla que permiten visualizar información relacionada con vehículos, clientes, servicios, empleados, etc., utilizando INNER JOIN, LEFT JOIN y subconsultas. Estas consultas simulan reportes y operaciones comunes de consulta en el sistema.

Ejemplo:  
**SELECT CONCAT(c.Nombres, ' ', c.Apellidos) AS Nombre\_Cliente, a.Marca AS Marca\_Auto, a.Tipo AS Tipo\_Auto, a.Color AS Color\_Auto, ov.Fecha\_Solicitud, ov.Total**

**FROM Orden\_Venta ov**

**INNER JOIN Cliente c ON ov.ID\_Cliente = c.ID\_Cliente**

**INNER JOIN Automovil a ON ov.ID\_Automovil = a.ID\_Automovil;**

6.2 Vistas

Se creó una vista llamada Vista\_Ordenes\_Clientes que permite observar fácilmente la información combinada de clientes, automóviles y órdenes de venta.

Código usado:

**CREATE VIEW Vista\_Ordenes\_Clientes AS**

**SELECT**

**CONCAT(c.Nombres, ' ', c.Apellidos) AS Nombre\_Cliente,**

**a.Marca AS Marca\_Auto,**

**a.Tipo AS Tipo\_Auto,**

**a.Color AS Color\_Auto,**

**ov.Fecha\_Solicitud,**

**ov.Total**

**FROM Orden\_Venta ov**

**INNER JOIN Cliente c ON ov.ID\_Cliente = c.ID\_Cliente**

**INNER JOIN Automovil a ON ov.ID\_Automovil = a.ID\_Automovil;**

6.3 Transacciones

Se desarrolló una transacción que intentaba insertar múltiples pagos a un crédito, algunos con errores para observar la validación y el uso del rollback.

Código aplicado:

