**Bases de datos II**

**Actividad 2**

**PostgreSQL**

Jonny Luna Guerrero

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](#_Toc138539610)

[2 Introducción 5](#_Toc138539611)

[3 Objetivos 6](#_Toc138539612)

[3.1 Objetivo General: 6](#_Toc138539613)

[3.2 Objetivos específicos 6](#_Toc138539614)

[4 Justificación 7](#_Toc138539615)

[5 Desarrollo 8](#_Toc138539616)

[5.1 Qué es y qué problema resuelve el uso del concepto de Vistas SQL en bases de datos Relacionales o SQL 8](#_Toc138539617)

[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). 8](#_Toc138539618)

[5.3 Usar la consulta anterior y crear con ella una Vista SQL. 9](#_Toc138539619)

[5.4 Mostrar la Vista SQL creada anteriormente 10](#_Toc138539620)

[5.5 Ejecutar la Vista SQL creada anteriormente 10](#_Toc138539621)

[5.6 Abrir la Vista SQL y modificarla para cambiar la consulta por otra consulta o modificarla existente. 10](#_Toc138539622)

[5.7 Volver a ejecutar la Vista SQL 11](#_Toc138539623)

[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](#_Toc138539624)

[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](#_Toc138539625)

[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](#_Toc138539626)

[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](#_Toc138539627)

[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](#_Toc138539628)

[5.9.1 Crear una base de datos llamada prueba\_transacciones\_grupoXYZ, siendo XYZ el nombre del ejercicio escogido como equipo. 13](#_Toc138539629)

[5.9.2 Crear tablas **Deudores**(**cc**, clave, nombre, apellido, email). email es único, todos son obligatorios 14](#_Toc138539630)

[5.9.3 Crear tablas **Creditos**(id, fecha, valor, cuotas, interes, estado, deudor\_id), todos son obligatorios, estado por defecto es Activo 14](#_Toc138539631)

[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. 15](#_Toc138539632)

[5.9.5 Insertar al menos un Deudor: 16](#_Toc138539633)

[5.9.6 Mostrar todos los Deudores insertados en la BD 17](#_Toc138539634)

[5.9.7 Insertar al menos un Crédito: 17](#_Toc138539635)

[5.9.8 Mostrar todos los Créditos insertados en la BD 18](#_Toc138539636)

[5.9.9 Insertar 5 pagos así (no modifique ningún valor del ejemplo, colocar las fechas correspondientes, el formato de fecha correspondiente para bd): 18](#_Toc138539637)

[5.9.10 Cambiar o actualizar el estado del Crédito con id=1, para que ahora sea Finalizado. 22](#_Toc138539638)

[5.9.11 Mostrar los datos del Crédito con id=1 27](#_Toc138539639)

[5.9.12 Mostrar todos los Pagos insertados al Crédito con ID=1 insertados en la BD 27](#_Toc138539640)

[5.9.13 Totalizar el total de valores pagados para el crédito con id=1, para que aparezca así: TOTAL PAGOS: $140000 28](#_Toc138539641)

[5.9.14 Analice el resultado de cada una de las operaciones anteriores, comente y explique el porqué de cada uno. 29](#_Toc138539642)

[5.9.15 Elimine todos los pagos de crédito con ID: 1 29](#_Toc138539643)

[5.9.16 Mostrar todos los pagos que existen para el crédito con id=1 29](#_Toc138539644)

[5.9.17 inicie una TRANSACCION 30](#_Toc138539645)

[5.9.18 dentro de la transacción volver a insertar los mismo pagos anteriores: 32](#_Toc138539646)

[5.9.19 Cambiar o actualizar el estado del Crédito con id=1, para que ahora sea Finalizado 34](#_Toc138539647)

[5.9.20 Mostrar los datos del Crédito con id=1 35](#_Toc138539648)

[5.9.21 Mostrar todos los Pagos insertados al Crédito con ID=1 insertados en la BD 35](#_Toc138539649)

[5.9.22 Totalizar el total de valores pagados para el crédito con id=1, para que aparezca así: TOTAL PAGOS: $140000 36](#_Toc138539650)

[5.9.23 Cancelar la transaccion (rollblack) 36](#_Toc138539651)

[5.9.24 Mostrar todos los Pagos insertados al Crédito con ID=1 insertados en la BD 36](#_Toc138539652)

[5.9.25 Totalizar el total de valores pagados para el crédito con id=1, para que aparezca así: TOTAL PAGOS: $140000 37](#_Toc138539653)

[5.9.26 Analice el resultado de cada una de las operaciones anteriores, comente y explique el por que de cada uno. 37](#_Toc138539654)

[5.10 Diligenciar la siguiente tabla con el mismo ejemplo por línea para los motores de bd: 38](#_Toc138539655)

[5.10.1 Ejemplo de subconsulta en el Select. 38](#_Toc138539656)

[5.10.2 Subconsulta en el From 39](#_Toc138539657)

[5.10.3 Subconsulta en el Where 40](#_Toc138539658)

[5.10.4 Subconsulta en el Having 41](#_Toc138539659)

[5.10.5 Uso de Check 42](#_Toc138539660)

[5.10.6 Join Inner 43](#_Toc138539661)

[5.10.7 Join Full 43](#_Toc138539662)

[5.10.8 Join Outer 44](#_Toc138539663)

[5.10.9 Join Lef 44](#_Toc138539664)

[5.10.10 Join Rigth 45](#_Toc138539665)

[5.10.11 Unión 45](#_Toc138539666)

[5.10.12 Intersect 46](#_Toc138539667)

[5.10.13 Crear una Vista SQL 47](#_Toc138539668)

[5.10.14 Usar una Vista SQL 47](#_Toc138539669)

[5.10.15 Editar una Vista SQL 48](#_Toc138539670)

[5.10.16 Mostrar el contenido de una Vista SQL 48](#_Toc138539671)

[5.10.17 Mostrar las Vistas SQL 49](#_Toc138539672)

[5.10.18 Eliminar una Vista SQL 49](#_Toc138539673)

[5.10.19 Crear una Transacción SQL 50](#_Toc138539674)

[5.10.20 Uso de Try para detectar cuando falla una transacción SQL 51](#_Toc138539675)

[6 Bibliografía 55](#_Toc138539676)

# Introducción

Las vistas en SQL son muy importantes debido a que ayudan a mejorar el rendimiento de las bases de datos, puesto que facilitan el uso de consultas frecuentes, además hacen que las mismas sean mas sencillas al poder se cambiar las consultas complejas por unas o varias mas sencillas.

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 Vistas, Transacciones, procedimientos almacenados, los diferentes tipos de subconsultas, el uso de los Joins en las bases de datos relacionales.

.

## Objetivos específicos

* Utilizar las vistas en PosgreSQL para que se facilite el uso de las subconsultas y que las bases de datos sean mas eficientes.
* Utilizar correctamente las transacciones en las operaciones donde se requieran utilizando PostgreSQL.
* Aplicar los procedimientos almacenados para que se faciliten las operqciones en las bases de datos con PostgreSQL.

# Justificación

Utilizar las vistas en PostgreSQL es de vital importancia para facilitar el proceso de consultas debido a que se pueden reutilizar las mismas cuando estas sean requeridas con cierta frecuencia, haciendo que la base de datos sea mas eficiente.

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 PostgreSQL es muy útil utilizar las vistas, las transacciones y los procedimientos almacenados en la búsqueda de mejorar el rendimiento y eficiencia de la base de datos.

# 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).

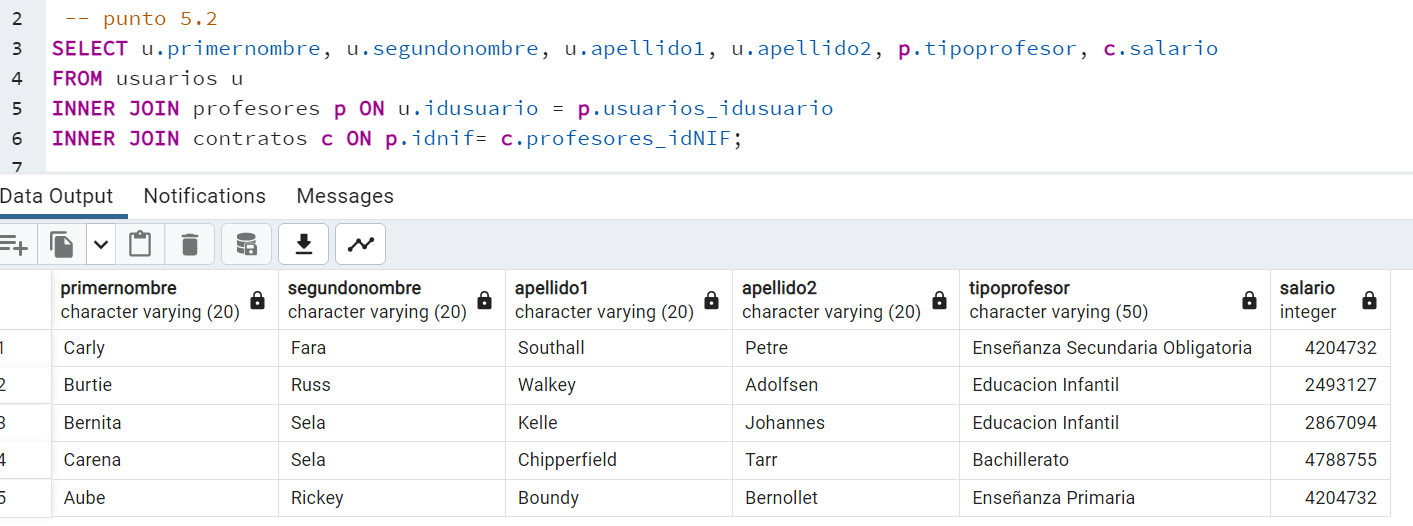
Sentencia SQL

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

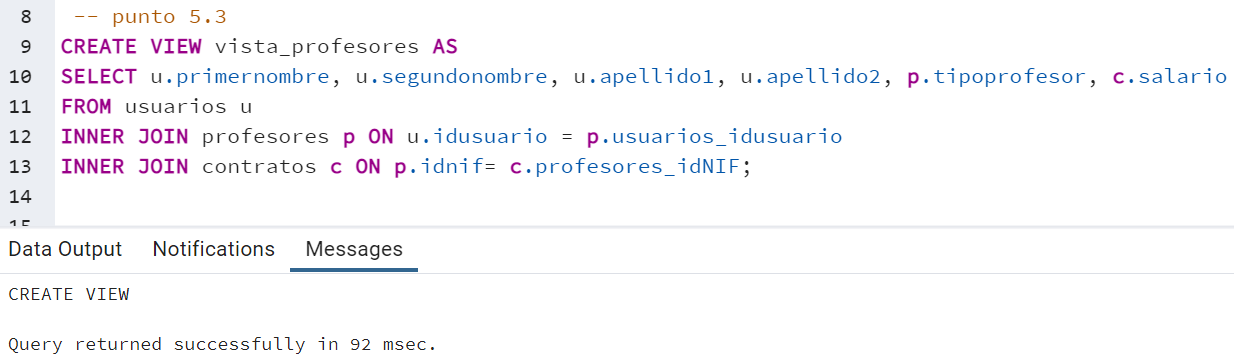
CREATE VIEW vista\_profesores AS

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;



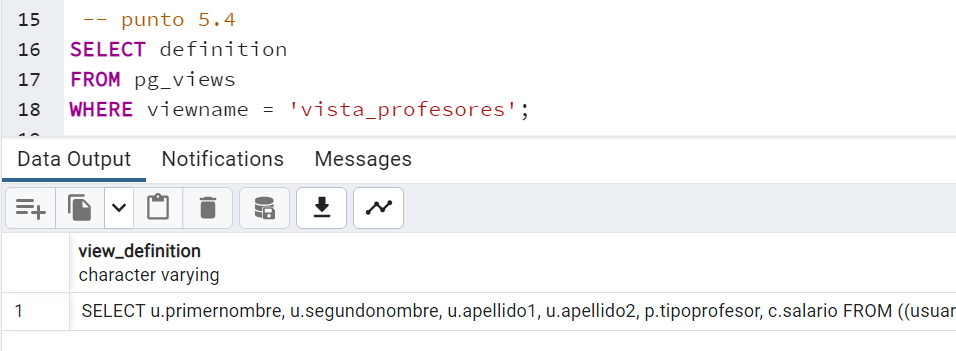
## Mostrar la Vista SQL creada anteriormente

