**Bases de datos II**

**Actividad 3**

**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](#_Toc139514433)

[2 Introducción 3](#_Toc139514434)

[3 Objetivos 4](#_Toc139514435)

[3.1 Objetivo General: 4](#_Toc139514436)

[3.2 Objetivos específicos 4](#_Toc139514437)

[4 Justificación 5](#_Toc139514438)

[5 Desarrollo 6](#_Toc139514439)

[5.1 Qué es y cuál es el propósito de una Función en SQL 6](#_Toc139514440)

[5.2 Qué es y cual es el propósito de un Procedimiento almacenado en SQL 7](#_Toc139514441)

[5.3 Cuáles son las diferencias más representativas entre Función y Procedimiento almacenado. 7](#_Toc139514442)

[5.4 Investigue cuales son las funciones predefinidas en PostgreSQL que nos permiten realizar operaciones sobre: 8](#_Toc139514443)

[5.5 Qué es y cuál es el objetivo de una Función personalizada o definida por el usuario 16](#_Toc139514444)

[5.6 Cual es la sintaxis general para crear e invocar una Función definida por el usuario. 16](#_Toc139514445)

[5.7 Crear una función que reciba como parámetros de entrada la cc y la clave de un deudor, entonces la función debe retornar Verdadero si existe un usuario con esa misma CC y Clave registrados en esa tabla, o retornar FALSO en caso contrario. 17](#_Toc139514446)

[5.8 Invocar y probar la función con datos que permitan mostrar resultado Falso y otros que permitan retornar resultado Verdadero 18](#_Toc139514447)

[5.9 Crear una función que reciba como parámetros la cédula de un deudor y devuelva un solo dato con el nombre del deudor, valor del último credito, fecha y estado del último crédito que tiene en el sistema. El resultado debe ser una cadena de texto en el siguiente formato: por ejemplo. NOMBRE: FULANITO DE TAL, FECHA: 12-04-2022, VALOR: $100000, ESTADO: ACTIVO 19](#_Toc139514448)

[5.10 Invocar y probar la función 21](#_Toc139514449)

[5.11 Crear una que reciba 2 números y devuelva la sumatoria acumulativa entre ellos. 21](#_Toc139514450)

[5.12 Crear un procedimiento almacenado con las siguientes características: 24](#_Toc139514451)

[5.13 Invocar y probar el procedimiento almacenado. 25](#_Toc139514452)

[6 Bibliografía 27](#_Toc139514453)

# Introducción

En el mundo de la gestión de bases de datos, SQL (Structured Query Language) el cual se ha convertido en un estándar ampliamente utilizado para interactuar con sistemas de gestión de bases de datos relacionales. PostgreSQL, es uno de los sistemas de gestión de bases de datos más populares o más utilizados y potentes, ofrece un conjunto muy completo de características y funcionalidades para aprovechar al máximo en SQL.

Las funciones SQL y los procedimientos almacenados son componentes claves en PostgreSQL que permiten la reutilización de código, la simplificación de tareas repetitivas y la mejora del rendimiento. En este trabajo, estaremos aplicando varias situaciones en las que las funciones SQL y los procedimientos almacenados pueden ser utilizados de manera efectiva, dando soluciones a cada uno de los puntos o situaciones problémicas planteadas por el tutor de la asignatura Base de Datos II.

# Objetivos

## **Objetivo General**:

Analizar y aplicar las funciones SQL y los procedimientos almacenados en PostgreSQL para resolver diferentes situaciones problémicas o desafíos que enfrenta en el mundo real el profesional de Ingeniería de Software.

## Objetivos específicos

Comprender los conceptos fundamentales de las funciones SQL y los procedimientos almacenados en PostgreSQL.

Explorar la sintaxis y las características de las funciones SQL y los procedimientos almacenados en PostgreSQL.

Resolver una variedad de situaciones problémicas mediante el diseño y la implementación de funciones SQL y procedimientos almacenados adecuados en PostgreSQL.

Evaluar las ventajas, desventajas y diferencias al utilizar funciones SQL y procedimientos almacenados en PostgreSQL.

Proponer alternativas de solución a los diferentes ejercicios propuestos que son de uso práctico que demuestren la utilidad y aplicabilidad de las funciones SQL y los procedimientos almacenados en PostgreSQL.