**START TRANSACTION;**

**SELECT fecha INTO @fecha\_credito FROM Creditos WHERE id = 1;**

**INSERT INTO Pagos VALUES (1, DATE\_ADD(@fecha\_credito, INTERVAL 1 MONTH), 28000, 1);**

**INSERT INTO Pagos VALUES (2, DATE\_ADD(CURDATE(), INTERVAL 1 MONTH), 28000, 1);**

**INSERT INTO Pagos VALUES (3, DATE\_ADD(@fecha\_credito, INTERVAL 2 MONTH), 28000, 1);**

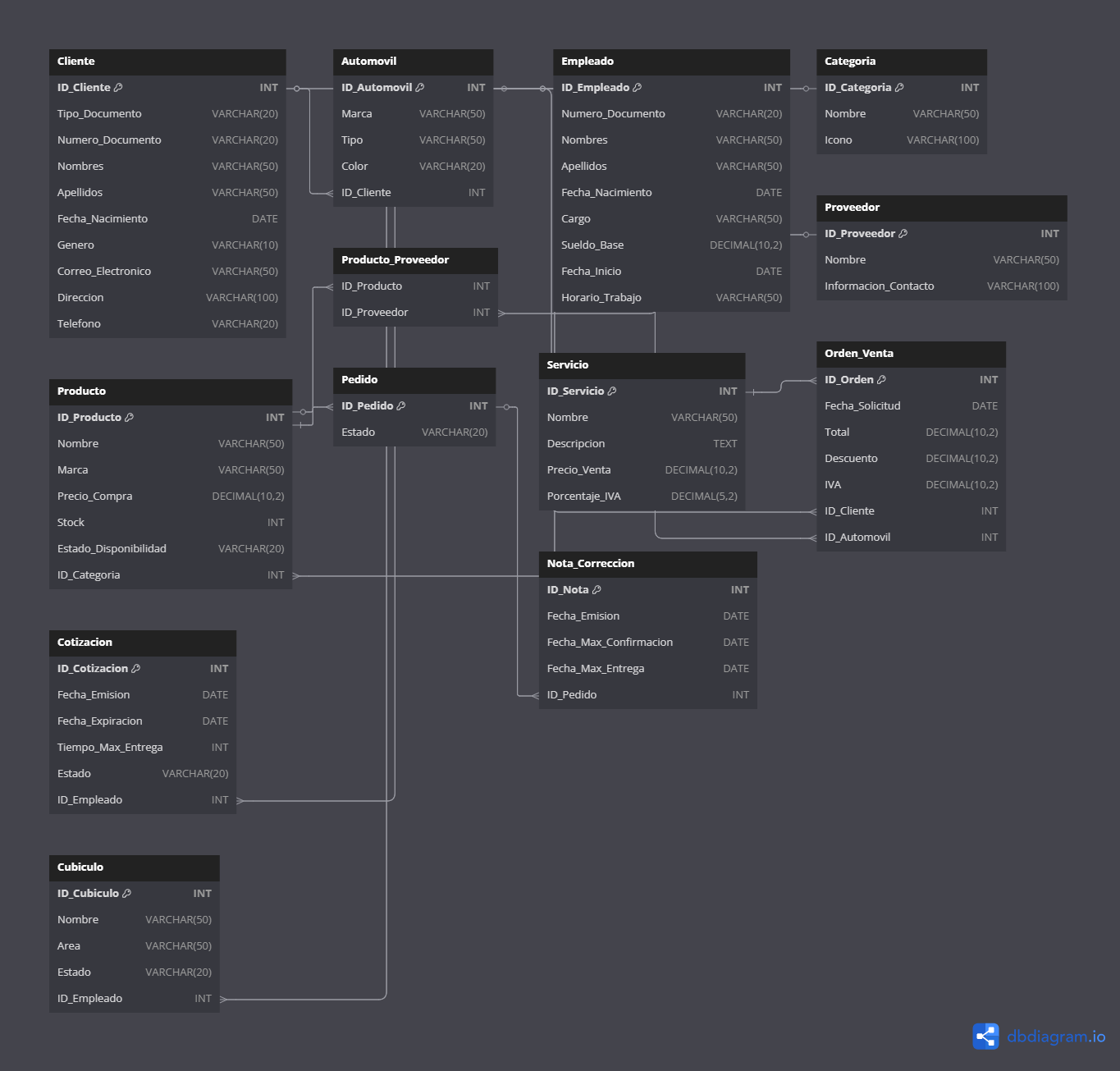
**INSERT INTO Pagos VALUES (4, DATE\_ADD(@fecha\_credito, INTERVAL 2 MONTH), 28000, 1);**

**INSERT INTO Pagos VALUES (5, DATE\_ADD(@fecha\_credito, INTERVAL 2 MONTH), -28000, 1);**

**UPDATE Creditos SET estado = 'Finalizado' WHERE id = 1;**

**ROLLBACK;**

**6.4 Enlace del repositorio GitHub**

[DeividPatino/TALLER2---2-CORTE---CONSULTAS-MULTITABLA---BD: Taller Unidad 2 - Bases de Datos 2](https://github.com/DeividPatino/TALLER2---2-CORTE---CONSULTAS-MULTITABLA---BD)

**INFORMACIÓN SOBRE LA ACTIVIDAD ACADÉMICA - UNIDAD 2**

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 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.

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

Explicación: Una vista en SQL es como una ventana virtual a una consulta. No almacena datos por sí sola, sino que muestra resultados de una consulta SELECT como si fuera una tabla.

¿Qué as͏unto ͏soluciona? **Facili͏ta preguntas complicadas:** una ͏pregunta grande se p͏uede guardar como vista para no volver a hacerla.

**Seg͏ur͏idad y c͏ontrol de en͏trada:** se p͏uede enseñar solo parte de los dato͏s (p͏or ej͏emplo, no mostrar salarios).

**Ab͏stracción d͏e información:** los ͏us͏uarios ve͏n ͏la vi͏sta co͏mo una ͏tabla; no tienen que conocer las unen, filtro͏s, o cuentas detrás.