Sentencia SQL

SELECT definition

FROM pg\_views

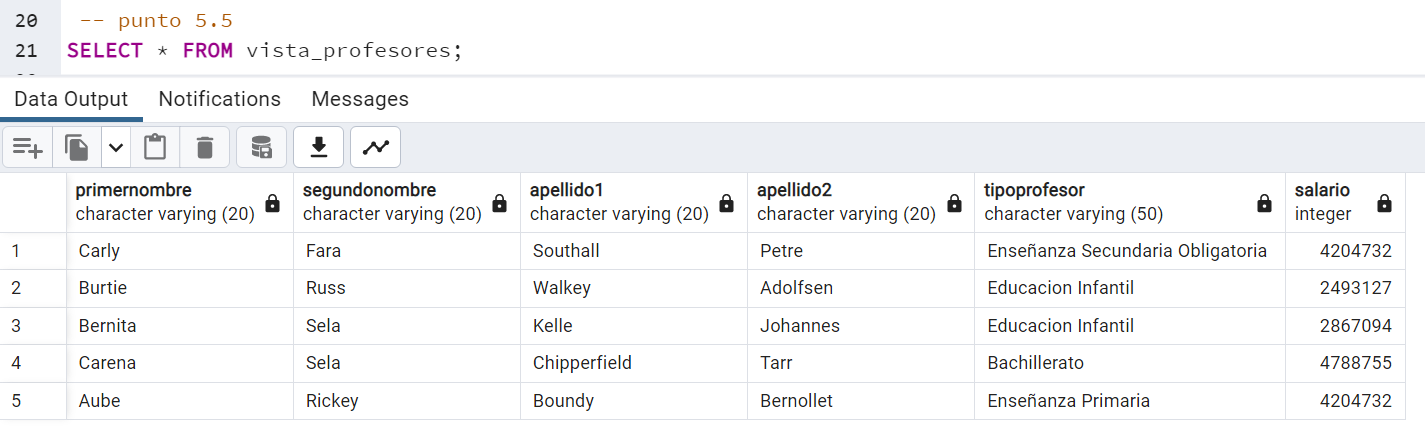
WHERE viewname = 'vista\_profesores';



## 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

CREATE OR REPLACE VIEW vista\_profesores AS

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

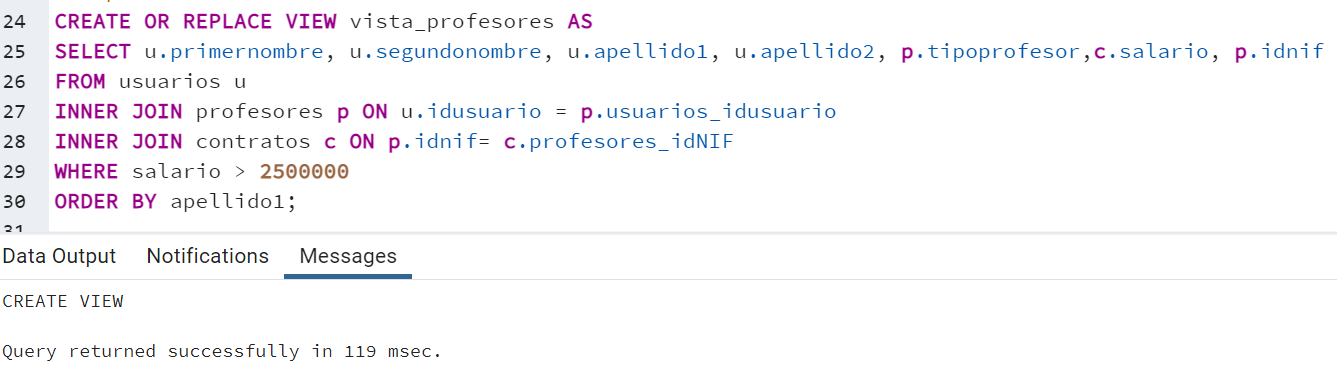
FROM usuarios u

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

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

WHERE salario > 2500000

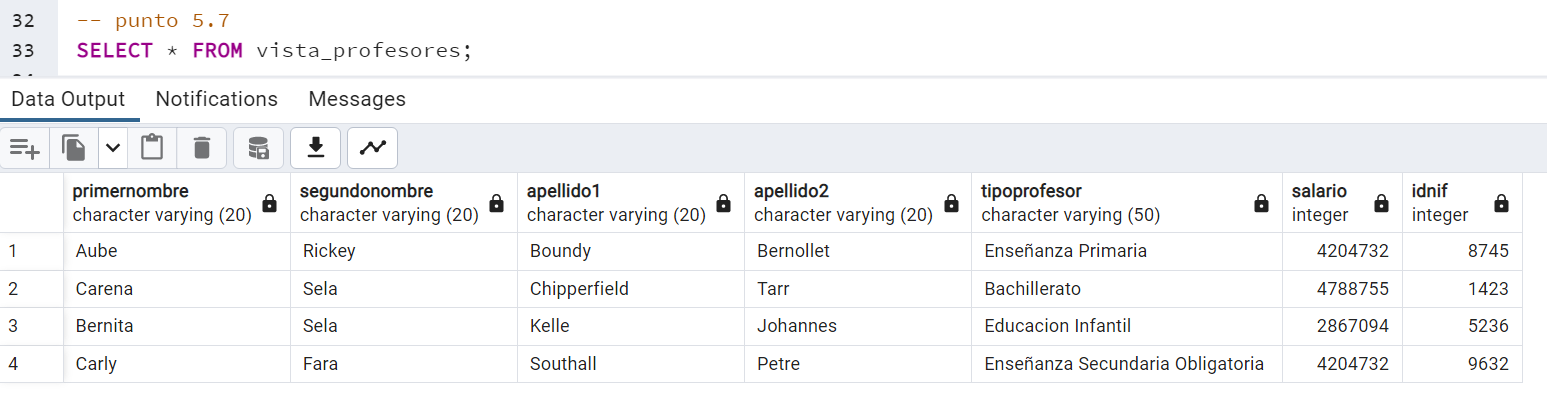
ORDER BY apellido1;



## 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.

Realmente no hay mucha diferencia entre las ventajas y desventajas que ofrece el uso de las vistas en los diferentes motores de bases de datos.

Las ventajas en el uso de vistas en PostgreSQL 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: PostgreSQL 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.

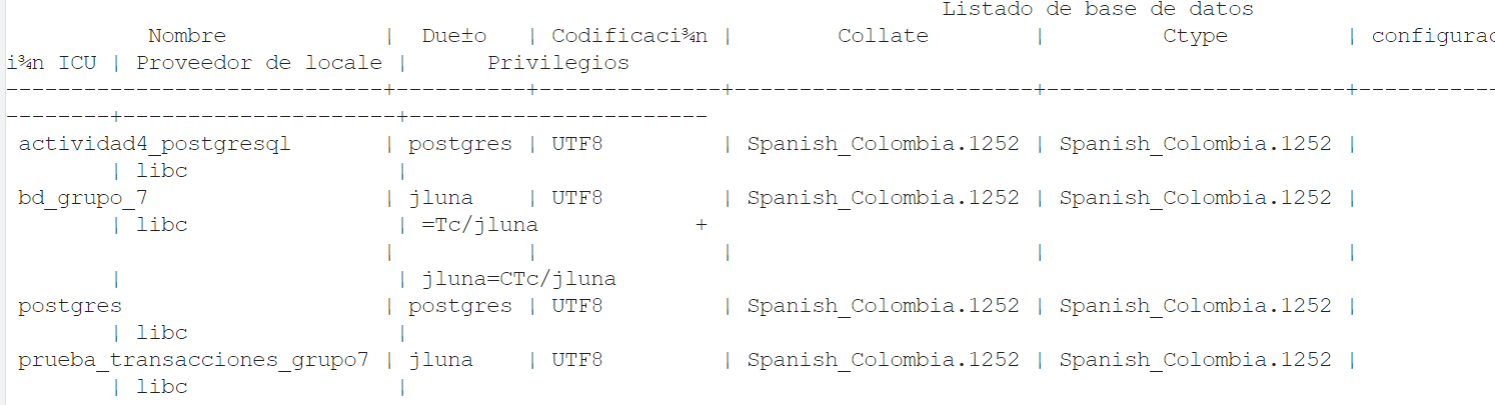
Las Limitaciones más comunes son: No se pueden indexar todas las vistas, restricciones en la modificación de datos y complejidad de mantenimiento.

## 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 prueba\_transacciones;



### 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,

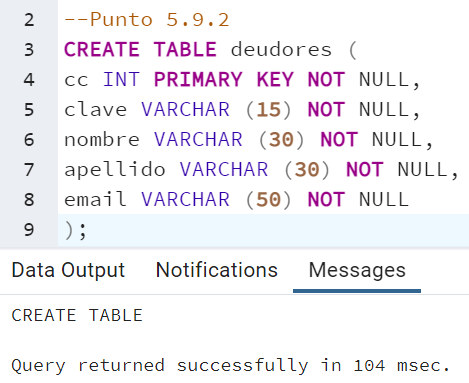
clave VARCHAR (15) NOT NULL,

nombre VARCHAR (30) NOT NULL,

apellido VARCHAR (30) NOT NULL,

email VARCHAR (50) NOT NULL

);



### 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 SERIAL 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,

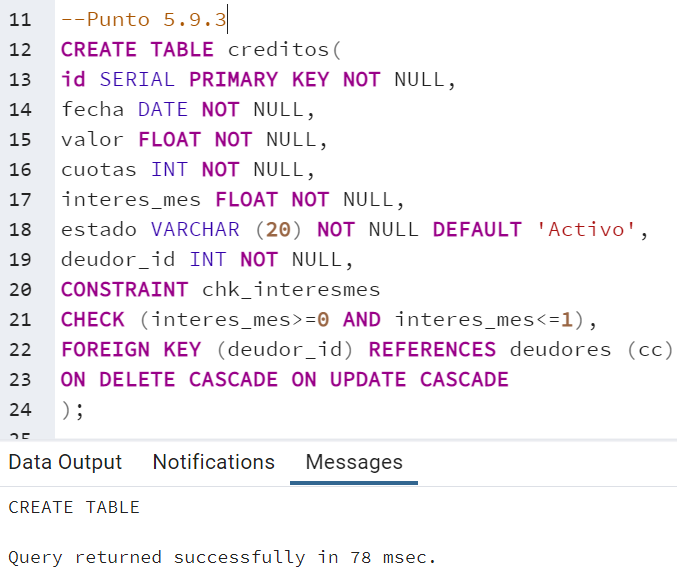
CONSTRAINT chk\_interesmes

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

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

ON DELETE CASCADE ON UPDATE CASCADE

);



### 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 SERIAL PRIMARY KEY NOT NULL,

fecha DATE NOT NULL,

valor FLOAT NOT NULL ,

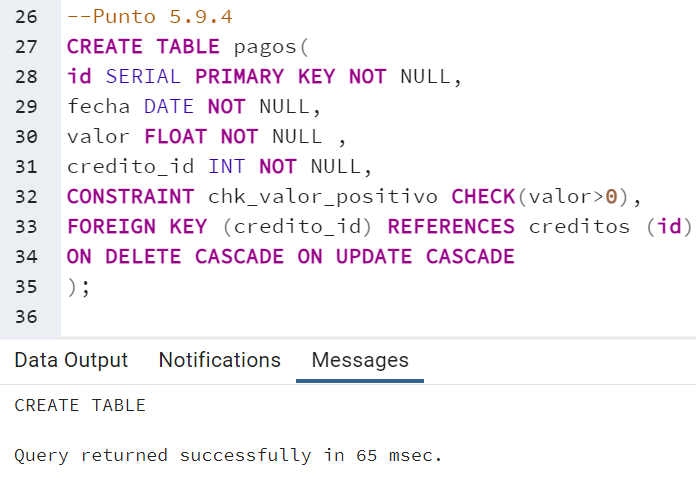
credito\_id INT NOT NULL,

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

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

ON DELETE CASCADE ON UPDATE CASCADE

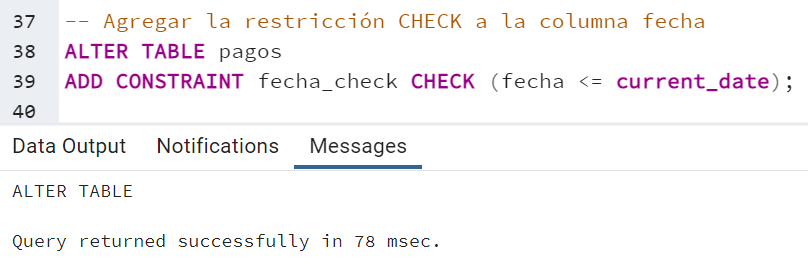
);



Se agregó la restricción CHECK a la columna fecha con la siguiente sentencia SQL

ALTER TABLE pagos

ADD CONSTRAINT fecha\_check CHECK (fecha <= current\_date);