# Justificación

La importancia de las funciones SQL y los procedimientos almacenados en el desarrollo de aplicaciones de bases de datos es muy importante. Al utilizar estas características avanzadas de PostgreSQL, es posible mejorar la eficiencia de las operaciones y facilitar la resolución de problemas con cierto grado de complejidad.

En este trabajo se hace necesario comprender y aplicar las funciones SQL y los procedimientos almacenados en PostgreSQL para aprovechar al máximo el potencial de esta potente herramienta de gestión de bases de datos. Al adquirir conocimientos y habilidades en este campo de las bases de datos, los profesionales de la ingeniería de Software, disciplinas afines y desarrolladores en general, podrán mejorar la eficiencia de sus proyectos, reducir errores y tomar decisiones fundamentadas para el diseño de soluciones basadas en bases de datos.

Este trabajo también busca proporcionar ejemplos prácticos y casos de uso reales que ilustren cómo las funciones SQL y los procedimientos almacenados pueden abordar problemas específicos en situaciones del mundo real.

En resumen, este trabajo sobre funciones SQL y procedimientos almacenados en PostgreSQL tiene como objetivo proporcionar un análisis detallado, ejemplos prácticos para comprobar la importancia y utilidad de estas características en el desarrollo y administración de bases de datos relacionales.

# Desarrollo

Para el desarrollo de esta actividad académica, usted como estudiante del programa de Ingeniería de Software en **modalidad a distancia** de la Universidad de Cartagena debe desarrollar, presentar y sustentar en tutoría los siguientes puntos, 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 Funciones, Procedimientos Almacenados y su aplicación en los motores de bases de datos relacionales OracleSQL, PostgreSQL y SQLServer.

## Qué es y cuál es el propósito de una Función en SQL

Una función SQL no es mas que un conjunto de códigos en los cuales se ingresan unos valores de entrada denominados argumentos y al ejecutar la función esta entrega como salida un resultado. Los argumentos pueden ser de cualquier tipo de dato y la salida el dato deseado.

Las funciones se pueden clasificar como funciones incorporadas y funciones definidas por el usuario.

***Las funciones incorporadas*** son aquella que proporciona el gestor de bases de datos.

Las funciones incorporadas pueden ser agregadas por ejemplo SUM(), funciones de operador por ejemplo (-), funciones de conversión, funciones escalares y las funciones de tabla.

Las funciones definidas por el usuario son creadas por el usuario a su antojo y dependiendo de las necesidades del mismo, estas son creadas mediante una sentencia de definición de datos SQL. Existen también otras formas de clasificar las funciones (IBM, 2023).

El propósito de una función es realizar una operación específica en una base de datos, normalmente se usan cuando son operaciones repetitivas por el usuario, por ejemplo si yo deseo obtener el número de estudiantes de determinado grupo de estudiantes de un colegio puedo crear una función que reciba como argumento el código del grupo, la función debe contar en la tabla donde estén registrados los estudiantes correspondiente al código de grupo ingresado y retornar un entero que corresponde al número de estudiantes del grupo. El usuario puede usar esta función para consultar otros grupos lo cual es una ventaja de las funciones el ser reutilizable.

## Qué es y cual es el propósito de un Procedimiento almacenado en SQL

Un procedimiento almacenado es muy similar a las funciones pero no necesariamente retornan valor, Un procedimiento almacenado es un conjunto de instrucciones SQL que se guardan en la base datos y que podemos invocar posteriormente para uso, los procedimientos almacenados también pueden recibir argumentos en PostgreSQL se puede escribir en múltiples lenguajes de programación, PL/pgSQL, PL/Perl, PL/Tcl y PL/Python, En PostgreSQL viene por defecto PL/pgSQL (Emc2Net, 2023).

El propósito de un procedimiento almacenado en SQL es muy parecido al de una función, se introduce en un bloque de código las instrucciones SQL a realizar y luego se llaman para ejecutarlas, también se pueden reutilizar, lo cual mejora el rendimiento en la base de datos.

## Cuáles son las diferencias más representativas entre Función y Procedimiento almacenado.

Existen diferencias importantes entre funciones y procedimientos almacenados, entre estas tenemos:

Las funciones necesariamente retornan un valor mientras que los procedimiento no necesariamente devuelven un valor.

Las funciones escalares solo pueden devolver una variable mientras que en un procedimiento almacenado puede devolver varias variables.

Un procedimiento puede invocar funciones y otros procedimientos almacenados, mientras que una función no puede invocar procedimientos almacenados

Las funciones a menudo se invocan en consultas SQL debido a que retornan un valor mientras que los procedimientos no pueden ser invocados en las consultas.