1. 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).

Instrucción SQL:

**SELECT**

**CONCAT(c.Nombres, ' ', c.Apellidos) AS Nombre\_Cliente,**

**a.Marca AS Marca\_Auto,**

**a.Tipo AS Tipo\_Auto,**

**a.Color AS Color\_Auto,**

**ov.Fecha\_Solicitud,**

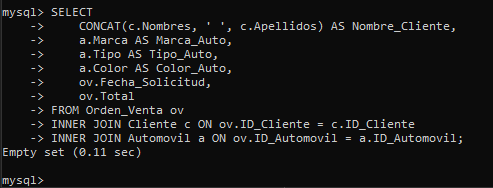
**ov.Total**

**FROM Orden\_Venta ov**

**INNER JOIN Cliente c ON ov.ID\_Cliente = c.ID\_Cliente**

**INNER JOIN Automovil a ON ov.ID\_Automovil = a.ID\_Automovil;**

Captura de pantalla:



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

Instrucción SQL:

**CREATE VIEW Vista\_Ordenes\_Clientes AS**

**SELECT**

**CONCAT(c.Nombres, ' ', c.Apellidos) AS Nombre\_Cliente,**

**a.Marca AS Marca\_Auto,**

**a.Tipo AS Tipo\_Auto,**

**a.Color AS Color\_Auto,**

**ov.Fecha\_Solicitud,**

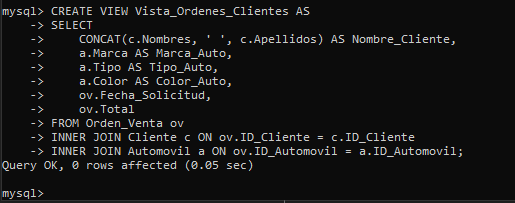
**ov.Total**

**FROM Orden\_Venta ov**

**INNER JOIN Cliente c ON ov.ID\_Cliente = c.ID\_Cliente**

**INNER JOIN Automovil a ON ov.ID\_Automovil = a.ID\_Automovil;**

Captura de pantalla:

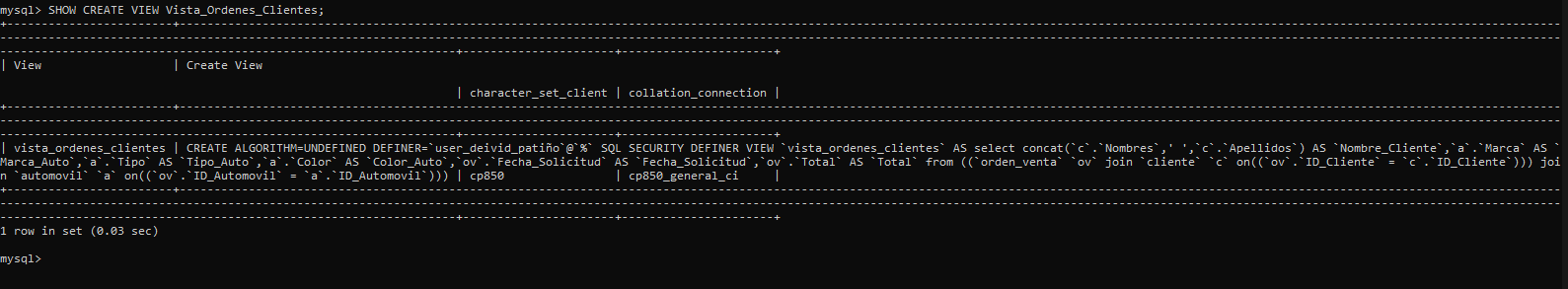


1. Mostrar la Vista SQL creada anteriormente

Instrucción SQL:

**SHOW CREATE VIEW Vista\_Ordenes\_Clientes;**

Captura de pantalla:

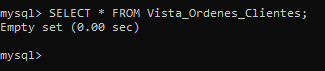


1. Ejecutar la Vista SQL creada anteriormente

Instrucción SQL:

**SELECT \* FROM Vista\_Ordenes\_Clientes;**

Captura de pantalla:



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

Instrucción SQL:

**DROP VIEW IF EXISTS Vista\_Ordenes\_Clientes;**

**CREATE VIEW Vista\_Ordenes\_Clientes AS**

**SELECT**

**CONCAT(c.Nombres, ' ', c.Apellidos) AS Nombre\_Cliente,**

**a.Marca AS Marca\_Auto,**

**a.Tipo AS Tipo\_Auto,**

**a.Color AS Color\_Auto,**

**ov.Fecha\_Solicitud,**

**ov.Total,**

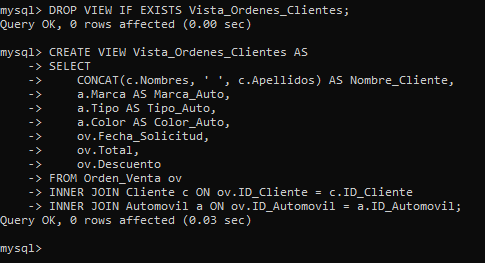
**ov.Descuento**

**FROM Orden\_Venta ov**

**INNER JOIN Cliente c ON ov.ID\_Cliente = c.ID\_Cliente**

**INNER JOIN Automovil a ON ov.ID\_Automovil = a.ID\_Automovil;**

Captura de pantalla:

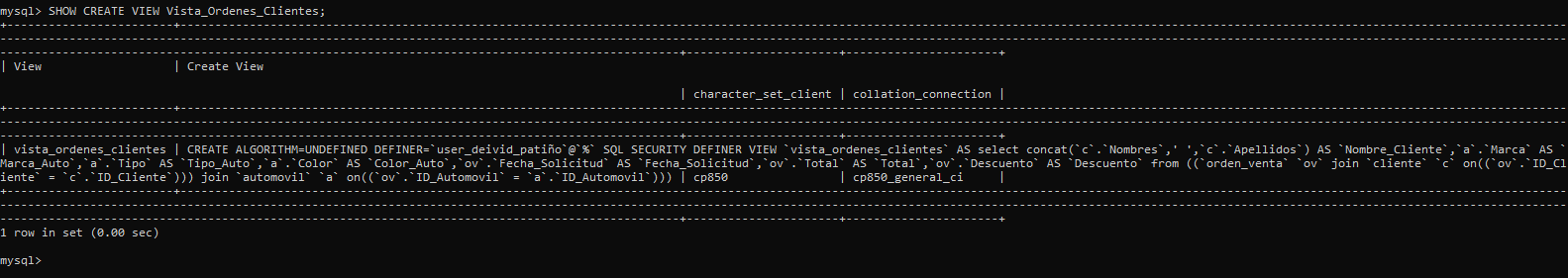


1. Volver a ejecutar la Vista SQL

Instrucción SQL:

**SHOW CREATE VIEW Vista\_Ordenes\_Clientes;**

Captura de pantalla:



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

Explicación: Las vistas permiten organizar mejor las consultas complejas y reutilizarlas fácilmente también ayudan a mejorar la seguridad ya que se pueden ocultar columnas sensibles y mostrar solo la información necesaria además, se pueden usar como si fueran tablas normales en otras consultas, lo que facilita el trabajo.

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

Explicación: MySQL no tiene buen soporte para vistas actualizables

especialmente si contienen JOINS En algunos casos no se pueden usar para insertar o actualizar datos lo que limita su utilidad y si se usan demasiado en consultas grandes pueden afectar el rendimiento.

* 1. 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.

Explicación: Hay cosas que las vistas no permiten como usar ORDER BY directamente en su definición también pueden fallar con funciones como RAND o NOW. Si la vista es muy compleja no siempre se puede actualizar o insertar datos desde ella.