### 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'),

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

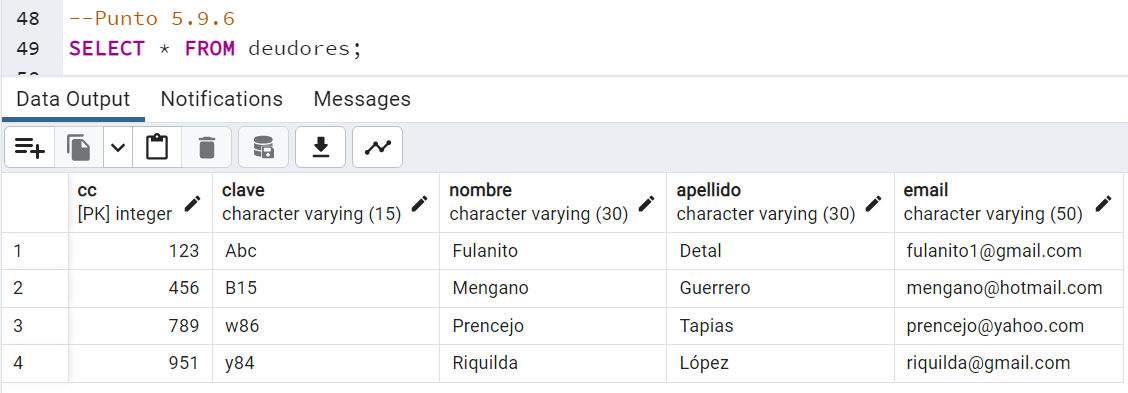
(789, 'w86', 'Prencejo', 'Tapias', 'prencejo@yahoo.com'),

(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

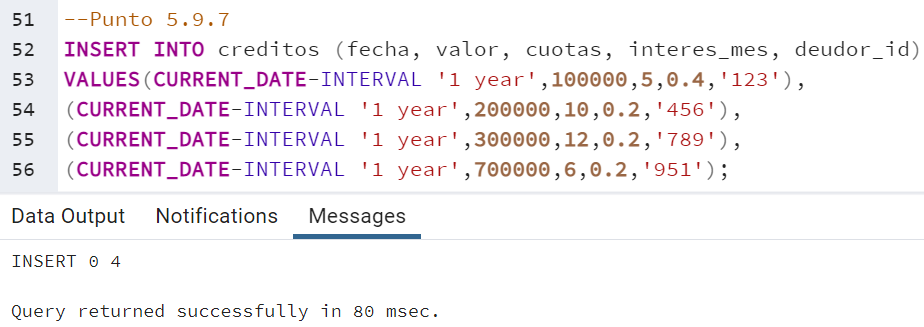
INSERT INTO creditos (fecha, valor, cuotas, interes\_mes, deudor\_id)

VALUES(CURRENT\_DATE-INTERVAL '1 year',100000,5,0.4,'123'),

(CURRENT\_DATE-INTERVAL '1 year',200000,10,0.2,'456'),

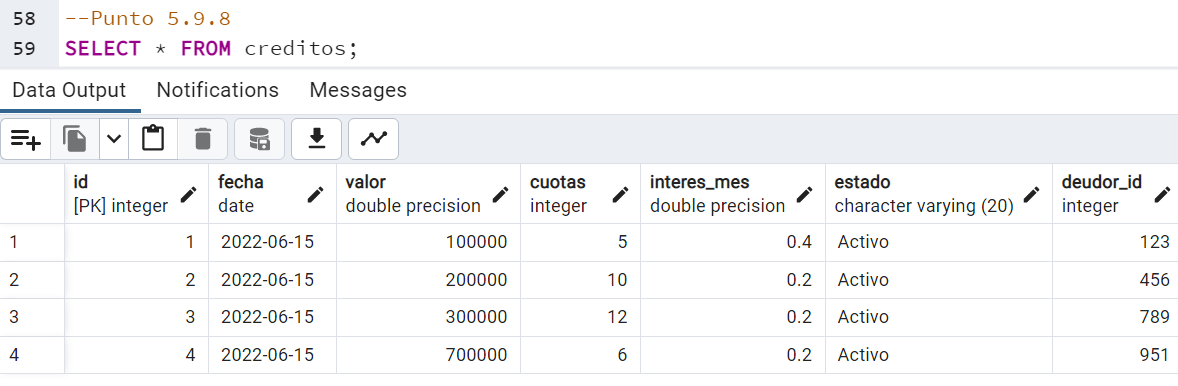
(CURRENT\_DATE-INTERVAL '1 year',300000,12,0.2,'789'),

(CURRENT\_DATE-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

CREATE OR REPLACE PROCEDURE InsertarPago

(pago\_id INTEGER,

pago\_fecha DATE,

pago\_valor FLOAT,

credito\_id INTEGER)

LANGUAGE plpgsql

AS $$

BEGIN

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

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

IF FOUND THEN

RAISE NOTICE 'El pago con Id %

del credito con Id %

ha sistematizado Exitosamente! ',pago\_id, credito\_id ;

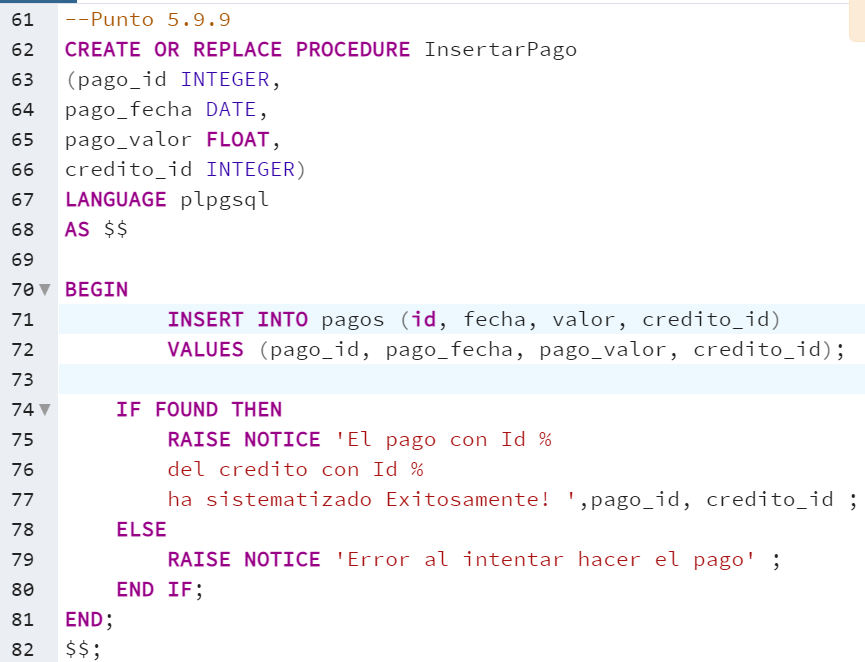
ELSE

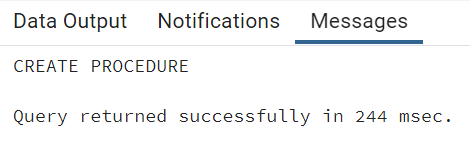
RAISE NOTICE 'Error al intentar hacer el pago' ;

END IF;

END;

$$;



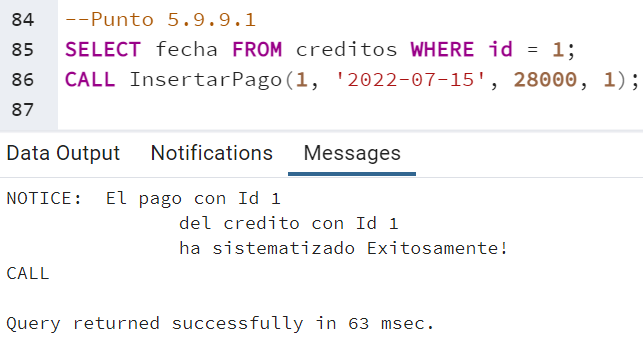


#### id=1, fecha=**un mes después del crédito**, valor=28000, credito\_id=1

Sentencia SQL

SELECT fecha FROM creditos WHERE id = 1;

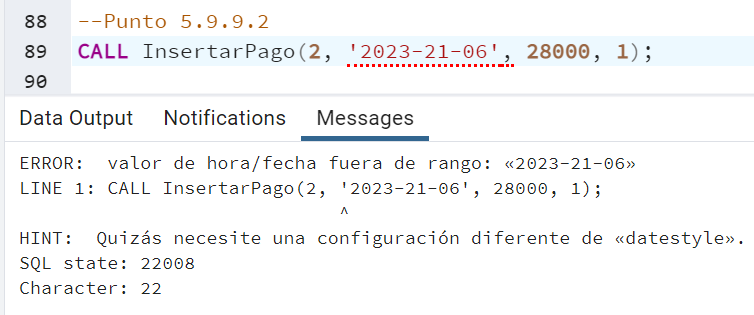
CALL InsertarPago(1, '2022-07-15', 28000, 1);



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

Sentencia SQL

CALL InsertarPago(2, '2023-21-06', 28000, 1);

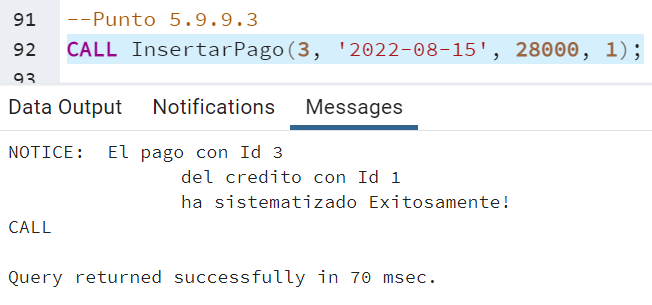


Se observa que al intentar introducir un pago con fecha mayor a la fecha actual da un error esto nos indica que las restricción colocada anteriormente es correcta.

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

Sentencia SQL

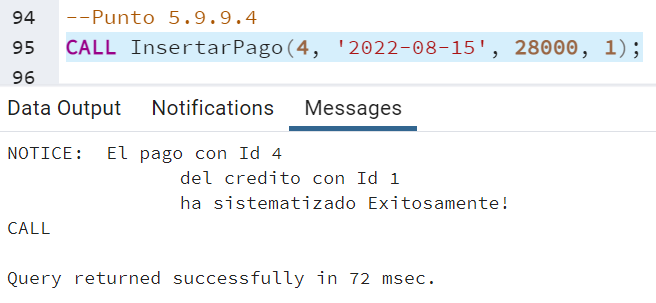
CALL InsertarPago(3, '2022-08-15', 28000, 1);



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

Sentencia SQL

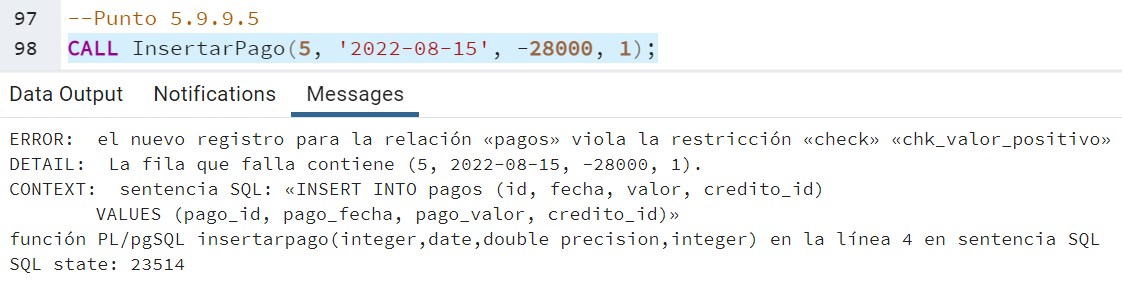
CALL InsertarPago(4, '2022-08-15', 28000, 1);



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

Sentencia SQL

CALL InsertarPago(5, '2022-08-15', -28000, 1);



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

### 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, se realiza 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.