Las funciones las utilizamos para efectuar cálculos tales como suma, promedio, conteo entre otros, mientras que los procedimientos se utilizan para ejecutar ciertas instrucciones como Insertar, actualizar, o eliminar datos.

Los procedimientos pueden presentar transacciones que permitan modificar algunas tablas o datos de las mismas, lo cual trae como consecuencia que los procedimientos garantizan la integridad de la base de datos y permite controlar los errores posibles. Las funciones no están diseñadas para realizar transacciones directamente. Generalmente se utilizan para realizar operaciones más sencillas. (SQLShack, 2023)

## Investigue cuales son las funciones predefinidas en PostgreSQL que nos permiten realizar operaciones sobre:

Todos los gestores de bases de datos traen funciones predefinidas que son de mucha utilidad para los profesionales que trabajan en el diseño, desarrollo, implementación y mantenimiento de las mismas. Es importante conocer las funciones predefinidas sobre todo las más utilizadas a diario, en este punto se acudió a la página web de la documentación de PostgeSQL (https://www.postgresql.org, 2023) y otras fuentes que proporcionaron la información necesaria y suficiente para aprender sobre las funciones predefinidas.

* 1. Cadenas

|  |  |  |
| --- | --- | --- |
| NOMBRE FUNCIÓN | ACCIÓN | EJEMPLO |
| ASCII | Devuelve el valor del código ASCII de un carácter. | ASCII(‘J’) |
| CHR | Convierte un código ASCII en un carácter | CHR(70) |
| CONCAT | Concatena dos o más cadenas en una sola cadena | CONCAT(‘Jonny’, ‘Luna’, ‘Guerrero’) |
| CONCAT\_WS | Concatena cadenas con un separador | CONCAT\_WS(‘ - ‘, ‘Jonny’, ‘Luna’)  Retorna **Jonny-Luna** |
| FORMAT | Da formatos a argumentos basados en una cadena de formato | FORMAT('Hola, %1$L','mundo');  Retorna **Hola, ‘mundo’** |
| INITCAP | Convierte la primera letra de cada palabra en mayúscula, como en los títulos | INITCAP(‘jonny luna’)  Retorna ‘Jonny Luna’ |
| LEFT | Devuelve los caracteres empezando desde la izquierda en base al número que se le especifique. | LEFT('Jonny',3);  Retorna ‘Jon’ |
| LENGTH | Retorna el número de caracteres de una cadena | LENGTH(‘Jonny’)  Retorna 5 |
| LOWER | Convierte una cadena en minúscula | LOWER(‘JONNY’)  Retorna jonny |
| LPAD | Rellena a la izquierda una cadena con el número de caracteres que se le especifica | LPAD(‘Jonny’, 8, ‘\*’)  Retorna \*\*\*Jonny |
| LTRIM | Elimina caracteres empezando por la izquierda otros caracteres. | LTRIM(‘Jonny’,’J’)  Retorna **onny** |
| REPLACE | Remplaza una subcadena en una cadena | SELECT REPLACE('Hola Mundo', 'Mundo', . 'Amigo'); |
| RIGHT | devuelve los caracteres empezando desde la derecha en base al número que se le especifique. | SELECT RIGHT (‘jlunag’,3); |
| SUBSTRING | Extrae una subcadena de una cadena dada. | SELECT SUBSTRING('Jonny Luna', 6, 5); |
| TRIM | Elimina los espacios en blanco al inicio y al final de una cadena. | SELECT TRIM('Jonny Luna'); |
| UPPER | Convierte una cadena a mayúsculas. | SELECT UPPER('Jonny Luna'); |

* 1. Cifrado y descifrado de datos.