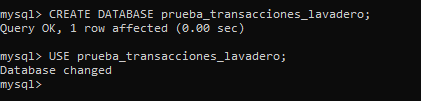
1. Para el desarrollo de los siguientes puntos referentes al tema de **Transacciones**, ustedes como grupo deben realizar las siguientes tareas sobre OracleSql, SQLServer y PostgreSQL y MySQL:
   1. Crear una base de datos llamada prueba\_transacciones\_grupoXYZ, siendo XYZ el nombre del ejercicio escogido como equipo.

Instrucción SQL:

**CREATE DATABASE prueba\_transacciones\_lavadero;**

**USE prueba\_transacciones\_lavadero;**

Captura de pantalla:



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

Instrucción SQL:

**CREATE TABLE Deudores (**

**cc VARCHAR(20) PRIMARY KEY,**

**clave VARCHAR(50) NOT NULL,**

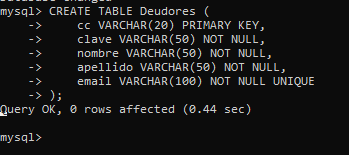
**nombre VARCHAR(50) NOT NULL,**

**apellido VARCHAR(50) NOT NULL,**

**email VARCHAR(100) NOT NULL UNIQUE**

**);**

Captura de pantalla:



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

Instrucción SQL:

**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,**

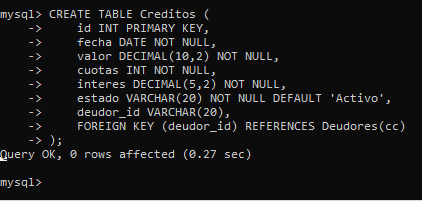
**estado VARCHAR(20) NOT NULL DEFAULT 'Activo',**

**deudor\_id VARCHAR(20),**

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

**);**

Captura de pantalla:



* 1. 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 mauyor que la fecha actual

Instrucción SQL:

**CREATE TABLE Pagos (**

**id INT PRIMARY KEY,**

**fecha DATE NOT NULL CHECK (fecha <= CURDATE()),**

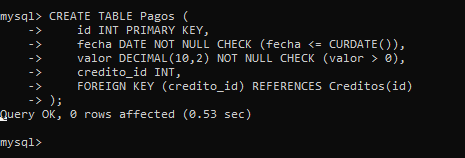
**valor DECIMAL(10,2) NOT NULL CHECK (valor > 0),**

**credito\_id INT,**

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

**);**

Captura de pantalla:



* 1. Insertar un al menor un Deudor:

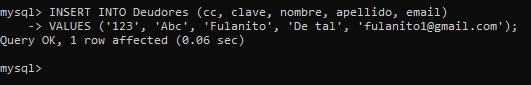
El primero con cc=123, calve=Abc, nombre = Fulanito, apellido=De tal, email=[fulanito1@gmail.com](mailto:fulanito1@gmail.com)}

Instrucción SQL:

**INSERT INTO Deudores (cc, clave, nombre, apellido, email)**

**VALUES ('123', 'Abc', 'Fulanito', 'De tal', 'fulanito1@gmail.com');**

Captura de pantalla:

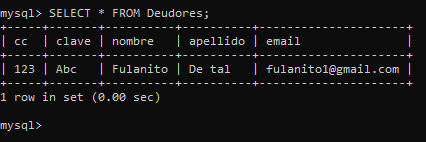


* 1. Mostrar todos los Deudores insertados en la BD

Instrucción SQL:

**SELECT \* FROM Deudores;**

Captura de pantalla:



* 1. Insertar al menos un Crédito:

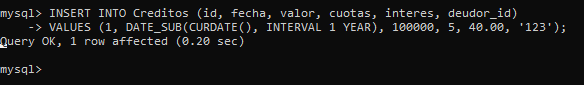
id=1, fecha=hace un año, valor=100000, cuotas=5, interes\_mes=40%, deudor\_id=123

