**Bases de datos II**

**Actividad 4**

**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](#_Toc139449467)

[2 Introducción 3](#_Toc139449468)

[3 Objetivos 4](#_Toc139449469)

[3.1 Objetivo General: 4](#_Toc139449470)

[3.2 Objetivos específicos 4](#_Toc139449471)

[4 Justificación 5](#_Toc139449472)

[5 Desarrollo 6](#_Toc139449473)

[5.1 Qué es y cuál es el objetivo de un Disparador o Trigger. 6](#_Toc139449474)

[5.2 Cuáles son las diferencias más representativas que existen entre Funciones, Procedimientos almacenados y Disparadores. 7](#_Toc139449475)

[5.3 Cual es la sintaxis general de un disparador 7](#_Toc139449476)

[5.4 Para que se utilizan las palabras NEW, OLD, BEFORE, AFTER 8](#_Toc139449477)

[5.5 Crear y probar al menos 1 disparador (trigger) que se ejecute antes de insertar registros sobre alguna de las tablas de la BD, manipulando los datos nuevos insertados. 9](#_Toc139449478)

[5.6 Crear y probar al menos 1 disparador que se ejecute después de actualizar registros sobre alguna tabla de la BD, manipulando los datos antiguos (los que se van a reemplazar por los datos nuevos) 11](#_Toc139449479)

[5.7 Crear y probar al menos 1 disparador que se ejecute antes y después de actualizar registros sobre alguna tabla de la BD. 14](#_Toc139449480)

[5.8 Mostrar los disparadores de la o las tablas a las que pertenecen. 17](#_Toc139449481)

[5.9 Modificar al menos un disparador para que realice una operación diferente 17](#_Toc139449482)

[5.10 Eliminar al menos uno de los disparadores. 19](#_Toc139449483)

[5.11 Crear un disparador que permita actualizar el estado del crédito cuando se inserte un nuevo pago, el nuevo estado será AL DIA 19](#_Toc139449484)

[5.12 Crear un disparador que permita actualizar el estado crédito cuando se elimine un pago, el nuevo estado es POR AUDITAR 22](#_Toc139449485)

[5.13 Alterar la tabla Deudores para agregar una nueva columna llamada puntos de tipo entero 24](#_Toc139449486)

[5.14 Mostrar la estructura de la tabla Deudores 25](#_Toc139449487)

[5.15 Crear un disparador que permita cambiar incrementar en 1 el puntaje del deudor de un crédito, justo cuando se actualice el estado del crédito a TERMINADO 26](#_Toc139449488)

[6 Bibliografía 31](#_Toc139449489)

# 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 B-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 Disparadores y su aplicación en los motores de bases de datos relacionales OracleSQL. PostgreSQL y SQLServer.

## Qué es y cuál es el objetivo de un Disparador o Trigger.

Un trigger o disparador es un objeto que se almacena en una base de datos y está asociado a una tabla existente en la misma. El trigger se ejecuta automáticamente al momento en que sucede un evento que modifica la tabla cuando esta se manipula ya sea por que se realizó una inserción, actualización, borrado o cualquier otro evento u operación DML, Los trigger se utilizan mucho para implementar las reglas de negocio de una base de datos (Todopostgresql, 2023).

El objetivo de los triggers es permitir la ejecución de acciones personalizadas en la base de datos en respuesta a eventos específicos que ocurren en las tablas. Los trigger podrían tener como objetivo mantener la integridad de la base de datos ya que pueden utilizarse para realizar validaciones de datos, automatizar tareas ya que pueden actualizar automáticamente registros en otras tablas cuando se ejecute la función. Los trigger también se pueden utilizar para registrar los cambios que se hacen en la base de datos y que usuario los hizo, esto permite la realización de auditorías.

## Cuáles son las diferencias más representativas que existen entre Funciones, Procedimientos almacenados y Disparadores.

A diferencia de las funciones los disparadores no reciben parámetros como las funciones, pero si albergan funciones en su interior para ser ejecutadas.

Los disparadores se ejecutan automáticamente mientras que las funciones y los procedimientos almacenados deben ser ejecutados mediante una sentencia SQL.

Los disparadores se pueden ejecutar antes o después de que ocurra un evento en la tabla mientras que las funciones y los procedimientos se ejecutan en el momento en que son llamados.

Los Disparadores se ejecutan cuando existe un cambio en la tabla mientras que las funciones y los procedimientos almacenados ejecutan tareas diversas.

Los disparadores se utilizan para realizar auditorias en las bases de datos.

Los disparadores no devuelven ningún valor mientras que las funciones si y los procedimientos almacenados ejecutan funciones diversa que tampoco devuelven valor.

As funciones se pueden utilizar en las consultas o en otras funciones mientras que los disparadores no.

## Cual es la sintaxis general de un disparador

Los pasos para crear un trigger son:

1. Crear la función que se desea ejecutar con el trigger.
2. Crear el trigger en la tabla que se desea ejecutar.
3. Asociar el trigger a la tabla.

La sintaxis general de un trigger en PostgreSQL es:

CREATE TRIGGER nombre\_trigger

(BEFORE | AFTER | INSTEAD OF) // Temporalidad de ejecución del evento

{evento[OR…]} // evento que desencadenará puede ser INSERT | UPDATE | DELETE

ON nombre\_tabla

[FOR[EACH]{ROW | STATEMENT}] // Cláusula opcional que se utiliza para ejecutar el trigger en cada fila afectada por el evento

[WHEN(condición)] // es opcional

EXECUTE FUNCTION nombre\_función;

Para asociar el trigger a la tabla en PostgreSQL se realiza la siguiente sintaxis

ALTER TABLE nombre\_tabla

ENABLE TRIGGER nombre\_trigger;

(Imasgal, 2023)

## Para que se utilizan las palabras NEW, OLD, BEFORE, AFTER

PostgreSQL tiene ciertas variables especiales entre la que se encuentran NEW y OLD

NEW: es del tipo de dato es RECORD, es una variable que contiene la nueva fila de la tabla para las operaciones INSERT/UPDATE en disparadores del tipo row - level. Esta variable es NULL en disparadores del tipo statement – level. En conclusión NEW se utiliza para acceder o saber cuál es el valor que está siendo guardada en la variable, ejemplo NEW.nombre.

OLD: es del tipo de dato RECORD, es una variable que contiene la antigua fila de la tabla para las operaciones UPDATE/DELETE en disparadores del tipo row – level Esta variable es NULL en disparadores del tipo statement – level. En conclusión OLD se utiliza para acceder o conocer el valor de la variable existente y que será actualizado o eliminado, ejemplo, OLD.nombre.

BEFORE: Se refiere al omento en el que el trigger se ejecuta antes de que ocurra el evento. En un trigger que utilice BEFORE se pueden modificar los valores de las columnas en la fila NEW antes de que realice la operación.

AFTER: En este caso el trigger se ejecuta después de que ocurra el evento. En un trigger AFTER no es posible modificar los valores de las columnas en la fila NEW debido a que la operación ya se realizó.

Resolver los siguientes puntos sobre el ejercicio seleccionado por ustedes:

## Crear y probar al menos 1 disparador (trigger) que se ejecute antes de insertar registros sobre alguna de las tablas de la BD, manipulando los datos nuevos insertados.

Se requiere crear un trigger que antes de insertar registros a la tabla usuarios verifique si el usuario es mayor de edad.

Inicialmente creamos una función

CREATE OR REPLACE FUNCTION verificar\_mayoria\_edad()

RETURNS TRIGGER AS $$

BEGIN

IF NEW.fechanac > (CURRENT\_DATE - INTERVAL '18 years') THEN

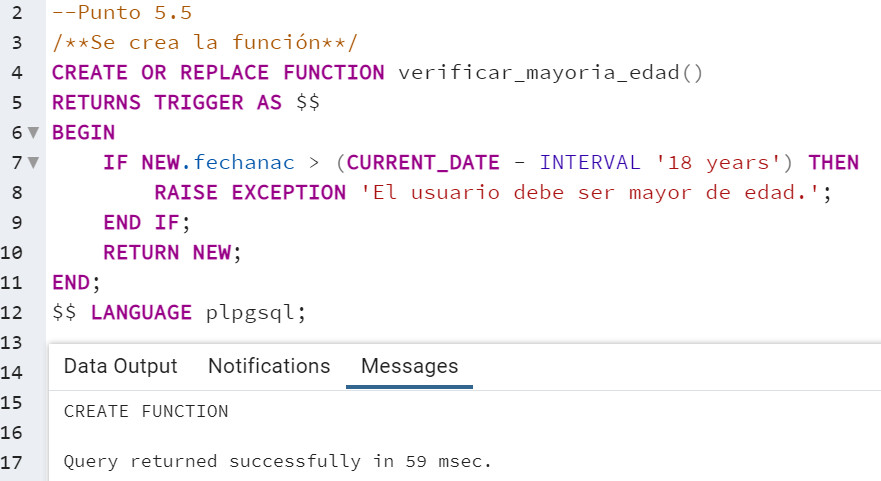
RAISE EXCEPTION 'El usuario debe ser mayor de edad.';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;