|  |  |  |
| --- | --- | --- |
| NOMBRE FUNCIÓN | ACCIÓN | EJEMPLO |
| MD5 | Se utiliza para retornar el hash MD5 de una cadena en hexadecimal | MD5('Jonny'); |
| ENCRYPT | Encripta una cadena utilizando el algoritmo de cifrado Blowfish. | SELECT ENCRYPT('secreto', gen\_salt('bf')); |
| DECRYPT | Descifra una cadena encriptada con el algoritmo Blowfish. | SELECT DECRYPT('ciphertext', 'key'); |
| SHA256 | Calcula el hash SHA-256 de una cadena. | SELECT SHA256('Jonny Luna'); |
| AES\_ENCRYPT | Encripta una cadena utilizando el algoritmo AES. | SELECT AES\_ENCRYPT('secreto', 'clave'); |
| BASE64\_ENCODE | Codifica una cadena en formato Base64. | SELECT BASE64\_ENCODE('Jonny Luna'); |
| BASE64\_DECODE | Decodifica una cadena codificada en formato Base64. | SELECT BASE64\_DECODE ('SGVsbG8gV29ybGQ='); |
| ROT13 | Realiza una rotación de 13 caracteres en una cadena. | SELECT ROT13('Jonny Luna'); |
| PG\_CLIENT\_ENCODING | Devuelve la codificación del cliente actual. | SELECT PG\_CLIENT\_ENCODING(); |
| PG\_COLUMN\_SIZE | Devuelve el tamaño de almacenamiento de una columna. | SELECT PG\_COLUMN\_SIZE('public', 'table\_name', 'column\_name'); |

* 1. Fechas

|  |  |  |
| --- | --- | --- |
| NOMBRE FUNCIÓN | ACCIÓN | EJEMPLO |
| CURRENT\_DATE | Devuelve la fecha actual. | SELECT CURRENT\_DATE; |
| EXTRACT | Extrae parte de una fecha o intervalo. | SELECT EXTRACT(YEAR FROM TIMESTAMP '2023-01-01 10:30:00'); |
| DATE\_TRUNC | Trunca una fecha a la unidad de tiempo especificada. | SELECT DATE\_TRUNC('month', TIMESTAMP '2023-07-15 09:45:00'); |
| AGE | Calcula la diferencia de años entre dos fechas. | SELECT AGE('2023-07-05', '1990-12-31'); |
| DATE\_PART | Devuelve una parte específica de una fecha. | SELECT DATE\_PART('day', TIMESTAMP '2023-07-05 12:30:00'); |
| DATE\_ADD | Agrega una cantidad específica de tiempo a una fecha. | SELECT DATE\_ADD('2023-07-01', INTERVAL '3 days'); |
| DATE\_DIFF | Calcula la diferencia entre dos fechas en una unidad de tiempo específica. | SELECT DATE\_DIFF('day', '2023-07-01', '2023-07-10'); |
| NOW | Devuelve la fecha y hora actual. | SELECT NOW(); |
| TIMEZONE | Convierte una fecha y hora a una zona horaria específica. | SELECT TIMEZONE('America/New\_York', TIMESTAMP '2023-07-01 12:00:00'); |
| INTERVAL | Crea un intervalo de tiempo. | SELECT INTERVAL '1 year 2 months 3 days 4 hours 5 minutes'; |

* 1. Matemáticas

|  |  |  |
| --- | --- | --- |
| NOMBRE FUNCIÓN | ACCIÓN | EJEMPLO |
| ABS | Devuelve el valor absoluto de un número. | SELECT ABS(-10); |
| SQRT | Calcula la raíz cuadrada de un número. | SELECT SQRT(25); |
| POWER | Calcula la potencia de un número dado. | SELECT POWER(2, 3); |
| CEIL | Redondea un número hacia arriba al entero más cercano. | SELECT CEIL(4.5); |
| FLOOR | Redondea un número hacia abajo al entero más cercano. | SELECT FLOOR(4.5); |
| TRUNC | Trunca un número a una cantidad específica de decimales. | SELECT TRUNC(3.14159, 2); |
| MOD | Calcula el resto de una división. | SELECT MOD(10, 3); |
| RANDOM | Genera un número decimal aleatorio entre 0 y 1. | SELECT RANDOM(); |
| ROUND | Redondea un número al entero más cercano. | SELECT ROUND(3.14159); |
| SIGN | Devuelve el signo de un número  (-1, 0 o 1). | SELECT SIGN(-10);1 |