Sentencia SQL

CREATE OR REPLACE PROCEDURE EstadoCredito(IN c\_credito\_id INTEGER)

LANGUAGE plpgsql

AS $$

DECLARE

valor\_credito FLOAT;

tasa\_interes FLOAT;

valor\_deuda FLOAT;

valor\_saldo FLOAT;

valor\_pagado FLOAT;

estado\_actual VARCHAR(20);

estado\_final VARCHAR(20);

BEGIN

-- 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 = c\_credito\_id;

-- Calculamos el valor de la deuda

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

-- Calculamos el valor total pagado

SELECT SUM(valor) INTO valor\_pagado

FROM pagos

WHERE pagos.credito\_id = c\_credito\_id;

-- calculamos el saldo

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

-- Determinar el estado del crédito

IF valor\_pagado >= valor\_deuda THEN

estado\_final := 'Finalizado';

ELSE

estado\_final := estado\_actual;

END IF;

-- Actualizar el estado del crédito en la tabla Creditos

UPDATE creditos

SET estado = estado\_final

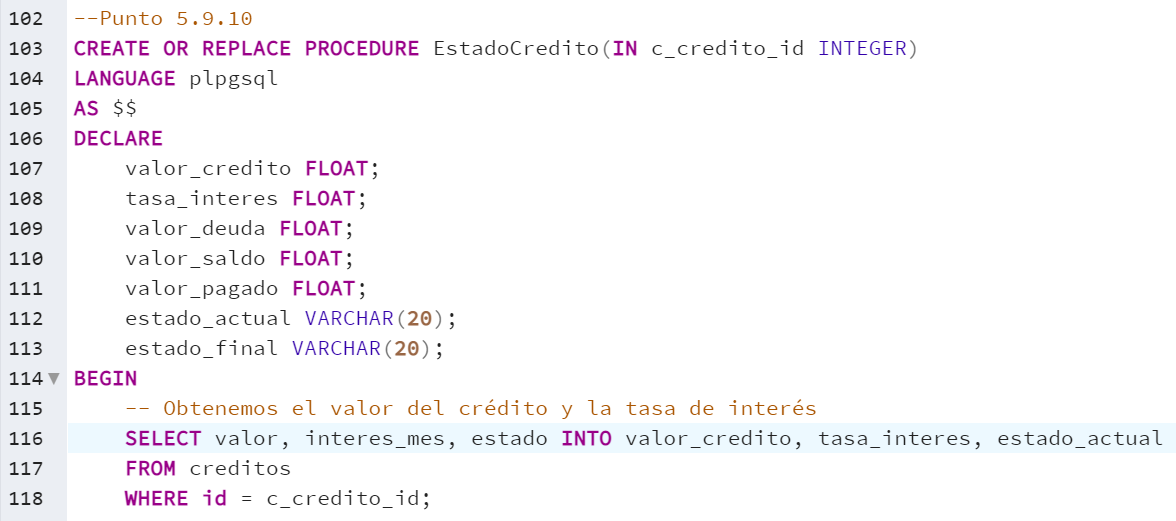
WHERE id = c\_credito\_id;

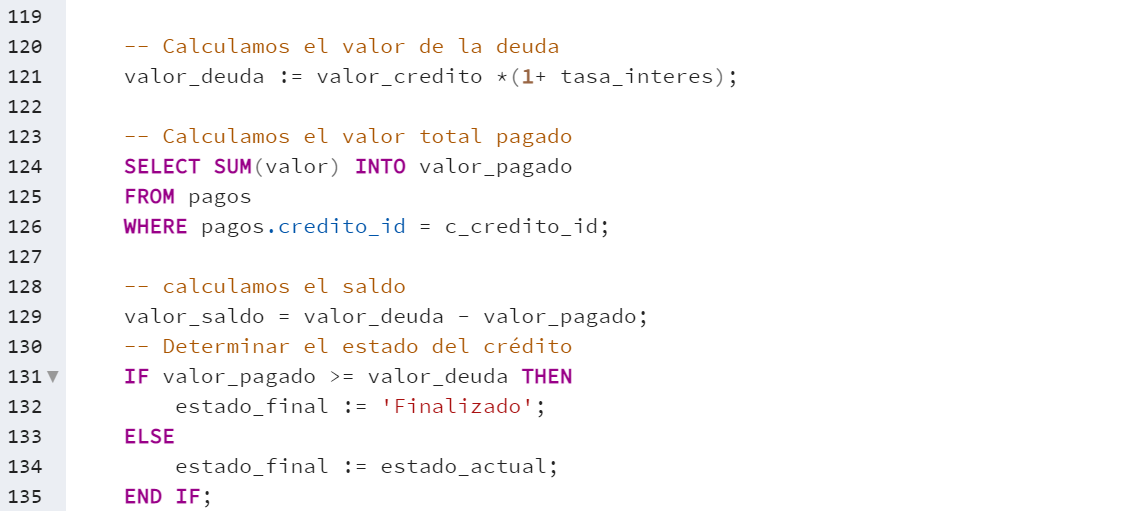
-- enviamos un mensaje con el estado final del credito

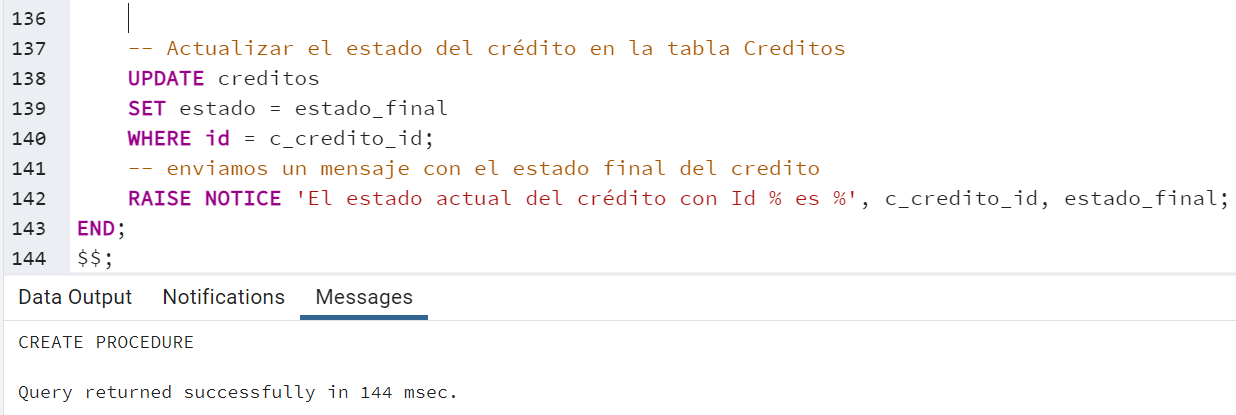
RAISE NOTICE 'El estado actual del crédito con Id % es %', c\_credito\_id, estado\_final;

END;

$$;

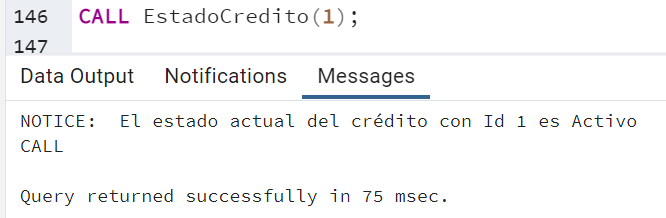






Ahora llamamos el procedimiento EstadoCredito() con id = 1

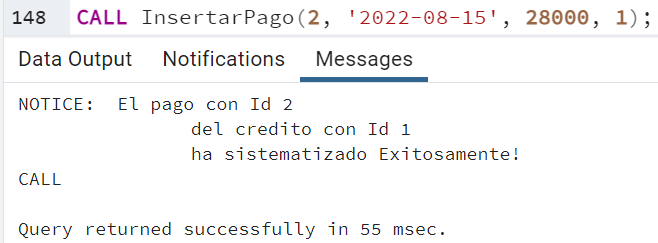
CALL EstadoCredito(1);



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

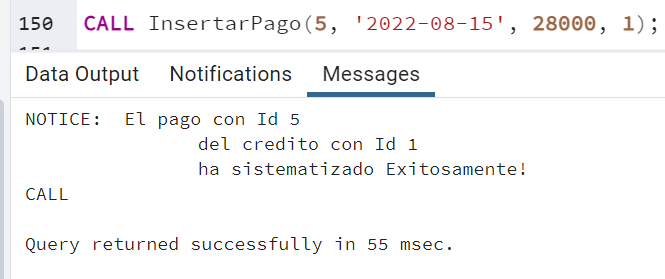
Se Corrige el pago con el id 2 el cual se hacía con un pago con fecha posterior a la fecha actual

CALL InsertarPago(2, '2022-08-15', 28000, 1);



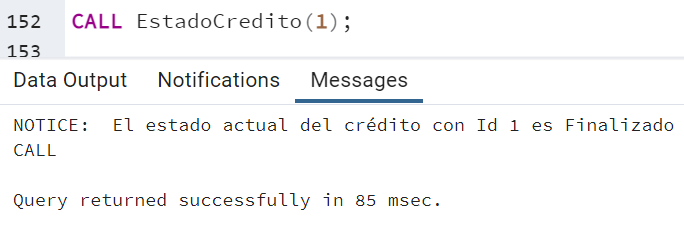
Se corrige el pago con el id 5 el cual se hacía con un pago de valor negativo.

CALL InsertarPago(5, '2022-08-15', 28000, 1);



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

CALL EstadoCredito(1);

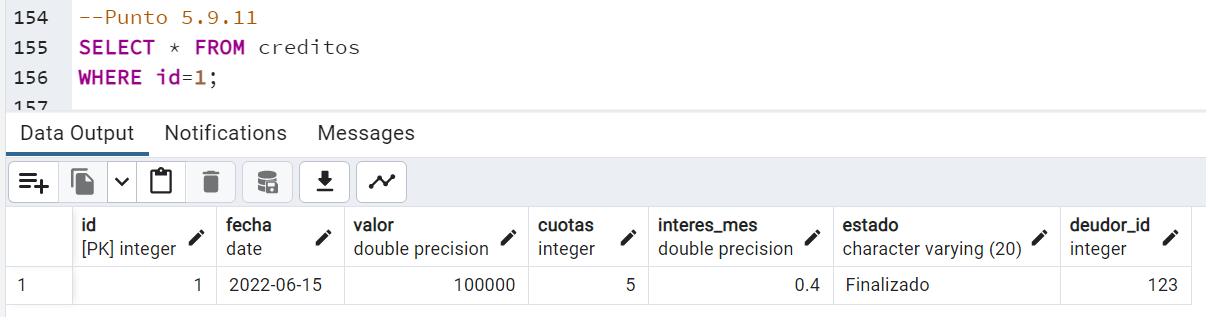


### Mostrar los datos del Crédito con id=1

Sentencia SQL

SELECT \* FROM creditos

WHERE id=1;

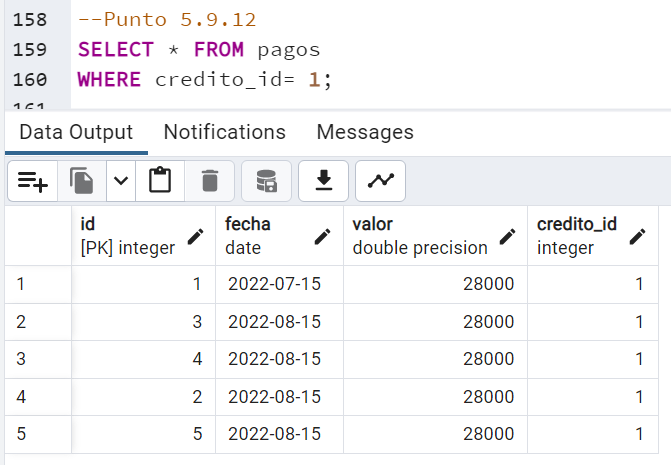


### Mostrar todos los Pagos insertados al Crédito con ID=1 insertados en la BD

Sentencia SQL

SELECT \* FROM pagos

WHERE credito\_id= 1;



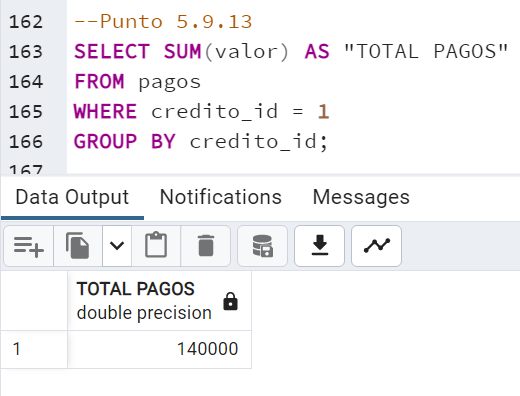
### Totalizar el total de valores pagados para el crédito con id=1, para que aparezca así: TOTAL PAGOS: $140000

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 porqué de cada uno.

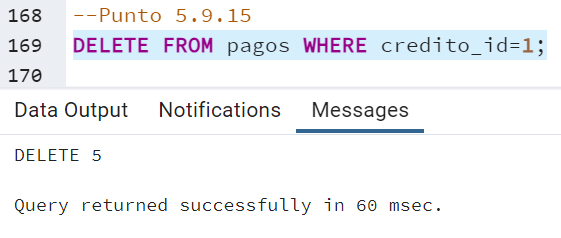
Se observa que en el crédito con id = 1 el cual corresponde a un valor de $ 100.000 con una tasa de interés del 40% mensual y cuotas a pagar 5 por un valor de $28.000 cada una. Al realizar los 5 pagos se observó que el pago con id = 2 no fue posible porque violaba la restricción hecha de que no se puede 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’, razón por la cual se corrigieron los pagos con id = 2 e id = 5 hicieron los dos para obtener el resultado esperado.