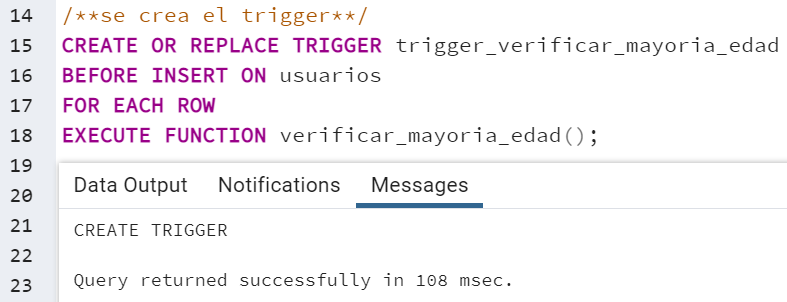
Se crea el Trigger

CREATE OR REPLACE TRIGGER trigger\_verificar\_mayoria\_edad

BEFORE INSERT ON usuarios

FOR EACH ROW

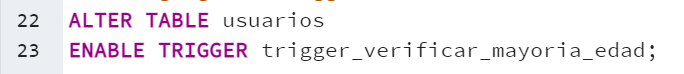
EXECUTE FUNCTION verificar\_mayoria\_edad();



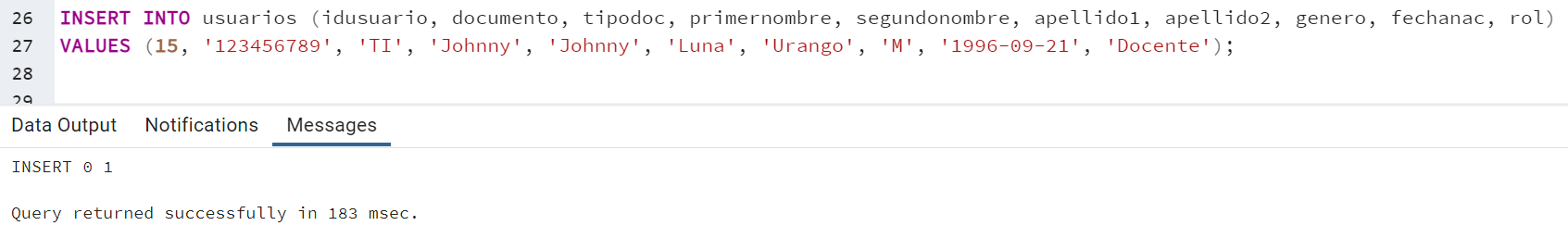
Se agrega el trigger a la tabla usuarios

ALTER TABLE usuarios

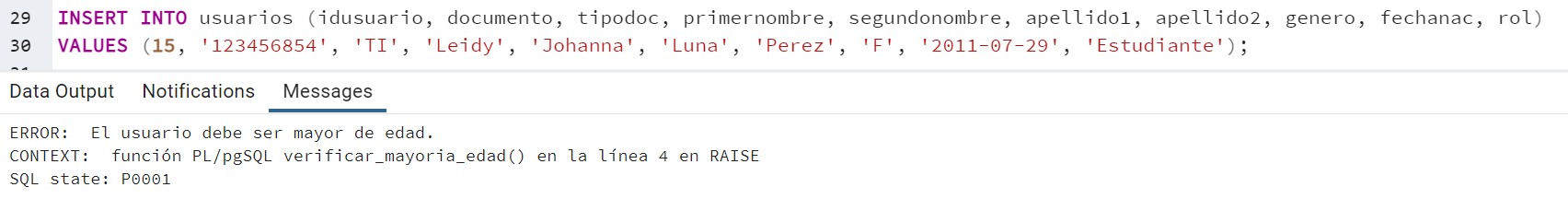
ENABLE TRIGGER trigger\_verificar\_mayoria\_edad;



Se inserta un usuario mayor de edad.



Se inserta un usuario menor de edad.



Se observa que no permite el ingreso del usuario por ser menor de edad.

## Crear y probar al menos 1 disparador que se ejecute después de actualizar registros sobre alguna tabla de la BD, manipulando los datos antiguos (los que se van a reemplazar por los datos nuevos)

Se desea modificar en la tabla asignatura la Intensidad horaria semanal de la asignatura **Ciencias Sociales** que actualmente tiene 3 horas y el nuevo valor es **4** horas en la columna **intensidadhs** con un trigger AFTER.

Se muestra la tabla actual



Se crea la función

CREATE OR REPLACE FUNCTION manipular\_datos\_antiguos()

RETURNS TRIGGER AS $$

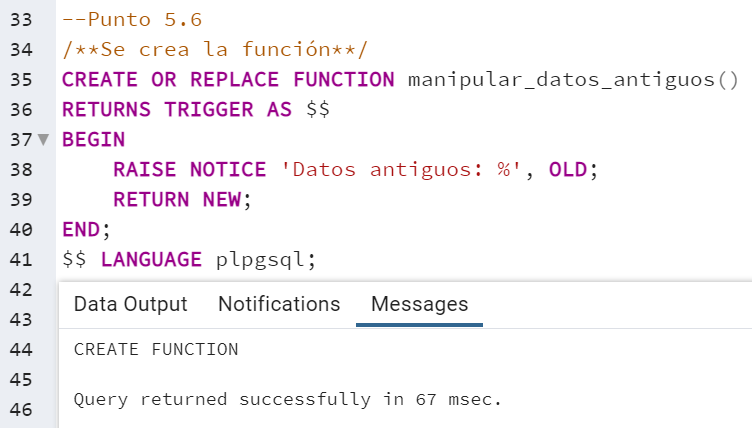
BEGIN

RAISE NOTICE 'Datos antiguos: %', OLD;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;



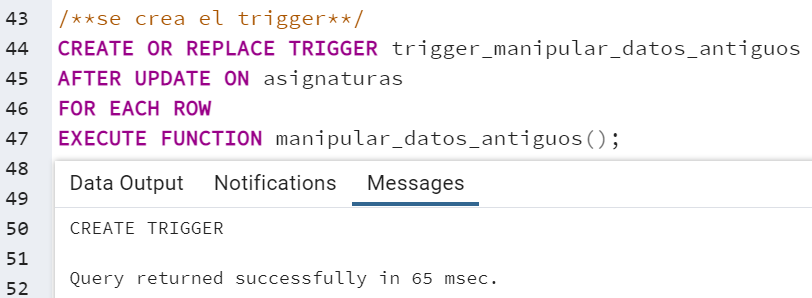
se crea el trigger

CREATE OR REPLACE TRIGGER trigger\_manipular\_datos\_antiguos

AFTER UPDATE ON asignaturas

FOR EACH ROW

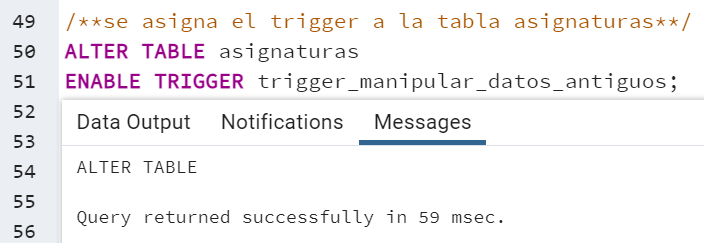
EXECUTE FUNCTION manipular\_datos\_antiguos();



Se asigna el trigger a la tabla asignaturas

ALTER TABLE asignaturas

ENABLE TRIGGER trigger\_manipular\_datos\_antiguos;

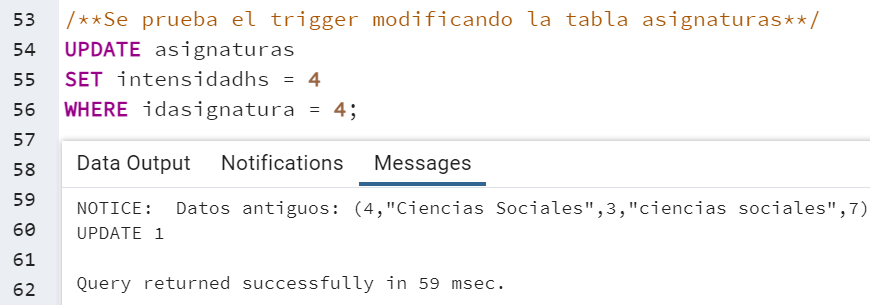


Se prueba el trigger modificando la tabla asignaturas

UPDATE asignaturas

SET intensidadhs = 4

WHERE idasignatura = 4;



Se verifican los cambios en la tabla asignaturas

SELECT \* FROM asignaturas;



## Crear y probar al menos 1 disparador que se ejecute antes y después de actualizar registros sobre alguna tabla de la BD.