* 1. Conversiones entre tipos de datos

|  |  |  |
| --- | --- | --- |
| NOMBRE FUNCIÓN | ACCIÓN | EJEMPLO |
| TO\_CHAR | Convierte un valor a una cadena de caracteres. | SELECT TO\_CHAR(123.456, '999.99'); |
| TO\_DATE | Convierte una cadena a un valor de fecha. | SELECT TO\_DATE('2023-07-01', 'YYYY-MM-DD'); |
| CAST | Convierte un valor a otro tipo de datos. | SELECT CAST('10' AS INTEGER); |
| TRY\_CAST | Intenta convertir un valor a otro tipo de datos y devuelve NULL si la conversión no es posible. | SELECT TRY\_CAST('10.5' AS INTEGER); |
| TO\_TIMESTAMP | Convierte una cadena a un valor de fecha y hora. | SELECT TO\_TIMESTAMP('2023-07-01 12:00:00', 'YYYY-MM-DD HH24:MI:SS'); |
| TO\_BINARY | Convierte un valor a su representación binaria. | SELECT TO\_BINARY(10); |
| TO\_HEX | Convierte un valor a su representación hexadecimal. | SELECT TO\_HEX(255); |

* 1. Manejo y obtener información de errores

|  |  |  |
| --- | --- | --- |
| NOMBRE FUNCIÓN | ACCIÓN | EJEMPLO |
| RAISE | Genera un error y lanza una excepción. | RAISE EXCEPTION 'Error personalizado'; |
| SQLSTATE | Devuelve el código SQLSTATE del último error. | SELECT SQLSTATE; |
| GET STACKED DIAGNOSTICS | Obtiene información sobre el estado de error actual. | GET STACKED DIAGNOSTICS v\_message = MESSAGE\_TEXT, v\_detail = PG\_EXCEPTION\_DETAIL; |
| COALESCE | Devuelve el primer valor no nulo de una lista de valores. | SELECT COALESCE(NULL, 'Valor Predeterminado'); |
| NULLIF | Compara dos expresiones y devuelve NULL si son iguales, o el primer valor si son diferentes. | SELECT NULLIF(10, 10); |
| ERROR\_MESSAGE | Devuelve el mensaje de error del último error. | SELECT ERROR\_MESSAGE(); |
| SQLERRM | Devuelve el mensaje de error del último error en PL/pgSQL. | RAISE EXCEPTION 'Error personalizado'; SQLERRM; |
| GET DIAGNOSTICS | Obtiene información sobre el estado de error actual en PL/pgSQL. | GET DIAGNOSTICS v\_message = PG\_EXCEPTION\_MESSAGE; |

* 1. Obtener información sobre el Motor de BD

|  |  |  |
| --- | --- | --- |
| NOMBRE FUNCIÓN | ACCIÓN | EJEMPLO |
| VERSION | Devuelve la versión de PostgreSQL. | SELECT VERSION(); |
| CURRENT\_USER | Devuelve el nombre del usuario actual. | SELECT CURRENT\_USER; |
| SESSION\_USER | Devuelve el nombre del usuario de sesión actual. | SELECT SESSION\_USER; |
| CURRENT\_DATABASE | Devuelve el nombre de la base de datos actual. | SELECT CURRENT\_DATABASE(); |
| CONNECTION\_ID | Devuelve el ID de conexión actual. | SELECT CONNECTION\_ID(); |
| PG\_SIZE\_PRETTY | Convierte un tamaño en bytes en una representación legible. | SELECT PG\_SIZE\_PRETTY(102410241024); |
| PG\_STAT\_GET\_BACKEND\_START | Devuelve la hora de inicio del proceso del servidor backend. | SELECT PG\_STAT\_GET\_BACKEND\_START(1); |
| IS\_SUPERUSER | Verifica si el usuario actual es un superusuario. | SELECT IS\_SUPERUSER(); |

* 1. entre otras

|  |  |  |
| --- | --- | --- |
| NOMBRE FUNCIÓN | ACCIÓN | EJEMPLO |
| GREATEST | Devuelve el valor más grande de una lista de valores. | SELECT GREATEST(10, 20, 15, 5); |
| LEAST | Devuelve el valor más pequeño de una lista de valores. | SELECT LEAST(10, 20, 15, 5); |
| REPEAT | Repite una cadena un número especificado de veces. | SELECT REPEAT('Hola ', 3); |
| RPAD | Rellena una cadena con un carácter específico en el lado derecho. | SELECT RPAD('Hola', 8, '\*'); |
| STRPOS | Devuelve la posición de la primera aparición de una subcadena en una cadena. | SELECT STRPOS('Hello World', 'World'); |
| SPLIT\_PART | Divide una cadena en partes basadas en un delimitador y devuelve la parte especificada. | SELECT SPLIT\_PART('Hola,Mundo,Amigo', ',', 2); |
| ASCII | Devuelve el código ASCII de un carácter. | SELECT ASCII('A'); |

(TUTORIAL, 2023)

## Qué es y cuál es el objetivo de una Función personalizada o definida por el usuario

El objetivo de una función personalizada es crear funciones que no estén definidas en las funciones predefinidas además que tenga una función específica según lo requiera el usuario.