Instrucción SQL:

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

**VALUES (1, DATE\_SUB(CURDATE(), INTERVAL 1 YEAR), 100000, 5, 40.00, '123');**

Captura de pantalla:

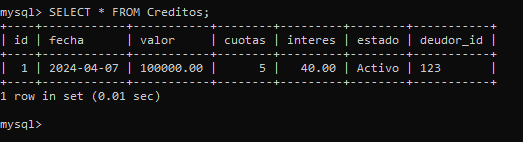


* 1. Mostrar todos los Créditos insertados en la BD

Instrucción SQL:

**SELECT \* FROM Creditos;**

Captura de pantalla:

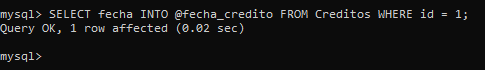


* 1. Insertar 5 pagos así (no modifique ningún valor del ejemplo, colocar las fechas correspondientes, el formato de fecha correspondiente para bd):

Instrucción SQL:

**SELECT fecha INTO @fecha\_credito FROM Creditos WHERE id = 1;**

Captura de pantalla:

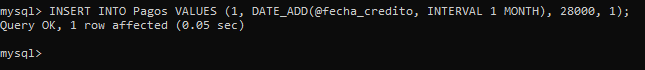


* + 1. id=1, fecha=**un mes después del crédito**, valor=28000, credito\_id=1

Instrucción SQL:

**INSERT INTO Pagos VALUES (1, DATE\_ADD(@fecha\_credito, INTERVAL 1 MONTH), 28000, 1);**

Captura de pantalla:

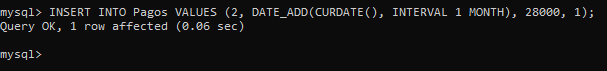


* + 1. id=2, fecha=**un mes después de la fecha actual,** valor=28000, credito\_id=1

Instrucción SQL:

**INSERT INTO Pagos VALUES (2, DATE\_ADD(CURDATE(), INTERVAL 1 MONTH), 28000, 1);**

Captura de pantalla:



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

Instrucción SQL:

**INSERT INTO Pagos VALUES (3, DATE\_ADD(@fecha\_credito, INTERVAL 2 MONTH), 28000, 1);**

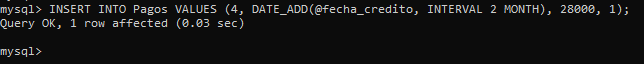
Captura de pantalla:



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

Instrucción SQL:  
**INSERT INTO Pagos VALUES (4, DATE\_ADD(@fecha\_credito, INTERVAL 2 MONTH), 28000, 1);**

Captura de pantalla:

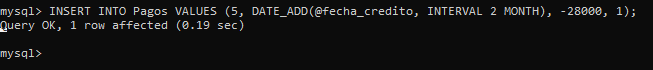


* + 1. id=5, fecha=**un mes después del primer pago,** valor=**-**28000, credito\_id=1

Instrucción SQL:

**INSERT INTO Pagos VALUES (5, DATE\_ADD(@fecha\_credito, INTERVAL 2 MONTH), -28000, 1);**

Captura de pantalla:



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

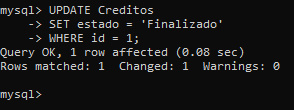
Instrucción SQL:

**UPDATE Creditos**

**SET estado = 'Finalizado'**

**WHERE id = 1;**

Captura de pantalla:



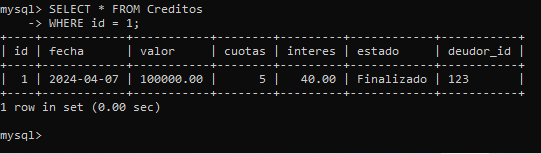
* 1. Mostrar los datos del Crédito con id=1