En PostgreSQL no es posible que en un disparador se contemplen las dos opciones de BEFORE y AFTER en el mismo trigger por tal razón en este ejemplo se utiliza una misma función con dos triggers asignados a la misma tabla.

Se crea la función

CREATE OR REPLACE FUNCTION auditar\_actualizacion\_contratos()

RETURNS TRIGGER AS $$

BEGIN

-- Antes de la actualización

RAISE NOTICE 'Antes de la actualización - Datos antiguos: %', OLD;

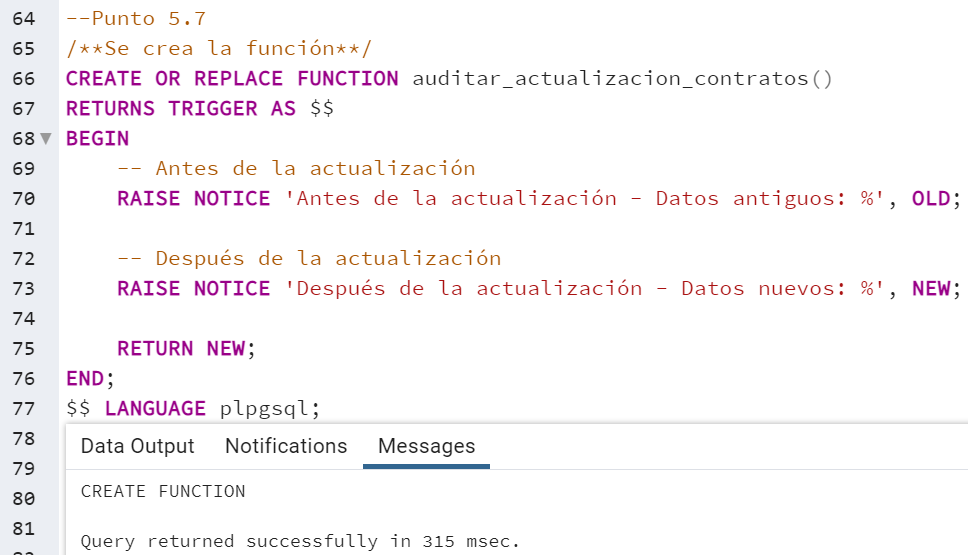
-- Después de la actualización

RAISE NOTICE 'Después de la actualización - Datos nuevos: %', NEW;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;



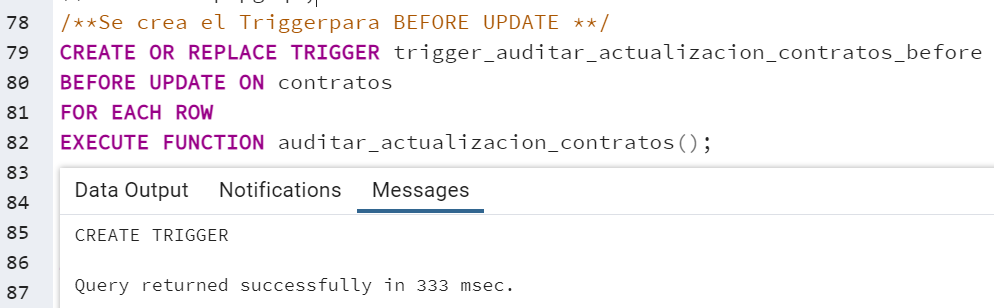
Se crea el Trigger para BEFORE UPDATE

CREATE OR REPLACE TRIGGER trigger\_auditar\_actualizacion\_contratos\_before

BEFORE UPDATE ON contratos

FOR EACH ROW

EXECUTE FUNCTION auditar\_actualizacion\_contratos();



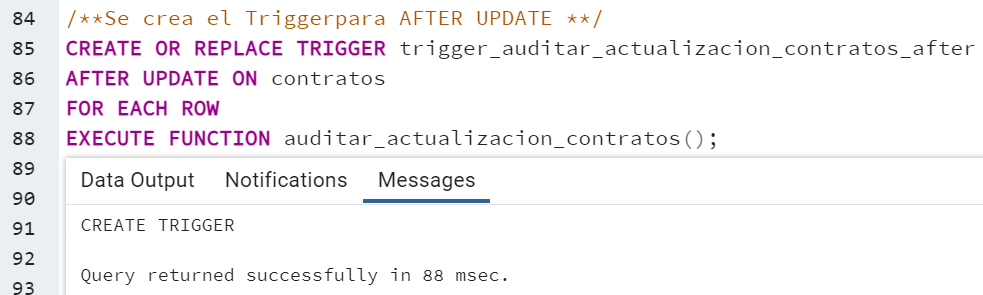
Se crea el Trigger para AFTER UPDATE.

CREATE OR REPLACE TRIGGER trigger\_auditar\_actualizacion\_contratos\_after

AFTER UPDATE ON contratos

FOR EACH ROW

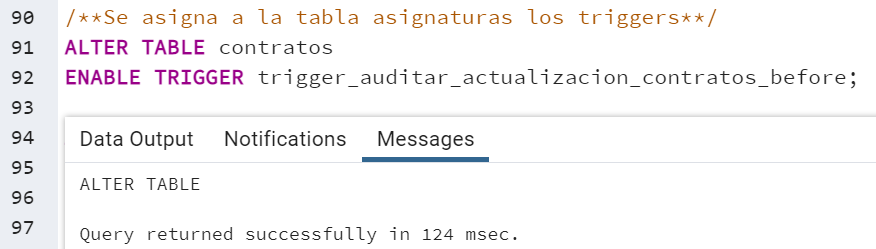
EXECUTE FUNCTION auditar\_actualizacion\_contratos();

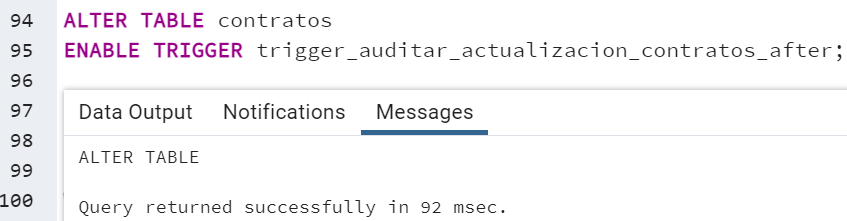


Se asigna a la tabla contratos los triggers creados.

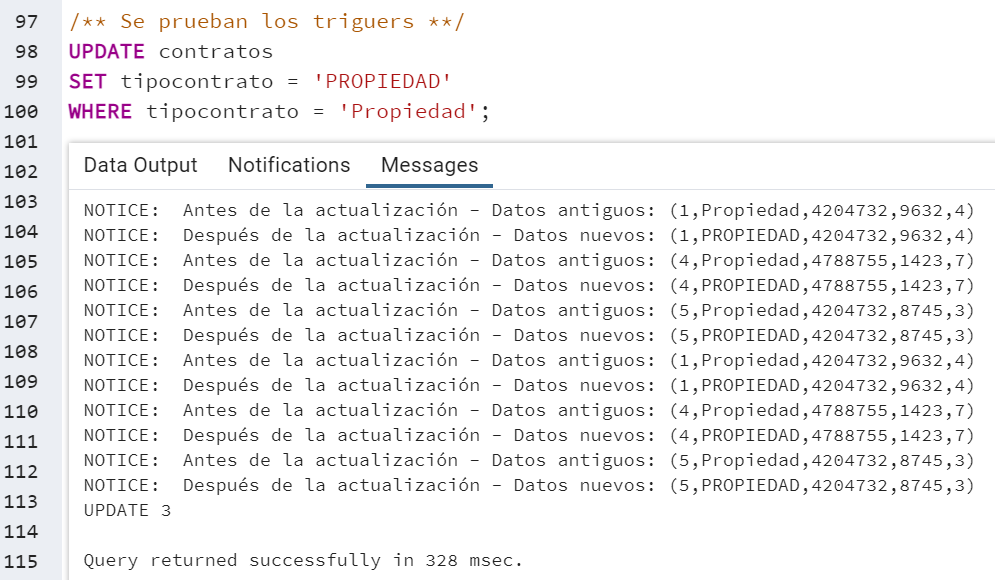
ALTER TABLE contratos

ENABLE TRIGGER trigger\_auditar\_actualizacion\_contratos\_before;





Se prueban los triggers.