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

Sentencia SQL

DELETE FROM pagos WHERE credito\_id=1;

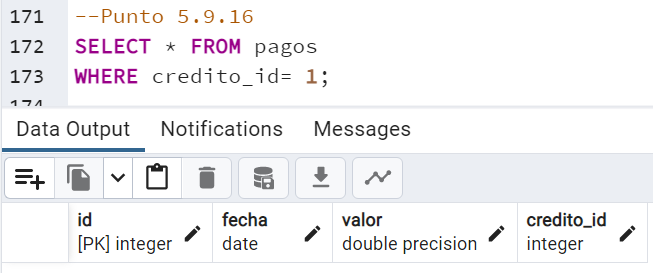


### Mostrar todos los pagos que existen para el crédito con id=1

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

CREATE OR REPLACE PROCEDURE insertar\_pagos(

p\_pago\_id INTEGER,

p\_fecha DATE,

p\_valor FLOAT,

p\_credito\_id INTEGER

)

LANGUAGE plpgsql

AS $$

DECLARE

v\_error\_message TEXT;

BEGIN

--se inicia la transacción

BEGIN

-- Validar que el valor del pago sea positivo

IF p\_valor <= 0 THEN

RAISE EXCEPTION 'El valor del pago debe ser mayor que cero.';

END IF;

-- Validar que la fecha de pago no sea mayor que la fecha actual

IF p\_fecha > CURRENT\_DATE THEN

RAISE EXCEPTION 'La fecha de pago no puede ser mayor que la fecha actual.';

END IF;

BEGIN

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

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

EXCEPTION

WHEN OTHERS THEN

GET STACKED DIAGNOSTICS v\_error\_message = MESSAGE\_TEXT;

RAISE NOTICE 'Error al insertar el pago: %', v\_error\_message;

RETURN;

END;

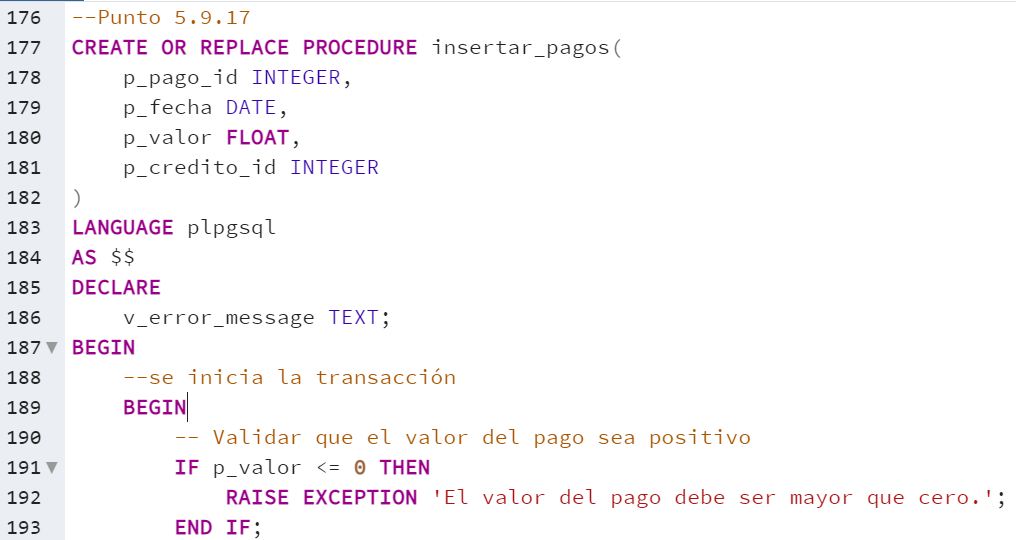
COMMIT;

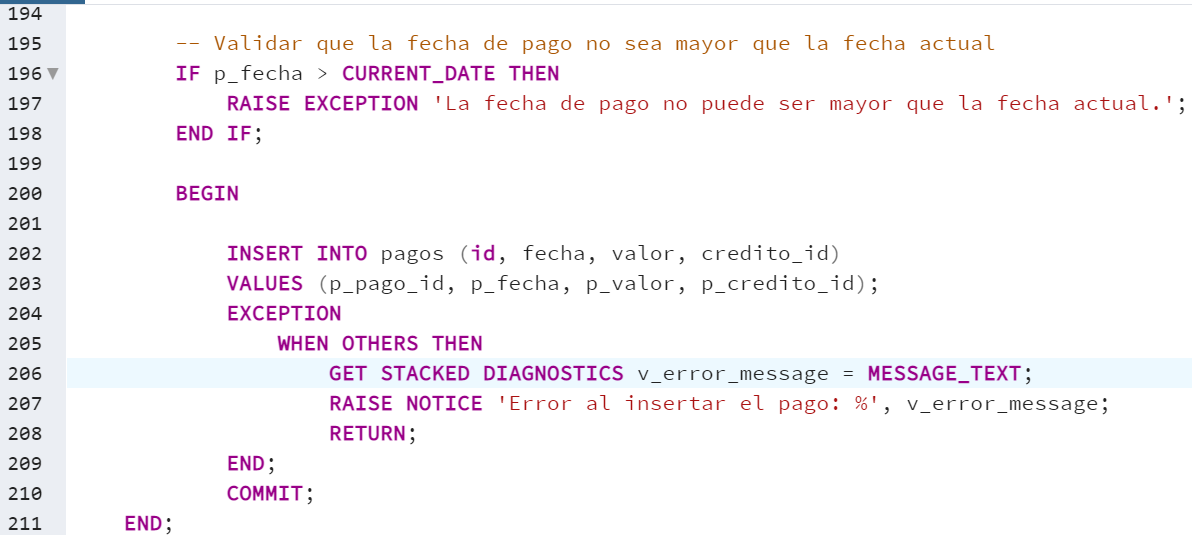
END;

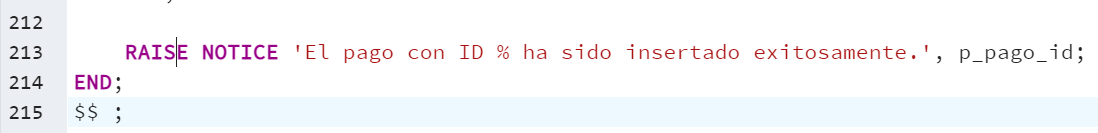
RAISE NOTICE 'El pago con ID % ha sido insertado exitosamente.', p\_pago\_id;

END;

$$ ;





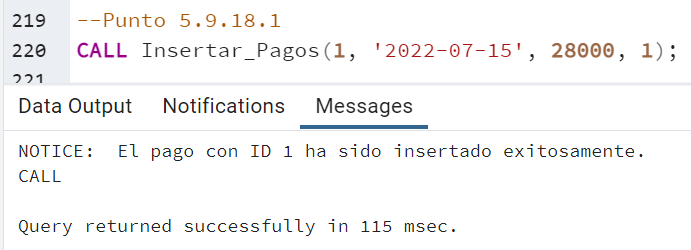


### dentro de la transacción volver a insertar los mismo pagos anteriores:

Insertamos los pagos llamando el procedimiento que contiene la transacción.

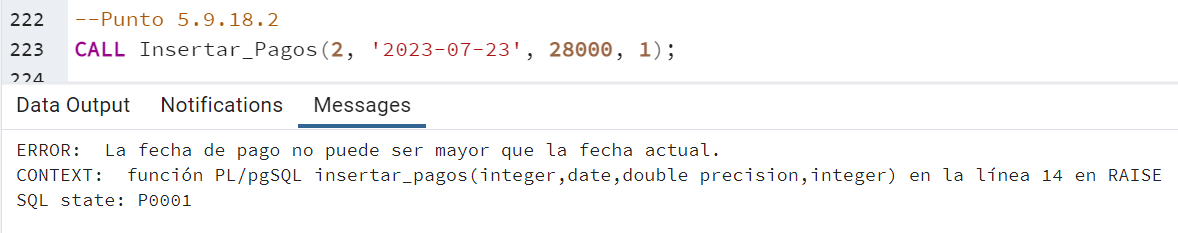
#### id=1, fecha=**un mes después del crédito**, valor=28000, credito\_id=1

CALL Insertar\_Pagos(1, '2022-07-15', 28000, 1);



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

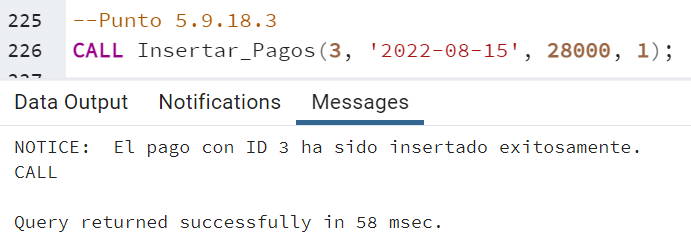
CALL Insertar\_Pagos(2, '2023-07-23', 28000, 1);



Se observa que se produce un error por la restricción de fecha que se había colocado pero este error es capturado por el procedimiento.

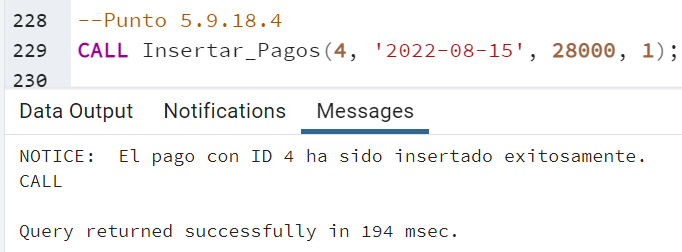
#### id=3, fecha=**un mes después del primer pago,** valor=28000, credito\_id=1

CALL Insertar\_Pagos(3, '2022-08-15', 28000, 1);



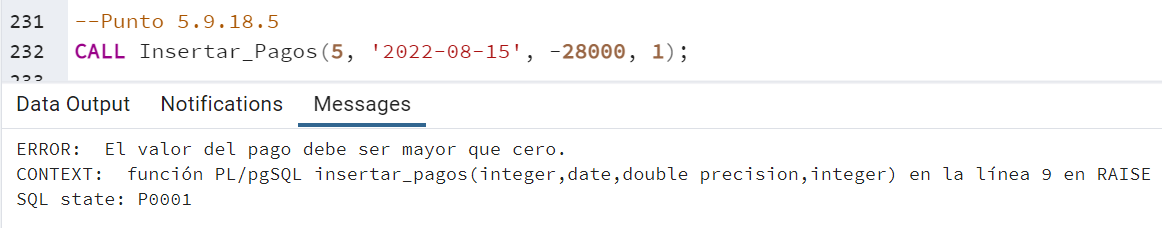
#### id=4, fecha=**un mes después del primer pago,** valor=28000, credito\_id=1

CALL Insertar\_Pagos(4, '2022-08-15', 28000, 1);



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

CALL Insertar\_Pagos(5, '2022-08-15', -28000, 1);

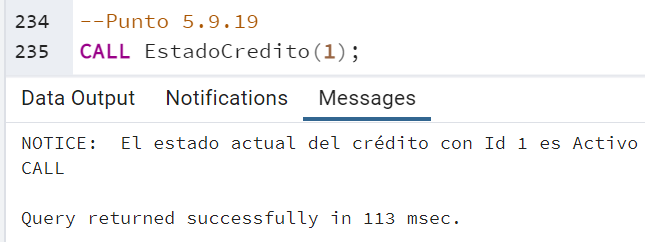


En este punto se observa que se viola la restricción del valor del pago el cual debe ser mayor que cero.

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

En este punto utilizamos el procedimiento realizado en el punto 5.9.10 con el nombre **Estado\_crédito**.

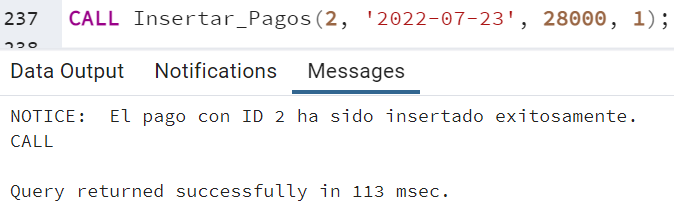
CALL EstadoCredito(1);



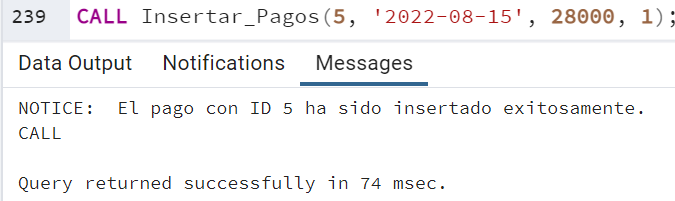
Se observa que el estado del crédito es “Activo” porque solo tres de los 5 pagos se hicieron efectivos.

Se procede a realizar correctamente los pagos 2 y 5. Usando la función del punto 5.9.17 de nombre **insertar\_pagos**.

CALL Insertar\_Pagos(2, '2022-07-23', 28000, 1);

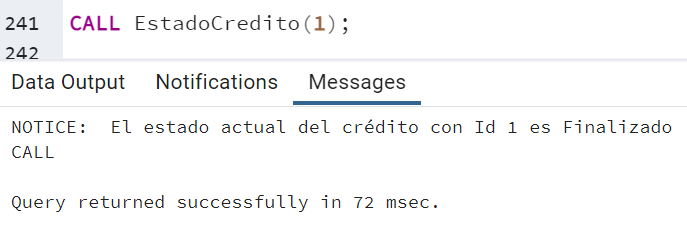


CALL Insertar\_Pagos(5, '2022-08-15', 28000, 1);



Nuevamente usamos el procedimiento almacenado de nombre **Estado\_crédito**.