Instrucción SQL:

**SELECT \* FROM Creditos**

**WHERE id = 1;**

Captura de pantalla:



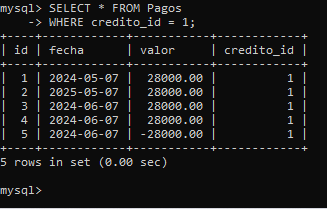
* 1. Mostrar todos los Pagos insertados al Crédito con ID=1 insertados en la BD

Instrucción SQL:

**SELECT \* FROM Pagos**

**WHERE credito\_id = 1;**

Captura de pantalla:



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

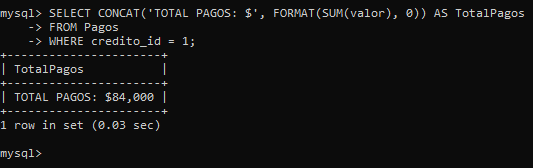
Instrucción SQL:

**SELECT CONCAT('TOTAL PAGOS: $', FORMAT(SUM(valor), 0)) AS TotalPagos**

**FROM Pagos**

**WHERE credito\_id = 1;**

Captura de pantalla:



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

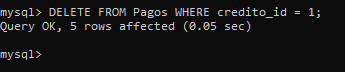
Explicación: El pago 2 no se agregó porque tení͏a una fecha futura, lo que incumple l͏a regla CHECK fecha <= CURDATE() el pago 5 falló porque el valor era negativo, y eso rompe el CHECK valor > ͏0 el estado del pago͏ se cambió bien usando un UPDATE.

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

Instrucción SQL:

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

Captura de pantalla:

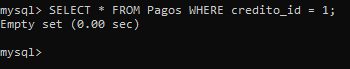


* 1. Mostrar todos los pagos que existen para el crédito con id=1

Instrucción SQL:

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

Captura de pantalla:

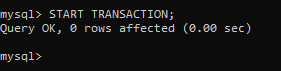


* 1. inicie una TRANSACCION

Instrucción SQL:

**START TRANSACTION;**

Captura de pantalla:

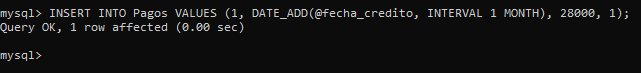


* 1. dentro de la transacción volver a insertar los mismo pagos anteriores:
     1. id=1, fecha=**un mes después del crédito**, valor=28000, credito\_id=1

Instrucción SQL:

**INSERT INTO Pagos VALUES (1, DATE\_ADD(@fecha\_credito, INTERVAL 1 MONTH), 28000, 1);**

Captura de pantalla:

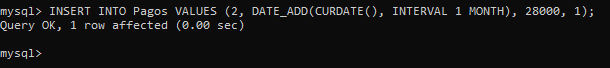


* + 1. id=2, fecha=**un mes después de la fecha actual,** valor=28000, credito\_id=1

Instrucción SQL:

**INSERT INTO Pagos VALUES (2, DATE\_ADD(CURDATE(), INTERVAL 1 MONTH), 28000, 1);**

Captura de pantalla:

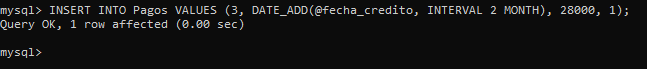


* + 1. id=3, fecha=**un mes después del primer pago,** valor=28000, credito\_id=1

Instrucción SQL:

**INSERT INTO Pagos VALUES (3, DATE\_ADD(@fecha\_credito, INTERVAL 2 MONTH), 28000, 1);**

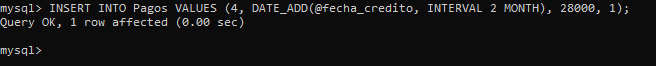
Captura de pantalla:



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

Instrucción SQL:  
**INSERT INTO Pagos VALUES (4, DATE\_ADD(@fecha\_credito, INTERVAL 2 MONTH), 28000, 1);**