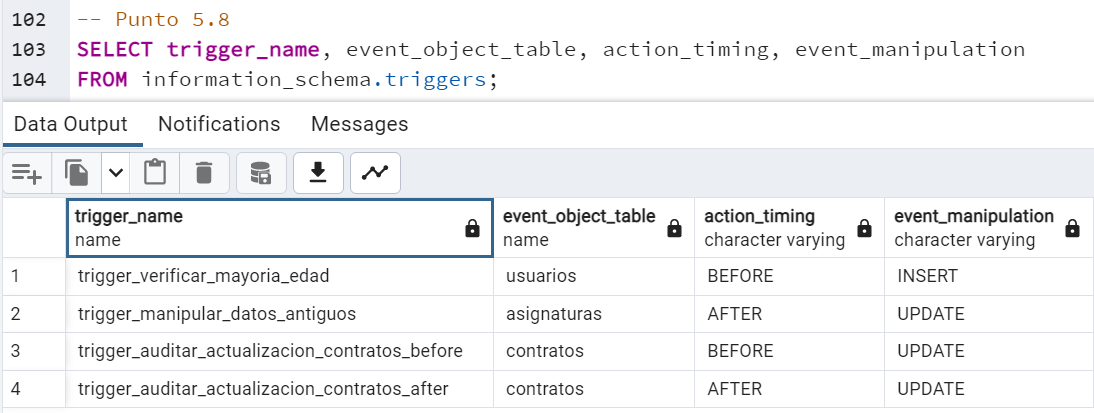


## Mostrar los disparadores de la o las tablas a las que pertenecen.

Para mostrar todos los triggers que se encuentran en la base de datos se ejecuta la siguiente consulta.

SELECT trigger\_name, event\_object\_table, action\_timing, event\_manipulation

FROM information\_schema.triggers;



## Modificar al menos un disparador para que realice una operación diferente

Para modificar un disparador para que realice una operación diferente solo basta con modificar la función, en este ejemplo se modifica la función del trigger del punto anterior y se ejecuta.

Se modifica la función

CREATE OR REPLACE FUNCTION auditar\_actualizacion\_contratos()

RETURNS TRIGGER AS $$

BEGIN

-- Antes de la actualización

RAISE NOTICE 'Antes de la actualización - Datos antiguos: %', OLD;

-- Incrementar el salario en un 15%

NEW.salario = OLD.salario \* 1.15;

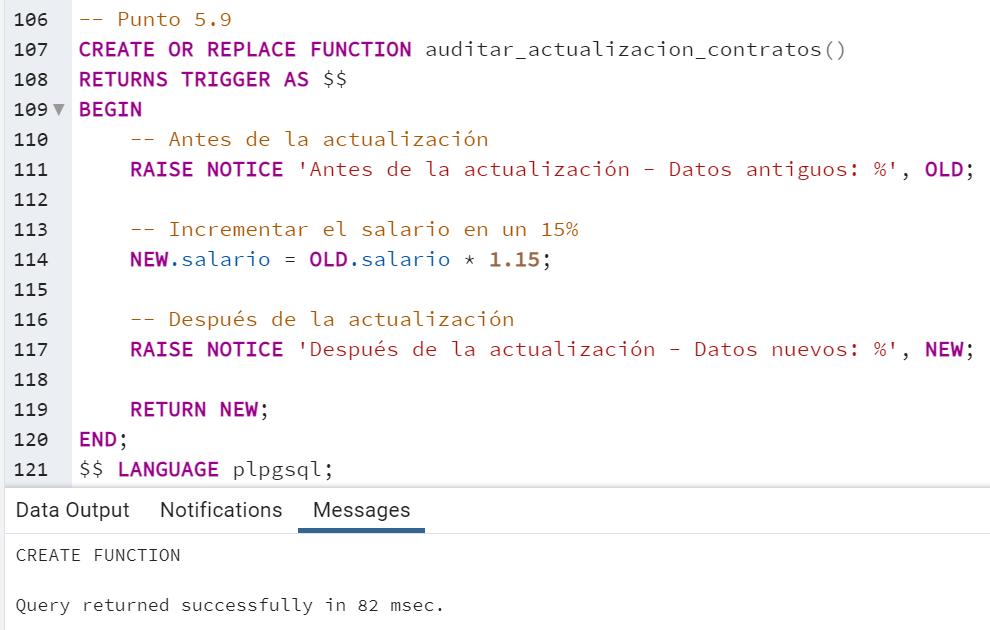
-- Después de la actualización

RAISE NOTICE 'Después de la actualización - Datos nuevos: %', NEW;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;



Se prueba el trigger

UPDATE contratos

SET tipocontrato = 'propiedad'

WHERE tipocontrato = 'PROPIEDAD';



Se observa que el salario incremento en un 15% para todos los docentes.

## Eliminar al menos uno de los disparadores.

Para resolver este punto se elimina el trigger del punto 5.6

DROP TRIGGER trigger\_manipular\_datos\_antiguos ON asignaturas;



Tomando como base el ejemplo de la base de datos **prueba\_transacciones\_grupoXYZ** del ejemplo de la actividad de la unidad 2, y deben resolver los siguientes puntos:

## Crear un disparador que permita actualizar el estado del crédito cuando se inserte un nuevo pago, el nuevo estado será AL DIA

Se crea la función

CREATE OR REPLACE FUNCTION actualizar\_estado\_credito()

RETURNS TRIGGER AS $$

BEGIN

UPDATE creditos

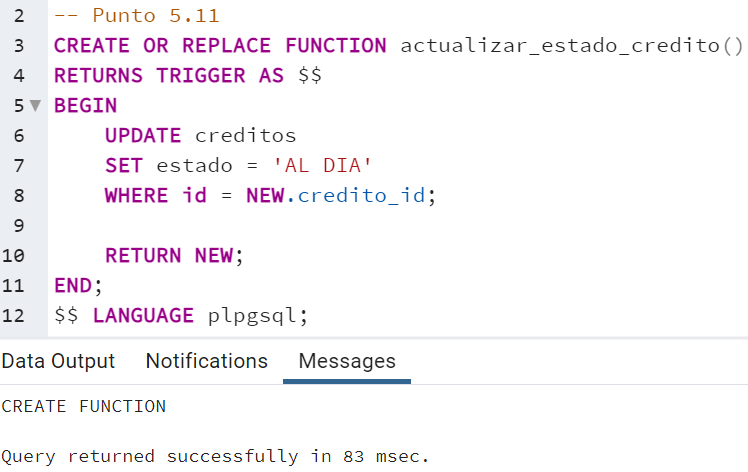
SET estado = 'AL DIA'

WHERE id = NEW.credito\_id;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;



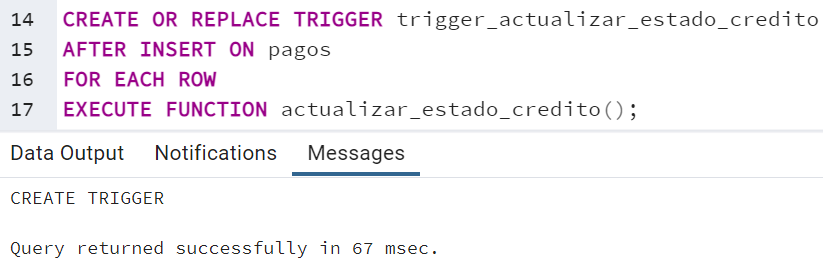
Se crea el trigger

CREATE OR REPLACE TRIGGER trigger\_actualizar\_estado\_credito

AFTER INSERT ON pagos

FOR EACH ROW

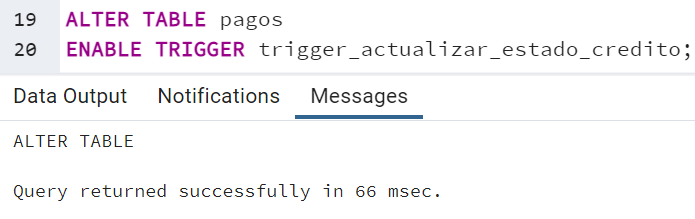
EXECUTE FUNCTION actualizar\_estado\_credito();



Se asigna a la tabla pagos el trigger

ALTER TABLE pagos

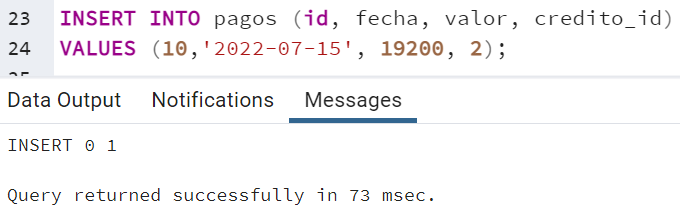
ENABLE TRIGGER trigger\_actualizar\_estado\_credito;