CALL EstadoCredito(1);

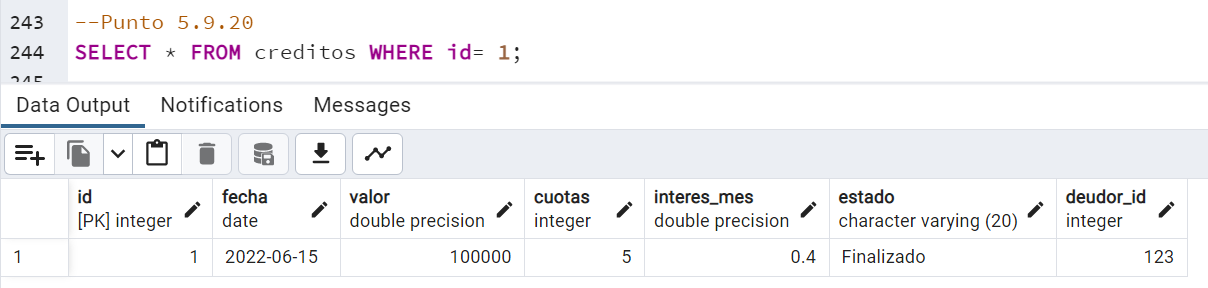


Se observa que ahora el estado del crédito es “Finalizado”.

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

Sentencia SQL

SELECT \* FROM creditos WHERE id= 1;

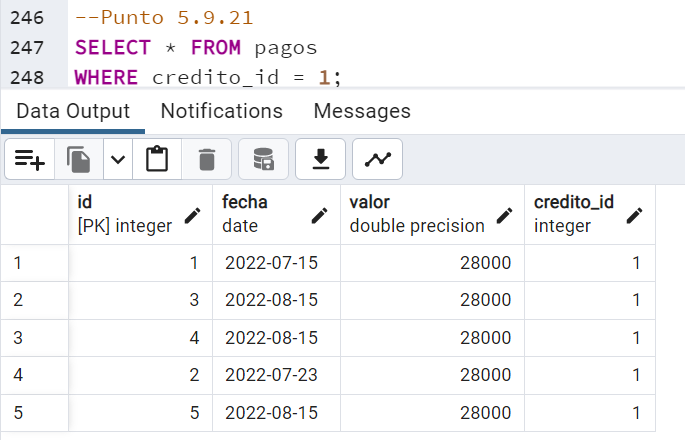


### Mostrar todos los Pagos insertados al Crédito con ID=1 insertados en la BD

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

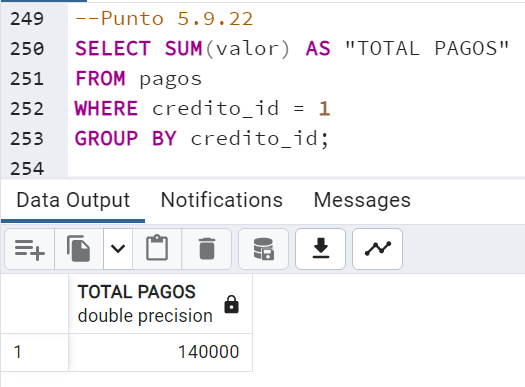
Sentencia SQL

SELECT SUM(valor) AS "TOTAL PAGOS"

FROM pagos

WHERE credito\_id = 1;

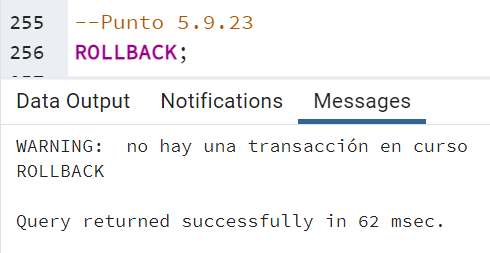
GROUP BY credito\_id;



### Cancelar la transaccion (rollblack)

Sentencia SQL

ROLLBACK;



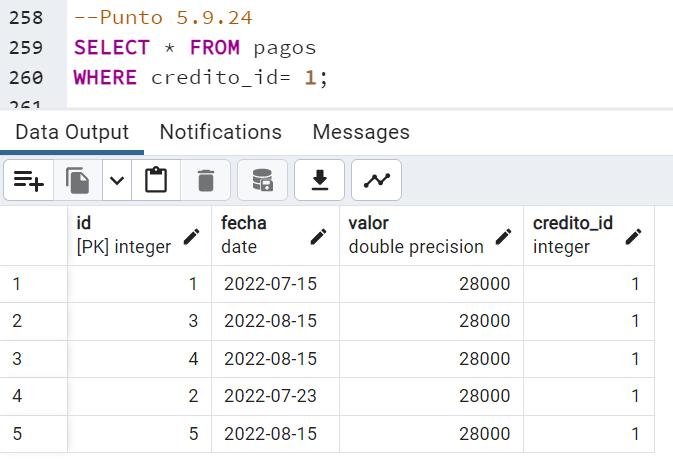
La sentencia ROLLBACK en este punto no aplica puesto que no hay una transacción en curso.

### 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

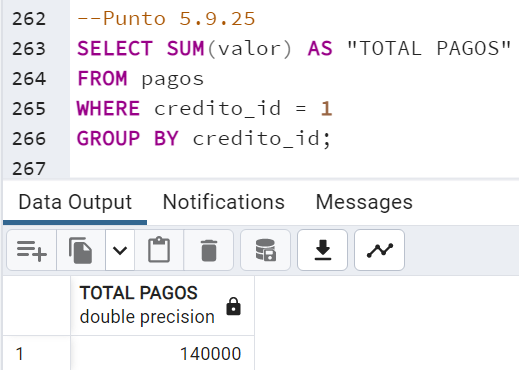
Sentencia SQL

SELECT SUM(valor) AS "TOTAL PAGOS"

FROM pagos

WHERE credito\_id = 1

GROUP BY credito\_id;



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

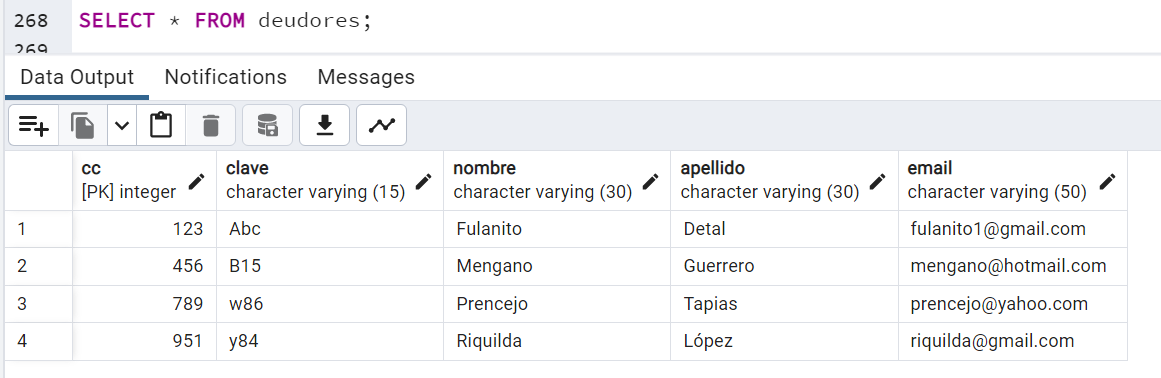
MySQL

### Ejemplo de subconsulta en el Select.

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

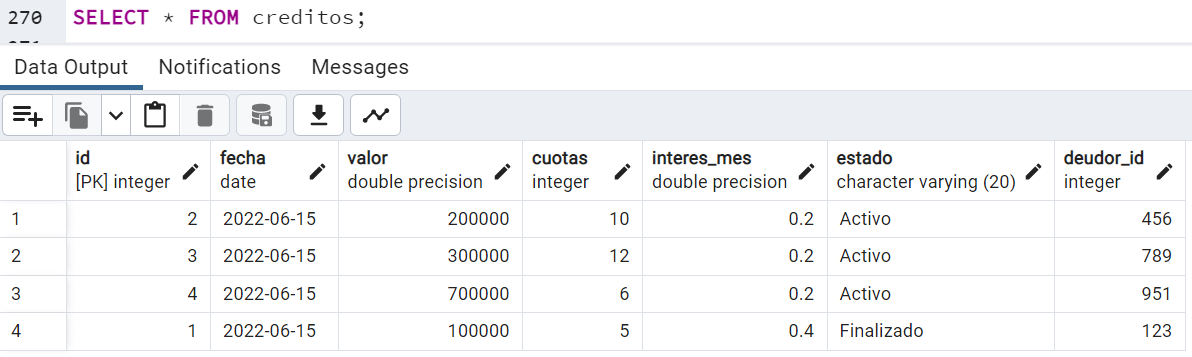
SELECT \* FROM deudores;

Tabla deudores



SELECT \* FROM creditos;

Tabla créditos



Sentencia SQL de la subconsulta en el FROM

SELECT d.nombre, (

SELECT COUNT(\*)

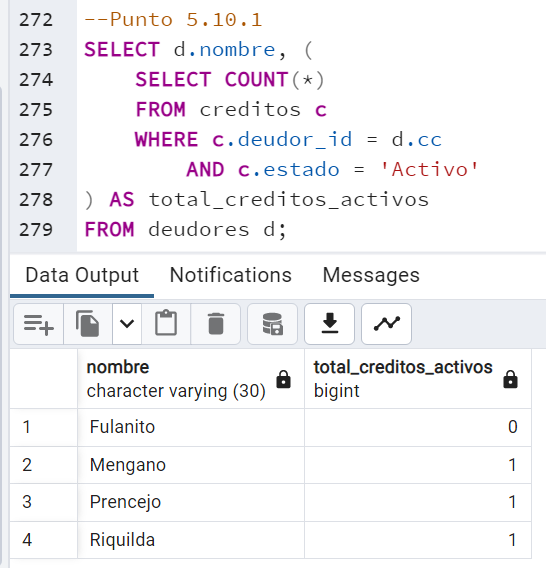
FROM creditos c

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

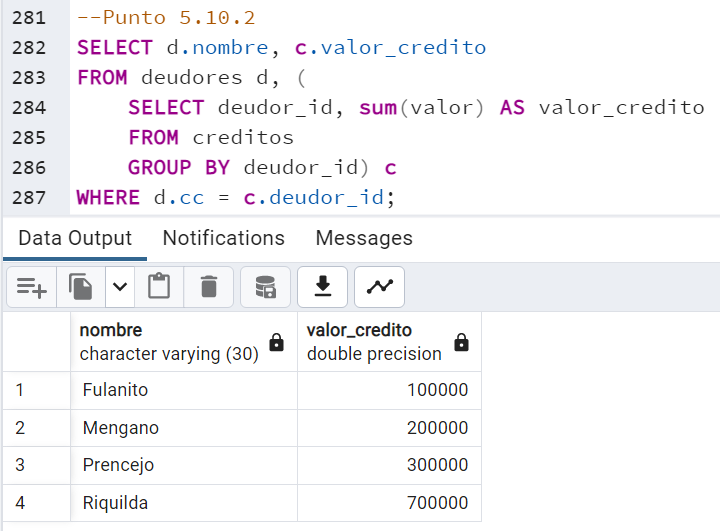
FROM deudores d, (

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

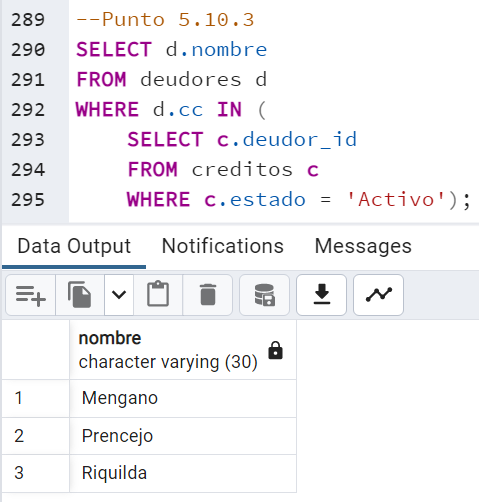
FROM deudores d

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 (

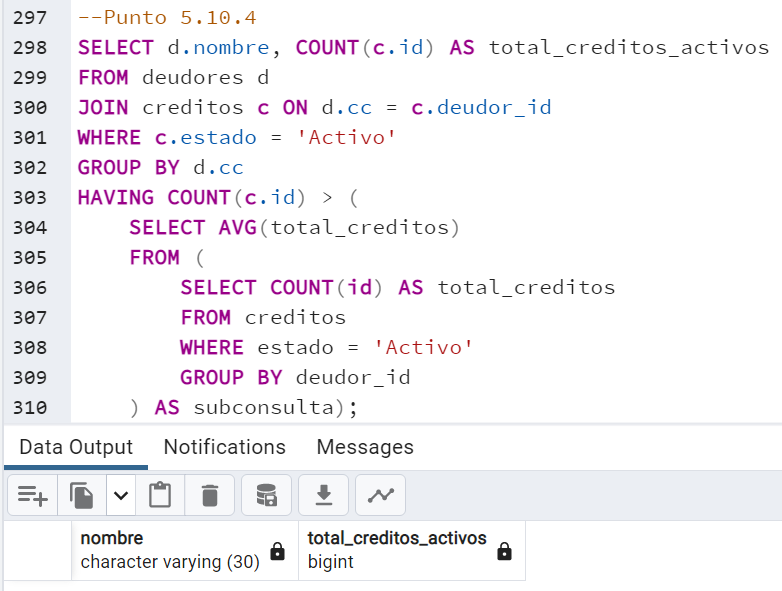
SELECT COUNT(id) AS total\_creditos

FROM creditos

WHERE estado = 'Activo'

GROUP BY deudor\_id

) AS subconsulta);



Se observa que ningún deudor cumple con el requisito solicitado, es decir todos los deudores tienen la misma cantidad de prestamos activos.