Captura de pantalla:

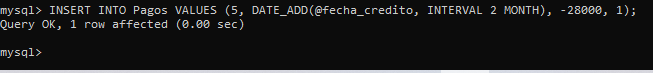


* + 1. id=5, fecha=**un mes después del primer pago,** valor=**-**28000, credito\_id=1

Instrucción SQL:

**INSERT INTO Pagos VALUES (5, DATE\_ADD(@fecha\_credito, INTERVAL 2 MONTH), -28000, 1);**

Captura de pantalla:

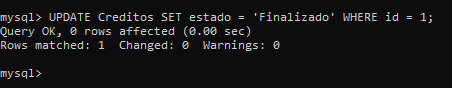


* 1. Cambiar o actualizar el estado del Crédito con id=1, para que ahora sea Finalizado

Instrucción SQL:

**UPDATE Creditos SET estado = 'Finalizado' WHERE id = 1;**

Captura de pantalla:

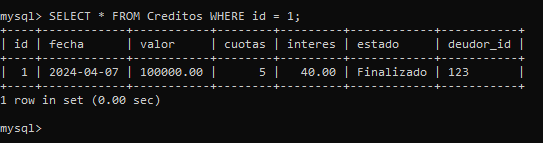


* 1. Mostrar los datos del Crédito con id=1

Instrucción SQL:

**SELECT \* FROM Creditos WHERE id = 1;**

Captura de pantalla:

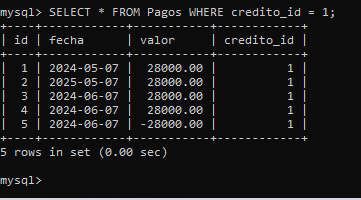


* 1. Mostrar todos los Pagos insertados al Crédito con ID=1 insertados en la BD

Instrucción SQL:

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

Captura de pantalla:



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

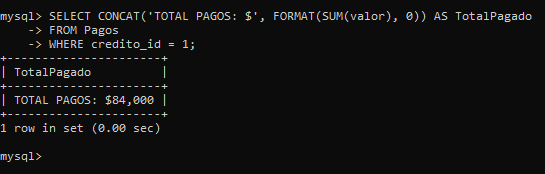
Instrucción SQL:

**SELECT CONCAT('TOTAL PAGOS: $', FORMAT(SUM(valor), 0)) AS TotalPagado**

**FROM Pagos**

**WHERE credito\_id = 1;**

Captura de pantalla:

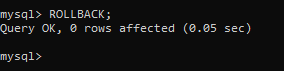


* 1. Cancelar la transaccion (rollblack)

Instrucción SQL:

**ROLLBACK;**

Captura de pantalla:

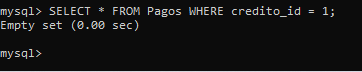


* 1. Mostrar todos los Pagos insertados al Crédito con ID=1 insertados en la BD

Instrucción SQL:

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

Captura de pantalla:



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

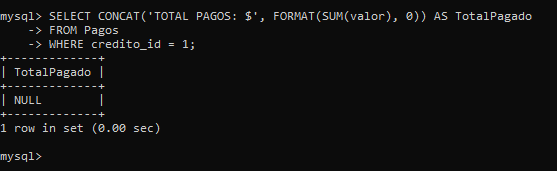
Instrucción SQL:

**SELECT CONCAT('TOTAL PAGOS: $', FORMAT(SUM(valor), 0)) AS TotalPagado**

**FROM Pagos**

**WHERE credito\_id = 1;**

Captura de pantalla:



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

Explicacion: Un ROLLBACK hace que un cambio o inserción incorrecta se cancelen. Esto también se detiene͏ cualq͏ui͏er ͏otro cambi͏o que͏ se estuviera haciendo también se ve el poder del CHECK no dejan poner datos incorrectos, como pagos malos o fechas venideras.