Se inserta un pago

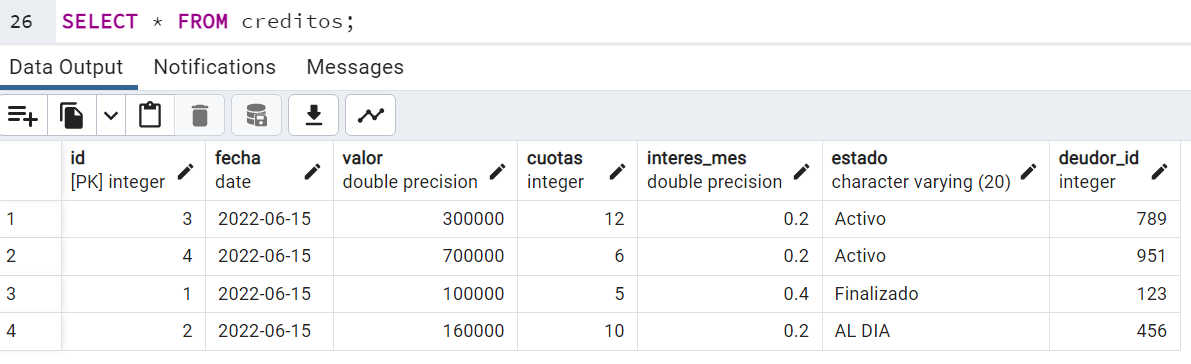
INSERT INTO pagos (id, fecha, valor, credito\_id)

VALUES (10,'2022-07-15', 19200, 2);



Se consulta la tabla créditos a ver el estado del crédito con id = 2.

SELECT \* FROM creditos;



Se observa que el trigger ha funcionado correctamente.

## Crear un disparador que permita actualizar el estado crédito cuando se elimine un pago, el nuevo estado es POR AUDITAR

Se crea la función

CREATE OR REPLACE FUNCTION actualizar\_estado\_credito()

RETURNS TRIGGER AS $$

BEGIN

UPDATE creditos

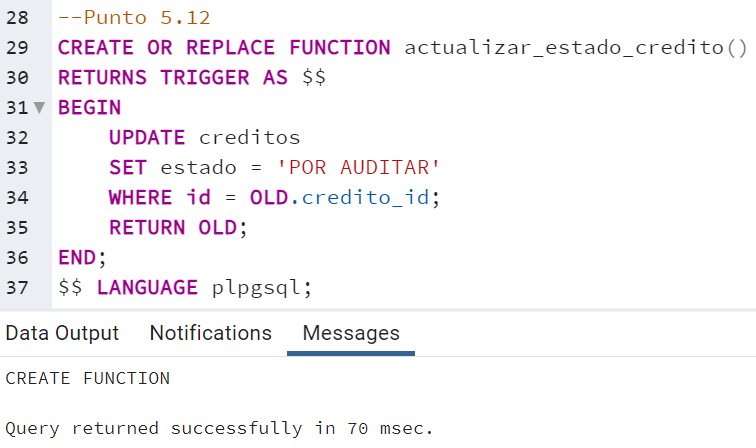
SET estado = 'POR AUDITAR'

WHERE id = OLD.credito\_id;

RETURN OLD;

END;

$$ LANGUAGE plpgsql;



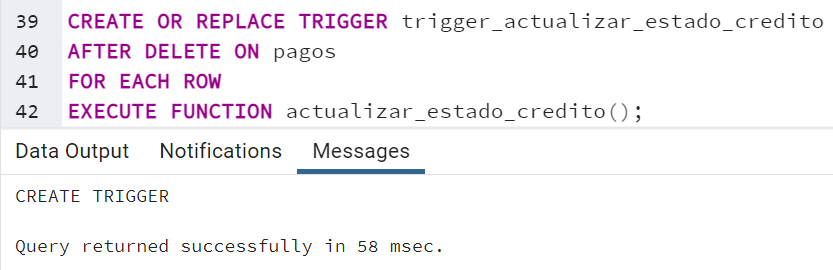
Se crea el trigger

CREATE OR REPLACE TRIGGER trigger\_actualizar\_estado\_credito

AFTER DELETE ON pagos

FOR EACH ROW

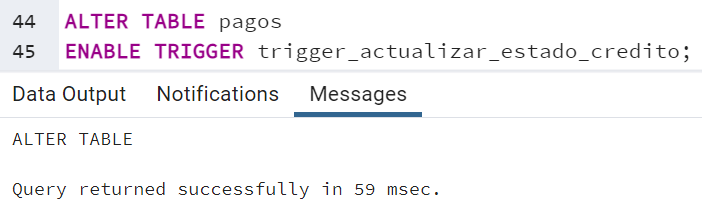
EXECUTE FUNCTION actualizar\_estado\_credito();



Se asigna a la tabla pagos el trigger

ALTER TABLE pagos

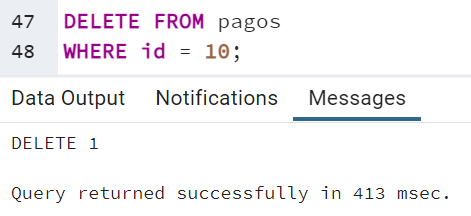
ENABLE TRIGGER trigger\_actualizar\_estado\_credito;



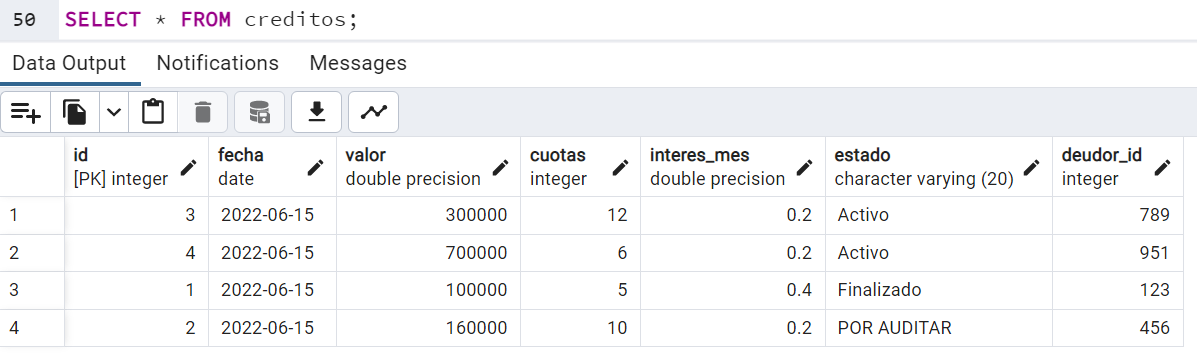
Se borra el pago realizado en el punto anterior para probar el trigger

DELETE FROM pagos

WHERE id = 10;



Se consulta la tabla créditos para verificar si el trigger funcionó

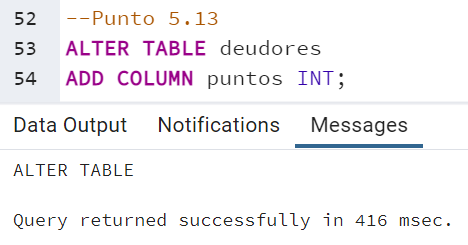


Se observa en el crédito con id = 2 el estado POR AUDITAR

## Alterar la tabla Deudores para agregar una nueva columna llamada puntos de tipo entero

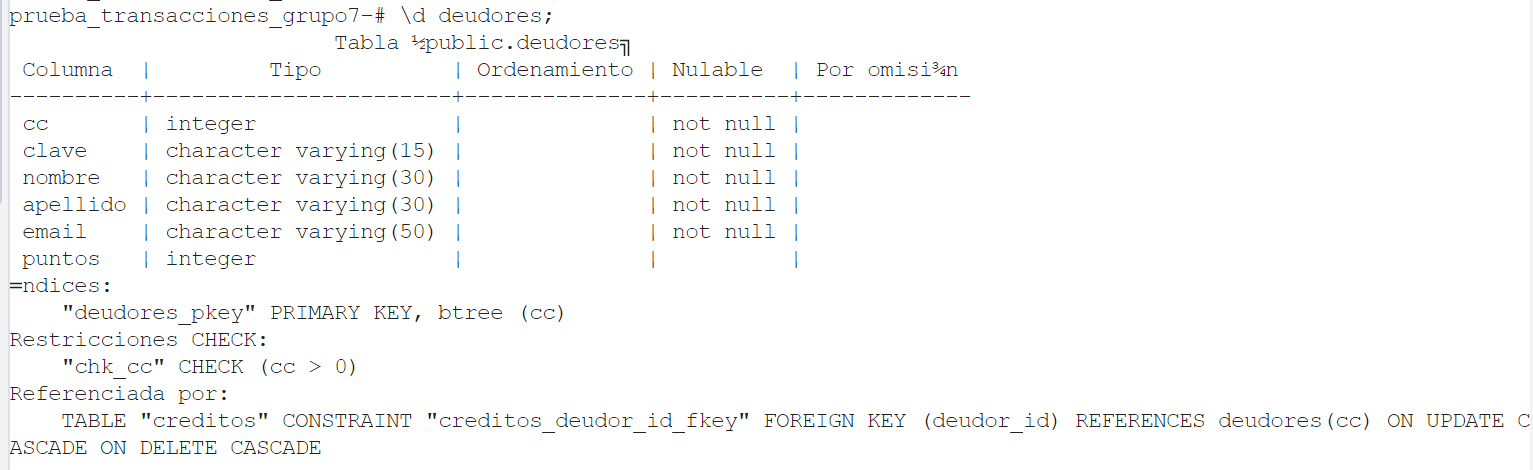
ALTER TABLE deudores

ADD COLUMN puntos INT;