### Uso de Check

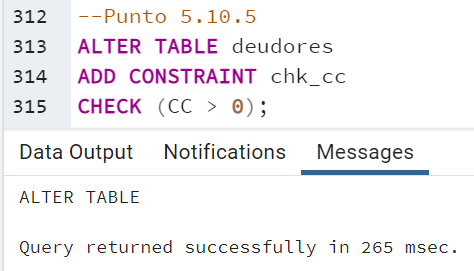
Se desea restringir el ingreso de numero de cedula menor o igual que cero en la tabla deudores en la columna cc.

Sentencia SQL

ALTER TABLE deudores

ADD CONSTRAINT chk\_cc

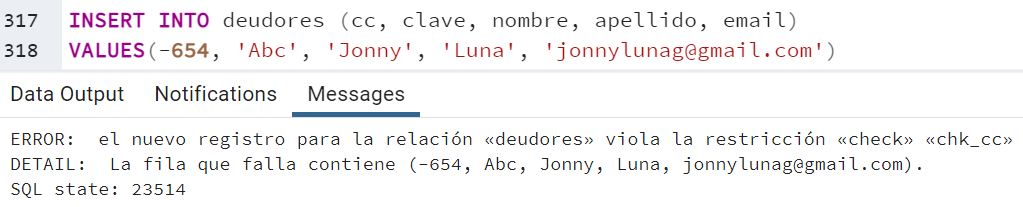
CHECK (CC > 0);



Ahora probamos la restricción hecha con la sentencia SQL tratando de ingresar un valor de cédula negativo.

INSERT INTO deudores

VALUES(-654, ‘Abc’, ‘Jonny’, ‘Luna’, ‘jonnylunag@gmail.com’)



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

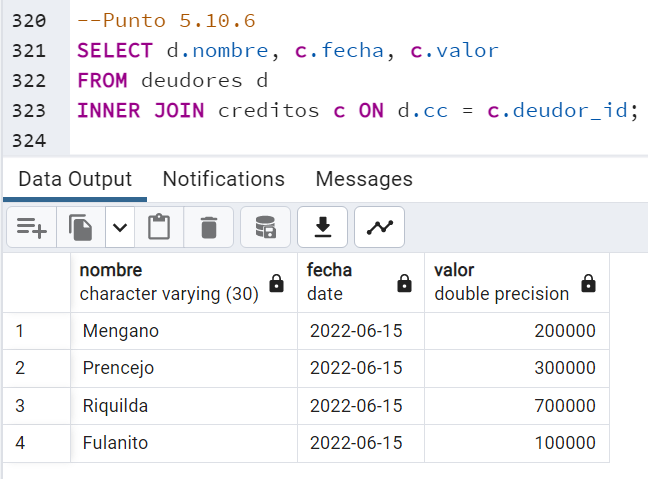
Se desea obtener una consulta que muestre el nombre del deudor en la tabla deudor y unirlo con la fecha y el valor del crédito de la tabla creditos.

Sentencia SQL

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

FROM deudores d

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



### Join Full

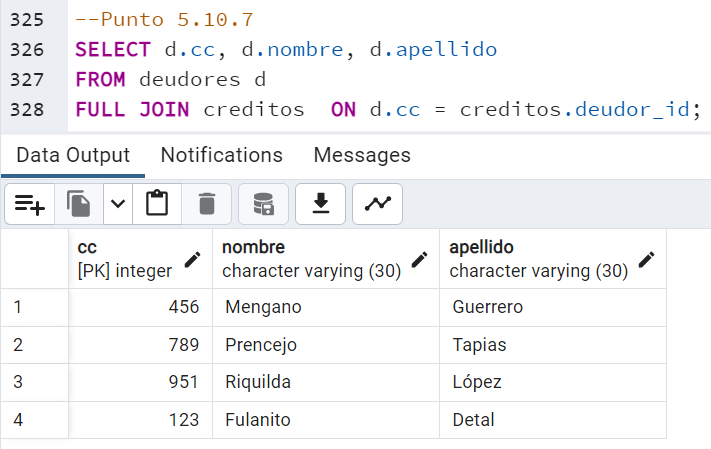
Se desea obtener una consulta que muestre el nombre del deudor en la tabla deudor y unirlo con la fecha y el valor del crédito de la tabla creditos.

Sentencia SQL

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

FROM deudores d

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



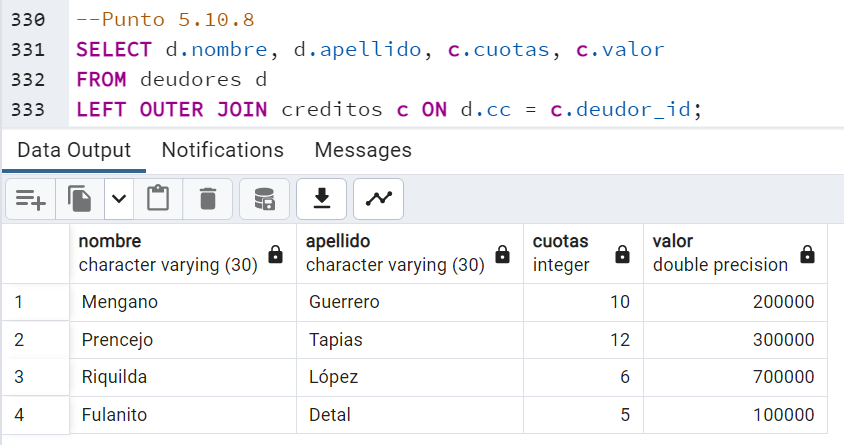
### Join Outer

Sentencia SQL

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

FROM deudores d

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



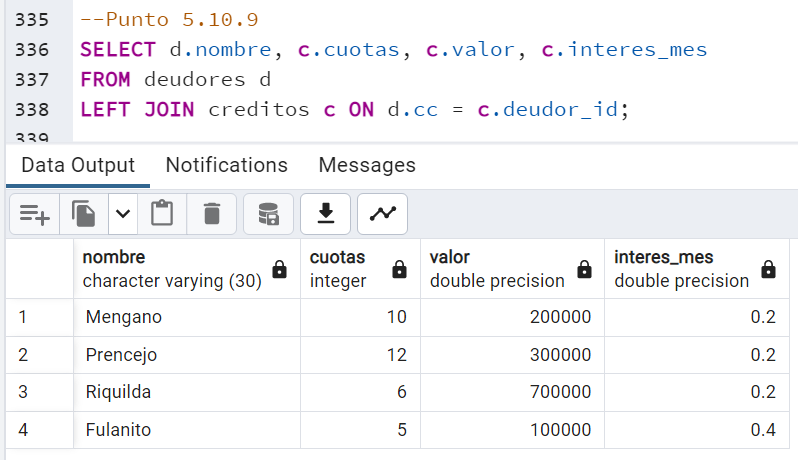
### Join Lef

Sentencia SQL

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

FROM deudores d

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



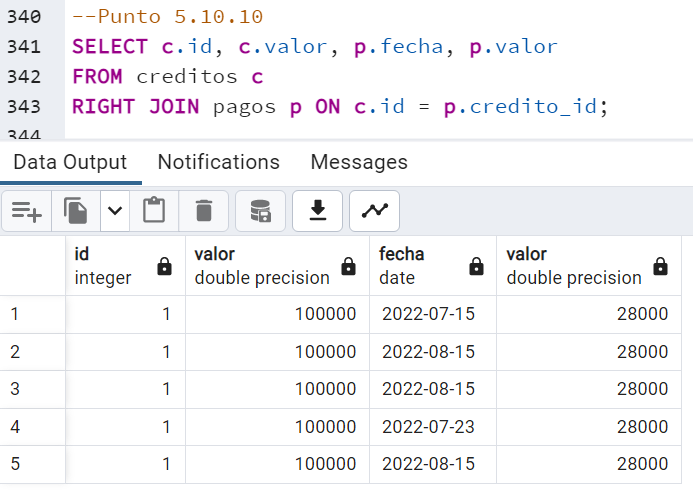
### Join Rigth

Sentencia SQL

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

FROM creditos c

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



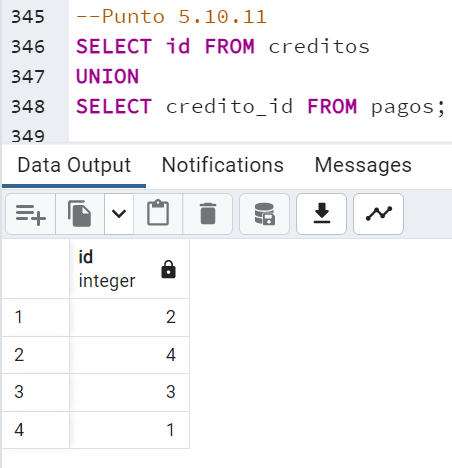
### Unión

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

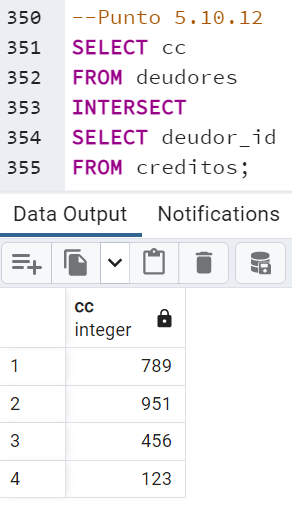
SELECT cc

FROM deudores

INTERSECT

SELECT deudor\_id

FROM creditos;



### Crear una Vista SQL

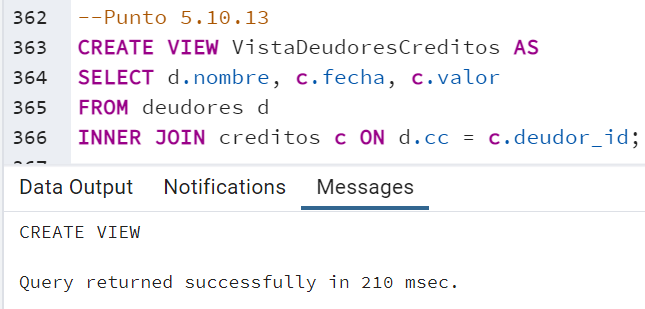
Sentencia SQL

CREATE VIEW VistaDeudoresCreditos AS

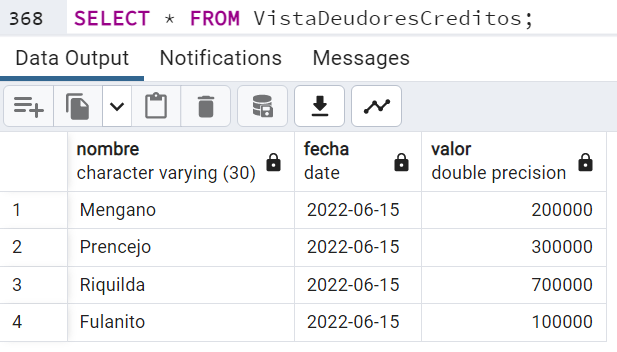
SELECT d.nombre, c.fecha, c.valor

FROM deudores d

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



SELECT \* FROM VistaDeudoresCreditos;



Aquí se observa la vista creada

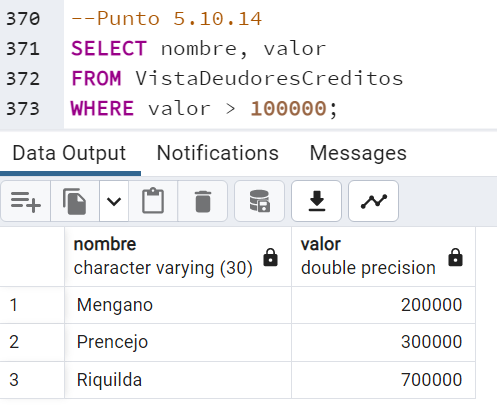
### Usar una Vista SQL

Sentencia SQL

SELECT nombre, valor

FROM VistaDeudoresCreditos

WHERE valor > 100000;