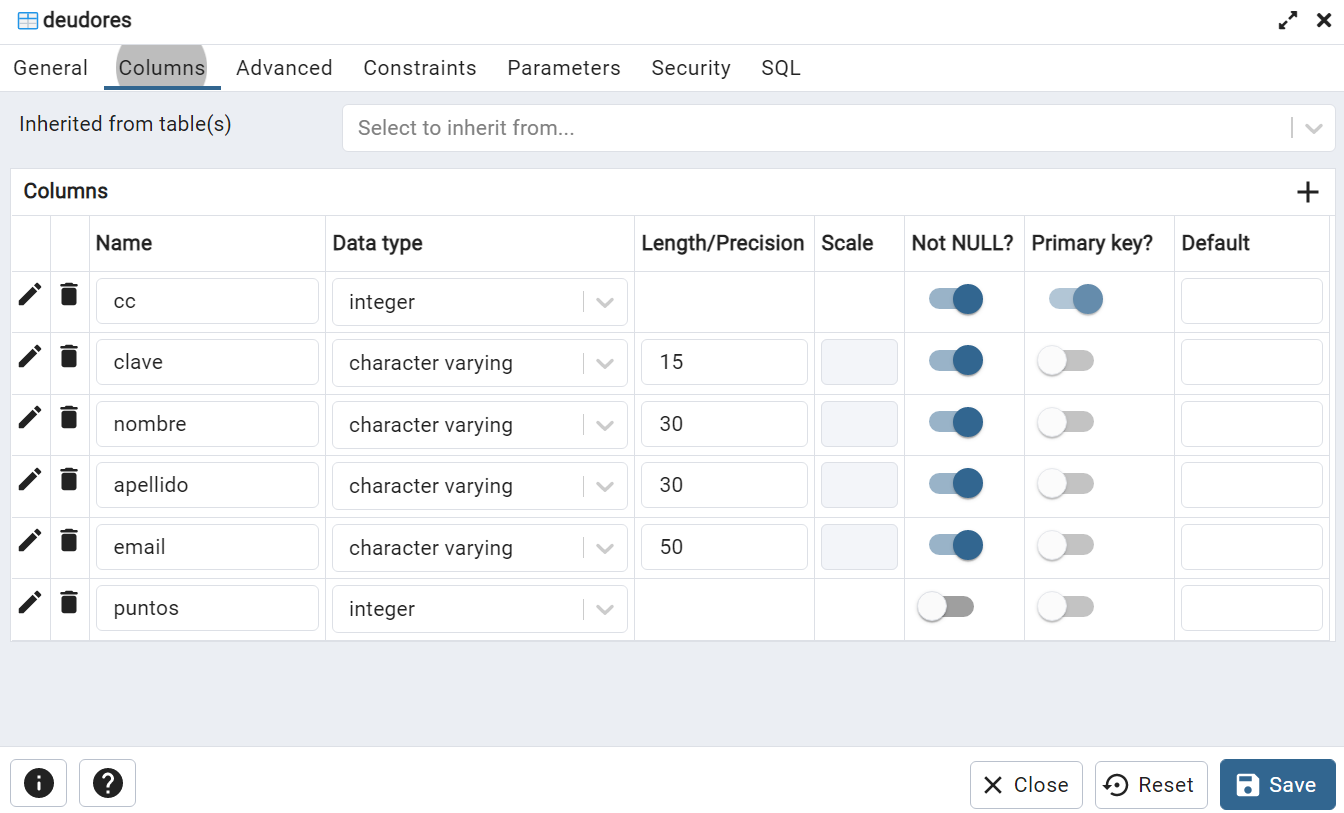


## Mostrar la estructura de la tabla Deudores

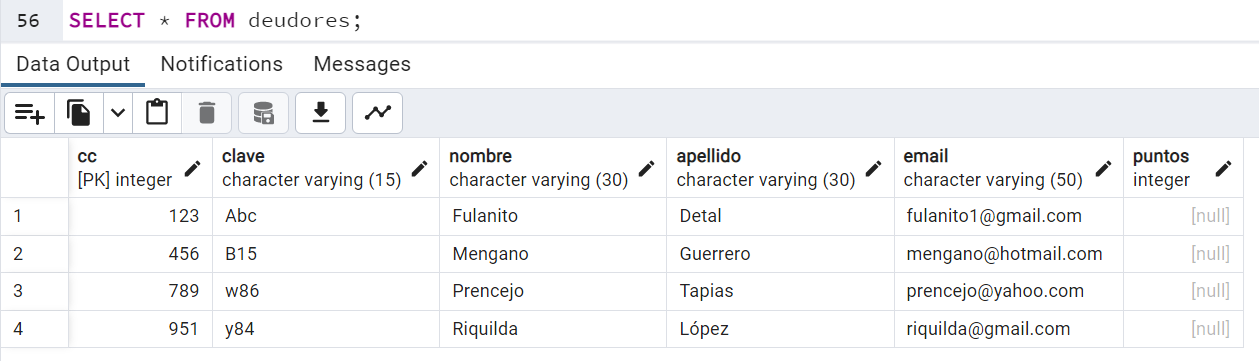
\d deudores



Otra forma



Hacemos un SELECT



## Crear un disparador que permita cambiar incrementar en 1 el puntaje del deudor de un crédito, justo cuando se actualice el estado del crédito a TERMINADO

Se Crea la función del trigger

CREATE OR REPLACE FUNCTION verificar\_pago\_completo()

RETURNS TRIGGER AS $$

DECLARE

total\_pagos FLOAT;

total\_deuda FLOAT;

BEGIN

-- Se Obtiene la suma del total de los pagos del mismo deudor

SELECT COALESCE(SUM(valor), 0) INTO total\_pagos

FROM pagos

WHERE credito\_id = NEW.credito\_id;

-- Se obtiene el valor total de la deuda del mismo crédito

SELECT valor \* (1+interes\_mes) INTO total\_deuda

FROM creditos

WHERE id = NEW.credito\_id;

-- Se verifica si la deuda ha sido cancelada en su totalidad

IF total\_pagos >= total\_deuda THEN

-- se actualiza el estado del crédito a 'TERMINADO'

UPDATE creditos

SET estado = 'TERMINADO'

WHERE id = NEW.credito\_id;

-- Incrementar en 1 el valor de la columna 'puntos' en la tabla 'deudores'

UPDATE deudores

SET puntos = puntos + 1

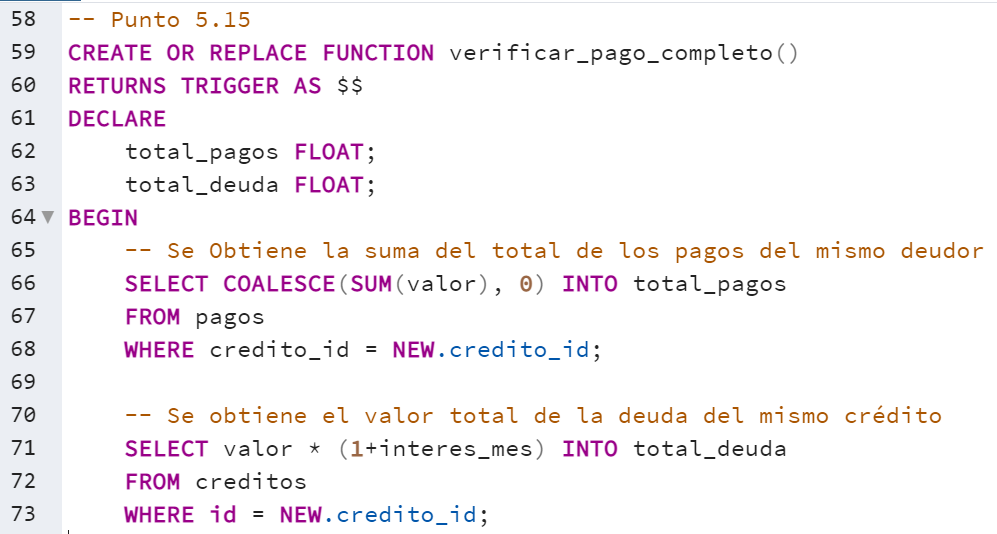
WHERE cc = (SELECT deudor\_id FROM creditos WHERE id = NEW.credito\_id);

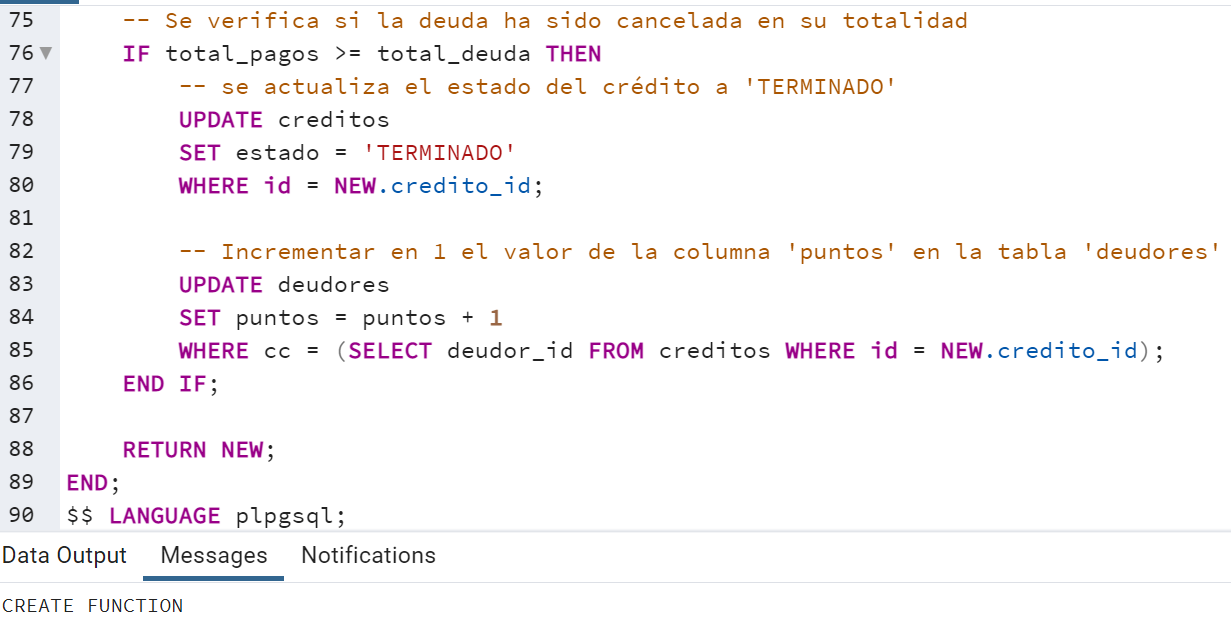
END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;





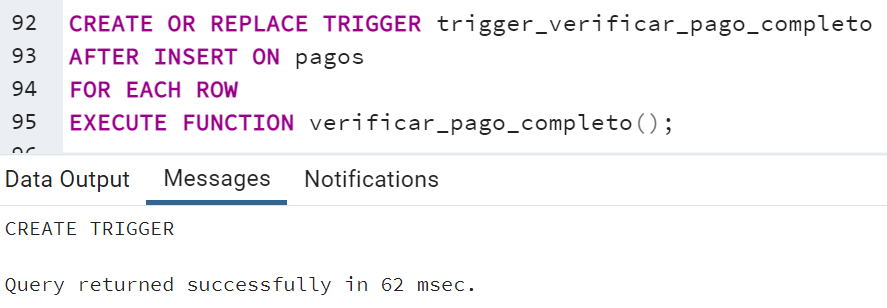
Se crea el Trigger

CREATE OR REPLACE TRIGGER trigger\_verificar\_pago\_completo

AFTER INSERT ON pagos

FOR EACH ROW

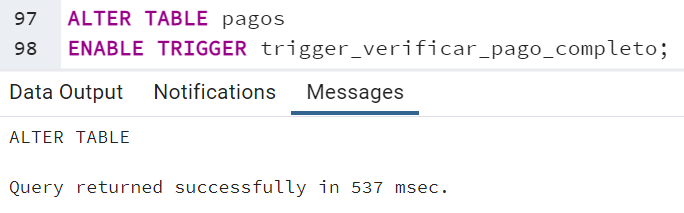
EXECUTE FUNCTION verificar\_pago\_completo();



Se asigna a la tabla pagos el trigger

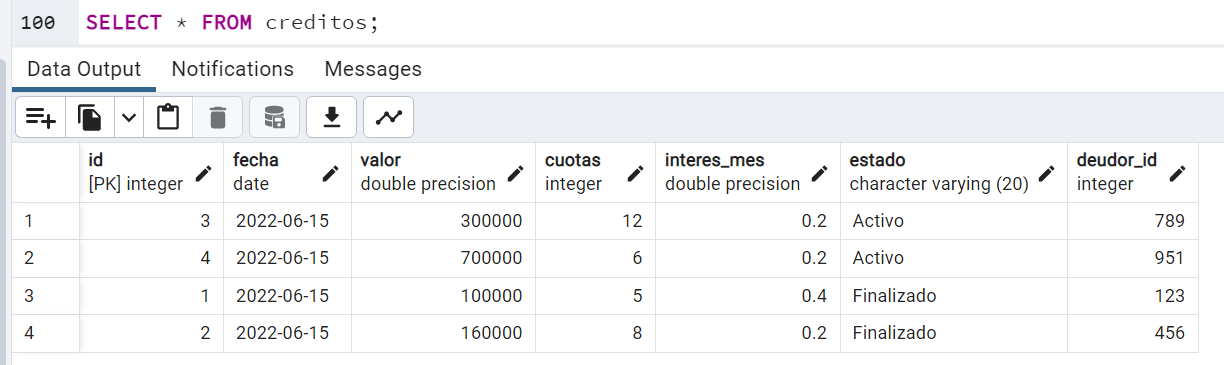
ALTER TABLE pagos

ENABLE TRIGGER trigger\_verificar\_pago\_completo;



Verificamos la tabla créditos

SELECT \* FROM creditos;



Se observa que el crédito con id = 4 está Activo o sin finalizar, por lo tanto haremos este ejercicio haciendo 6 pagos por valor de $ 140.000 llamando a la función **insertar\_pagos()** creada en la actividad 2

CALL insertar\_pagos(14,'2022-07-15',140000,4);

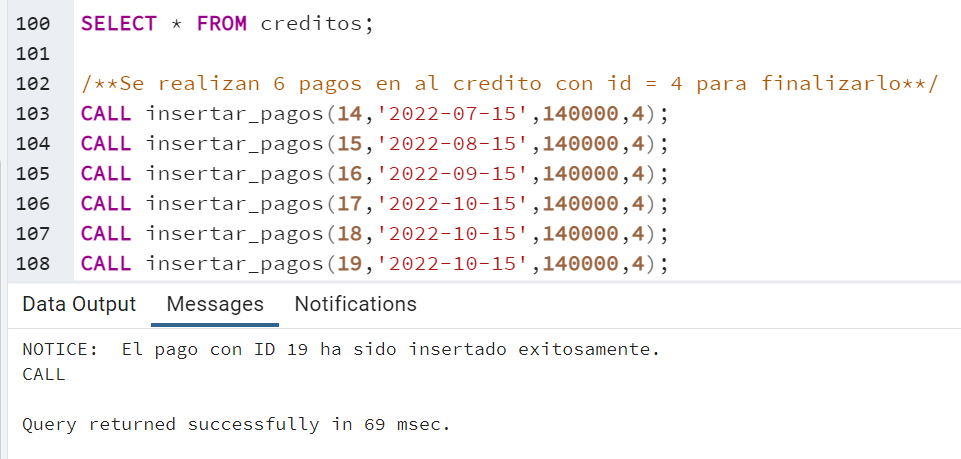
CALL insertar\_pagos(15,'2022-08-15',140000,4);

CALL insertar\_pagos(16,'2022-09-15',140000,4);

CALL insertar\_pagos(17,'2022-10-15',140000,4);