### Editar una Vista SQL

Sentencia SQL

CREATE OR REPLACE VIEW VistaDeudoresCreditos AS

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

FROM deudores d

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

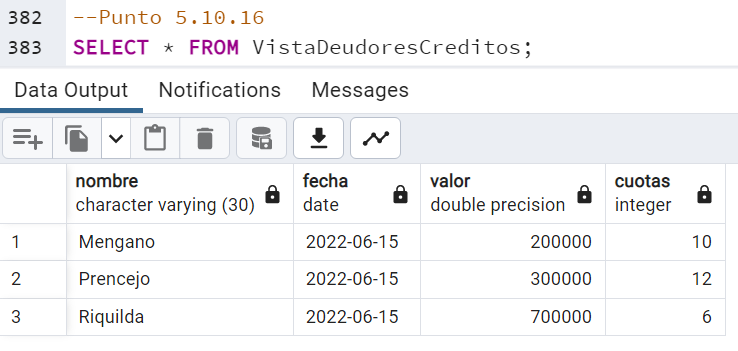
WHERE c.estado = 'Activo';



### Mostrar el contenido de una Vista SQL

Sentencia SQL

SELECT \* FROM VistaDeudoresCreditos;



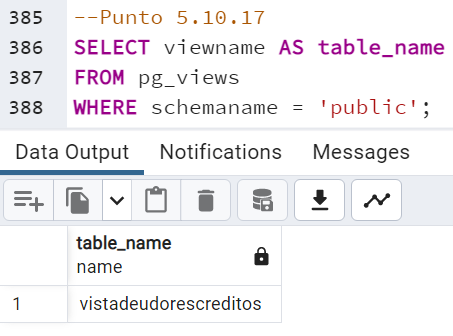
### Mostrar las Vistas SQL

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

SELECT viewname AS table\_name

FROM pg\_views

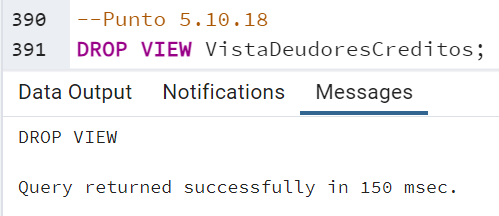
WHERE schemaname = 'public';



### Eliminar una Vista SQL

Sentencia SQL

DROP VIEW VistaDeudoresCreditos;



### Crear una Transacción SQL

En este ejemplo se hace un pago al crédito con id = 2, esta vez afectaremos en la transacción la tabla créditos en la cual se le descontarán

Sentencia SQL

START TRANSACTION;

--Insertar ocho nuevos registros de pago en la tabla "Pagos" al credito con id = 2

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

VALUES (6,'2022-07-15',24000, 2);

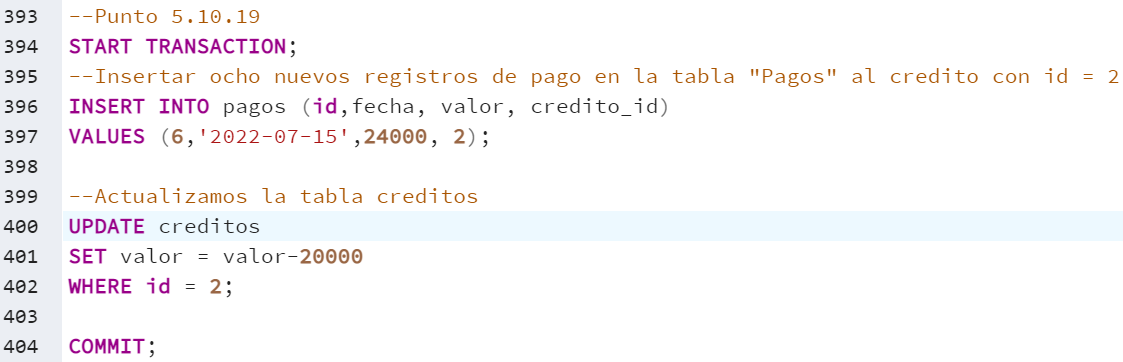
--Actualizamos la tabla creditos

UPDATE creditos

SET valor = valor-20000

WHERE id = 2;

COMMIT;



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

PostgreSQL no admite el bloque TRY CATCH pero se pueden manejar los errores de una forma distinta usando procedimientos almacenados.

Sentencia SQL

CREATE OR REPLACE PROCEDURE insertar\_pagos2(

p\_pago\_id INTEGER,

p\_fecha DATE,

p\_valor FLOAT,

p\_credito\_id INTEGER

)

LANGUAGE plpgsql

AS $$

DECLARE

v\_error\_message TEXT;

BEGIN

--se inicia la transacción

BEGIN

-- Validar que el valor del pago sea positivo

IF p\_valor <= 0 THEN

RAISE EXCEPTION 'El valor del pago debe ser mayor que cero.';

END IF;

-- Validar que la fecha de pago no sea mayor que la fecha actual

IF p\_fecha > CURRENT\_DATE THEN

RAISE EXCEPTION 'La fecha de pago no puede ser mayor que la fecha actual.';

END IF;

BEGIN

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

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

EXCEPTION

WHEN OTHERS THEN

GET STACKED DIAGNOSTICS v\_error\_message = MESSAGE\_TEXT;

RAISE NOTICE 'Error al insertar el pago: %', v\_error\_message;

RETURN;

END;

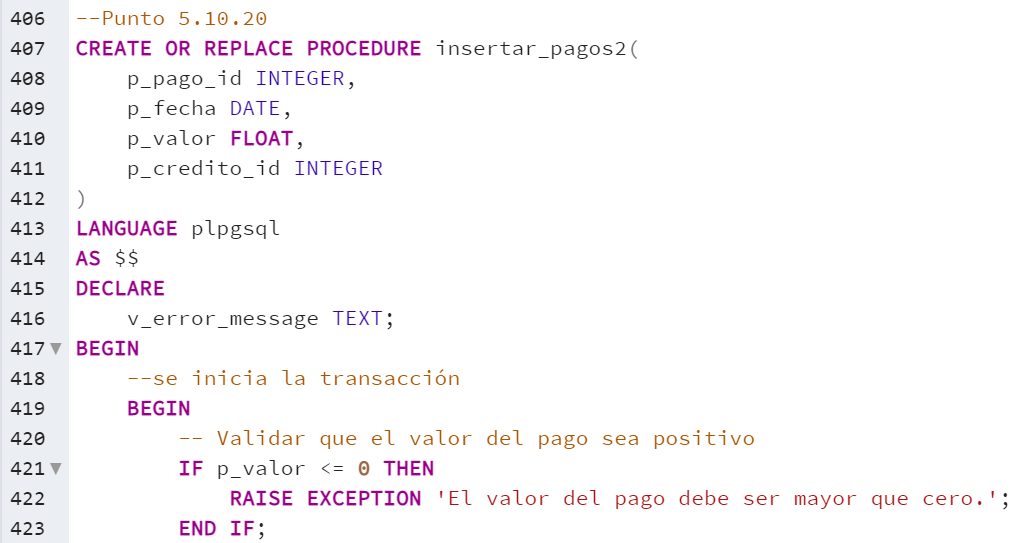
COMMIT;

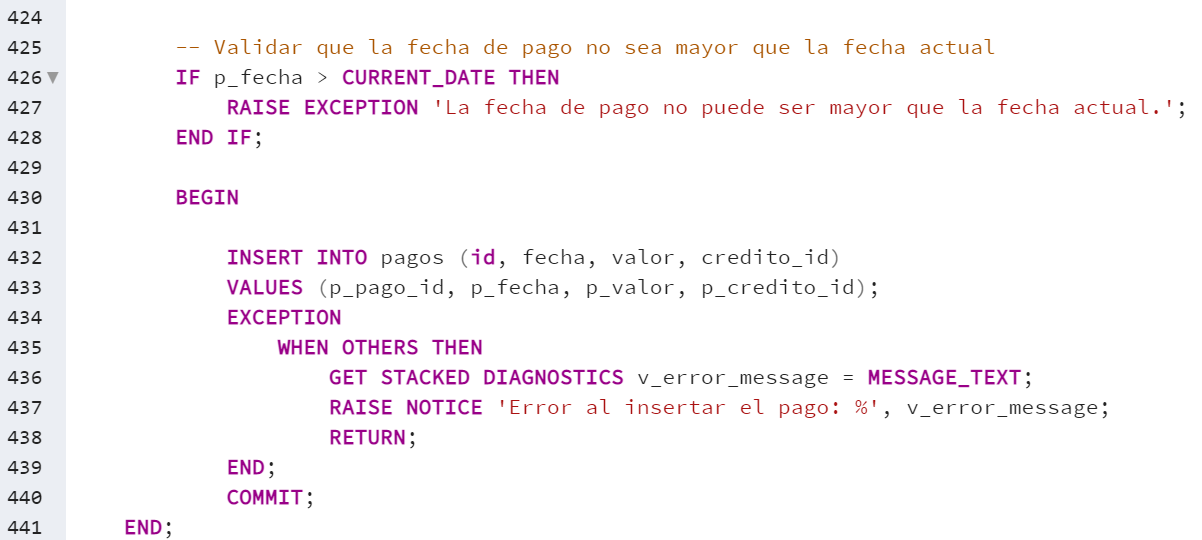
END;

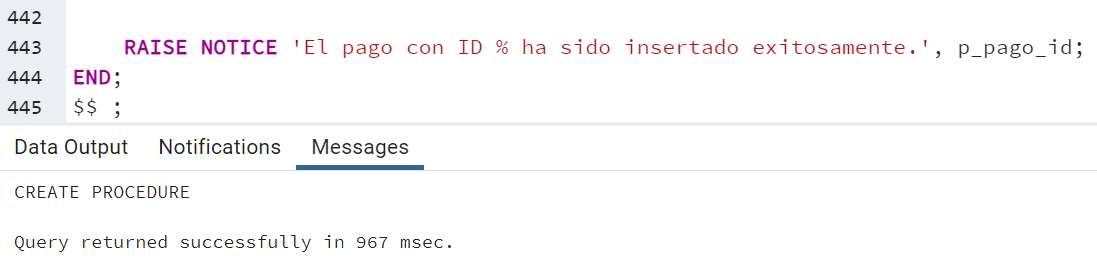
RAISE NOTICE 'El pago con ID % ha sido insertado exitosamente.', p\_pago\_id;

END;

$$ ;

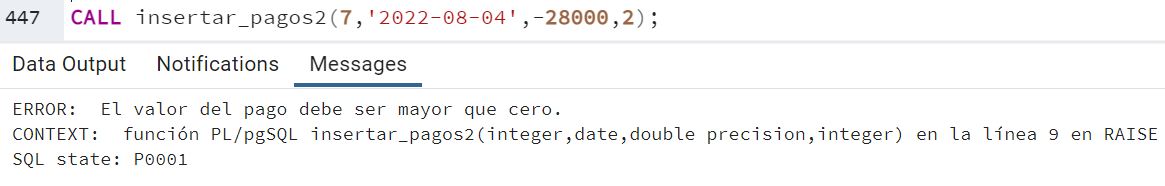






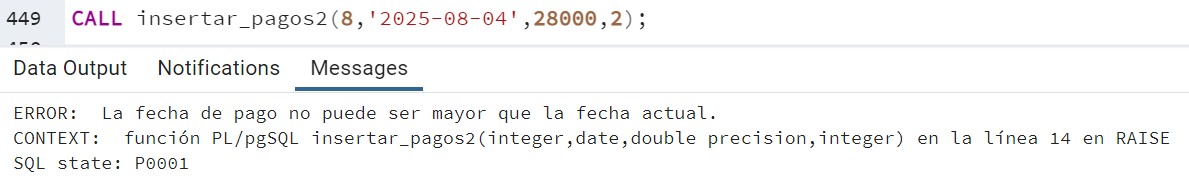
Al ejecutar el procedimiento almacenado con un valor del pago negativo tenemos:

CALL insertar\_pagos2(7,'2022-08-04',-28000,2);



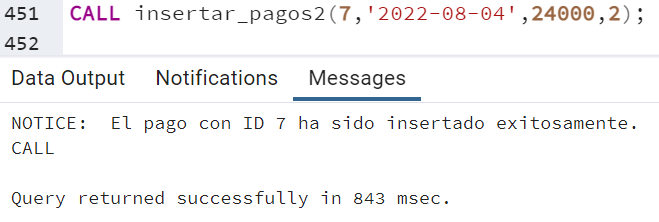
Al ejecutar el procedimiento almacenado con una fecha posterior a la fecha actual tenemos:

CALL insertar\_pagos2(8, '2025-08-04',28000,2);



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

CALL insertar\_pagos2('2025-08-04',28000,2);



Con los ejemplo anteriores observamos que el procedimiento almacenado captura los errores como lo haría el bloque TRY -CATCH.

.

# 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