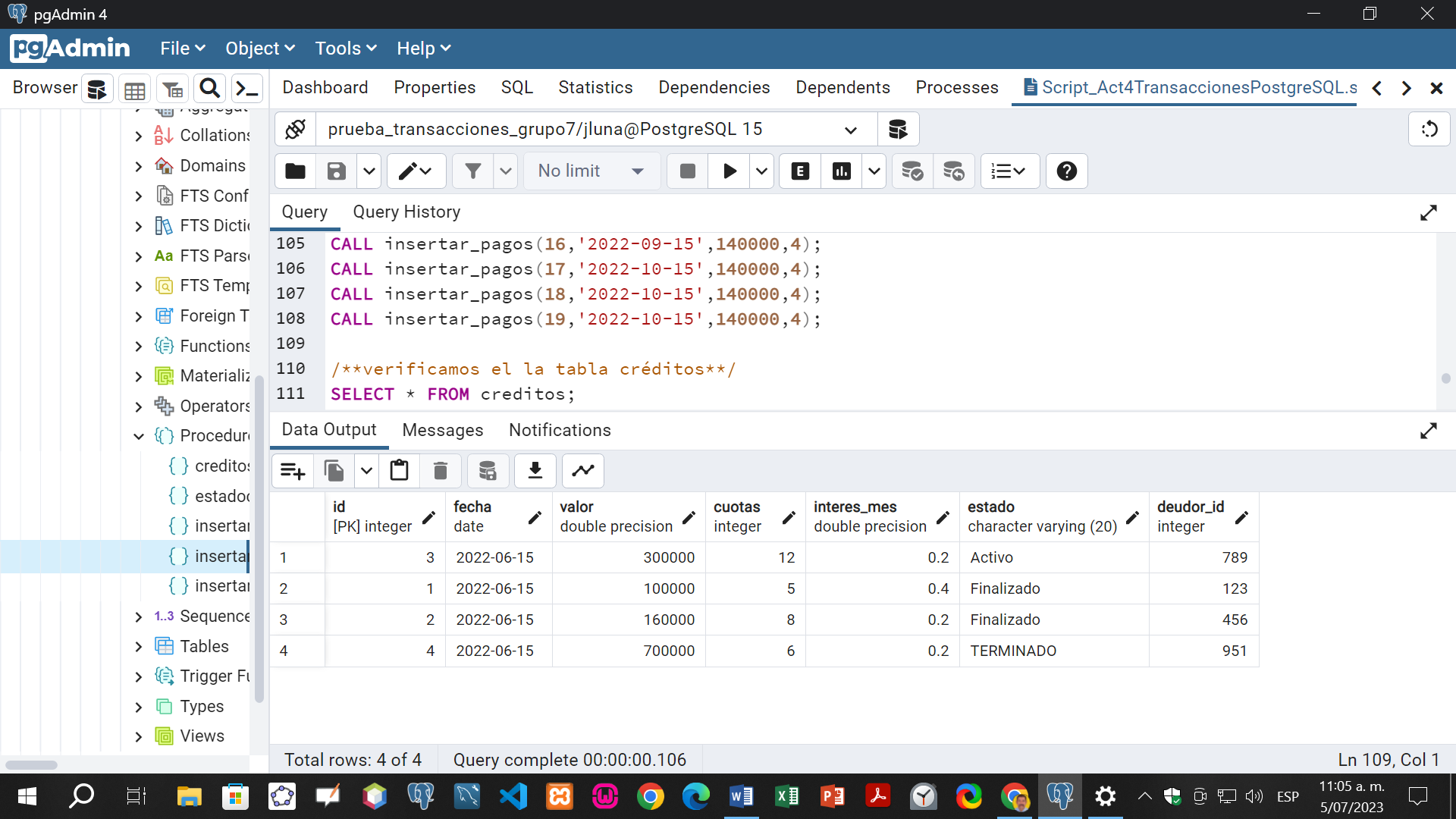
CALL insertar\_pagos(18,'2022-10-15',140000,4);

CALL insertar\_pagos(19,'2022-10-15',140000,4);



Se veerifica la tabla créditos

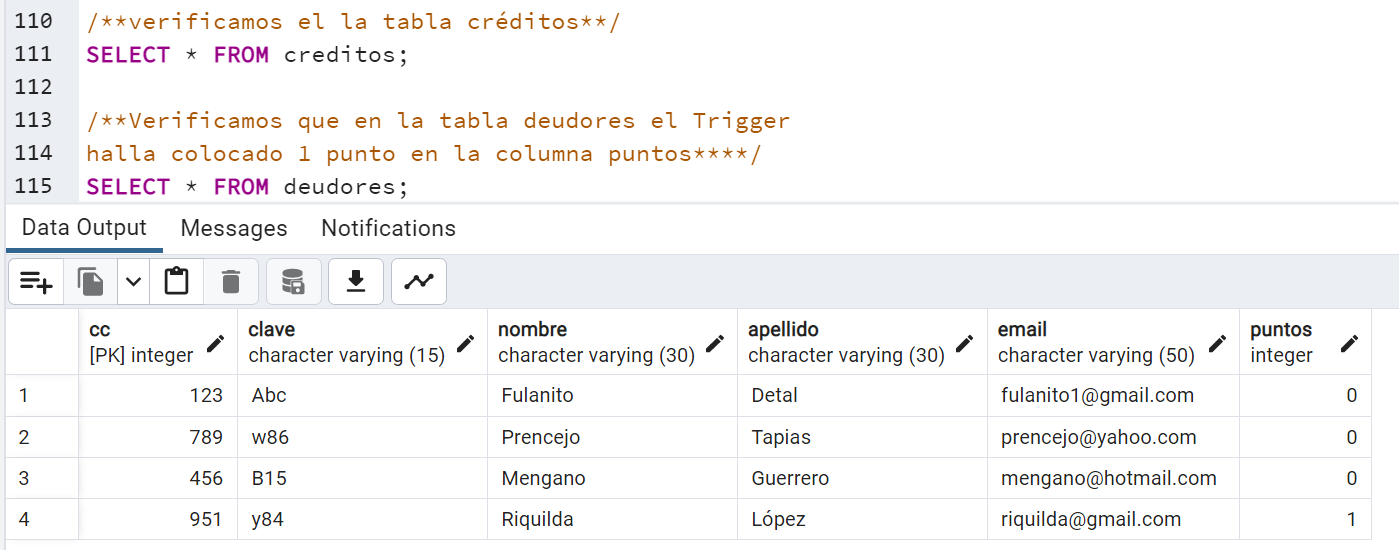
SELECT \* FROM creditos;



Se observa el crédito con id = 4 en estado TERMINADO,

Verificamos que en la tabla deudores se halla insertado un 1 en la columna puntos del crédito con id = 4.

SELECT \* FROM deudores;



Se observa que en la columna puntos del deudor con cc = 951 tiene un punto.

# Bibliografía

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

Emc2Net. (28 de 06 de 2023). *e-mc2.net*. Obtenido de e-mc2.net: https://e-mc2.net/blog/procedimientos-almacenados-y-plpgsql/

IBM. (28 de 06 de 2023). *IBM documentacion*. Obtenido de https://www.ibm.com/docs/es/db2/11.1?topic=elements-functions

Imasgal. (04 de 07 de 2023). *https://imasgal.com*. Obtenido de https://imasgal.com: https://imasgal.com/uso-de-disparadores-en-postgresql/#:~:text=Un%20disparador%20(trigger%20en%20ingl%C3%A9s,sentencias%20INSERT%2C%20UPDATE%20o%20DELETE.

Mediacloud. (29 de 05 de 2023). *https://blog.mdcloud.es*. Obtenido de https://blog.mdcloud.es/crear-vistas-en-mysql-que-es-y-ventajas-de-hacerlo/#:~:text=Las%20vistas%20en%20MySQL%20(VIEWS)%20son%20tablas%20virtuales.&text=Solo%20muestran%20los%20datos%20que,a%20la%20fuente%20en%20s%C3%AD.

Microsoft. (29 de 05 de 2023). *https://learn.microsoft.com/*. Obtenido de https://learn.microsoft.com/es-es/sql/relational-databases/views/views?view=sql-server-ver16

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

PostgreSQL. (28 de 06 de 2023). *https://www.postgresql.org/*. Obtenido de https://www.postgresql.org/: https://www.postgresql.org/docs/current/sql-createfunction.html

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

SQLShack. (28 de 06 de 2023). *www.sqlshack.com*. Obtenido de www.sqlshack.com: https://www.sqlshack.com/functions-stored-procedures-comparisons-sql-server/

Todopostgresql. (04 de 07 de 2023). *https://www.todopostgresql.com*. Obtenido de https://www.todopostgresql.com: https://www.todopostgresql.com/postgresql-create-trigger-disparador-postgresql/

TUTORIAL, P. (28 de 06 de 2023). *www.postgresqltutorial.com*. Obtenido de www.postgresqltutorial.com: https://www.postgresqltutorial.com/postgresql-string-functions/

*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