## Cual es la sintaxis general para crear e invocar una Función definida por el usuario.

Sentencia SQL de la sintaxis general tomada de (PostgreSQL, 2023)

CREATE [ OR REPLACE ] FUNCTION

***name*** ( [ [ ***argmode*** ] [ ***argname*** ] ***argtype*** [ { DEFAULT | = } ***default\_expr*** ] [, ...] ] )

[ RETURNS ***rettype***

| RETURNS TABLE ( ***column\_name*** ***column\_type*** [, ...] ) ]

{ LANGUAGE ***lang\_name***

| TRANSFORM { FOR TYPE ***type\_name*** } [, ... ]

| WINDOW

| { IMMUTABLE | STABLE | VOLATILE }

| [ NOT ] LEAKPROOF

| { CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }

| { [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER }

| PARALLEL { UNSAFE | RESTRICTED | SAFE }

| COST ***execution\_cost***

| ROWS ***result\_rows***

| SUPPORT ***support\_function***

| SET ***configuration\_parameter*** { TO ***value*** | = ***value*** | FROM CURRENT }

| AS '***definition***'

| AS '***obj\_file***', '***link\_symbol***'

| ***sql\_body***

} ...

De otra forma más sencilla

CREATE FUNCTION nombre\_funcion ([parámetros])

[RETURNS tipo\_de\_dato]

[LANGUAGE lenguaje]

AS $$

-- Cuerpo de la función: lógica y operaciones

-- ...

-- Puede contener múltiples sentencias SQL

$$;

Usar la base de datos prueba\_transacciones\_grupoXYZ creada en la actividad de la Unidad 2 (punto 5 al punto 11), entonces deben hacer los siguientes:

## Crear una función que reciba como parámetros de entrada la cc y la clave de un deudor, entonces la función debe retornar Verdadero si existe un usuario con esa misma CC y Clave registrados en esa tabla, o retornar FALSO en caso contrario.

Sentencia SQL

CREATE OR REPLACE FUNCTION verificar\_deudor(d\_cc INTEGER, d\_clave TEXT)

RETURNS BOOLEAN AS $$

DECLARE

existe BOOLEAN;

BEGIN

SELECT EXISTS(

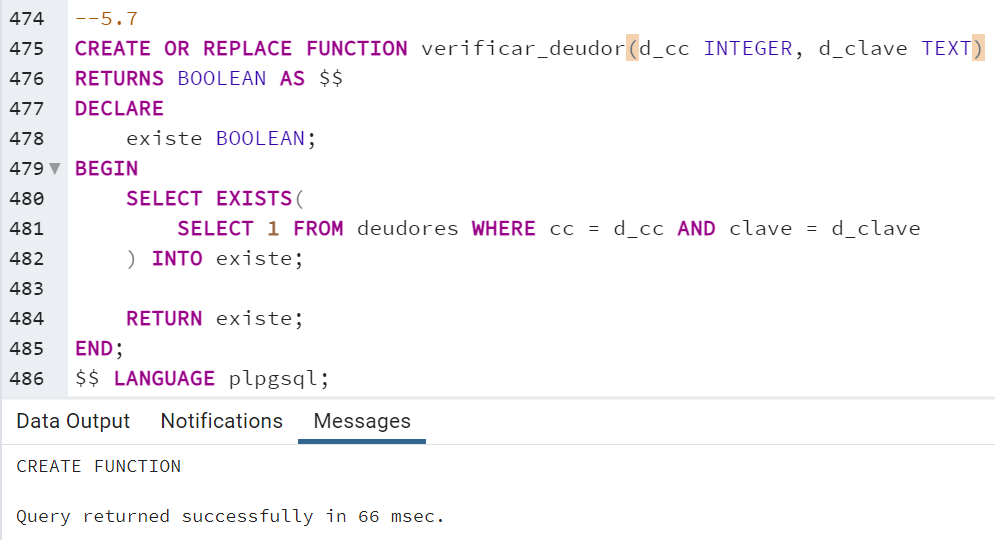
SELECT 1 FROM deudores WHERE cc = d\_cc AND clave = d\_clave

) INTO existe;

RETURN existe;

END;

$$ LANGUAGE plpgsql;

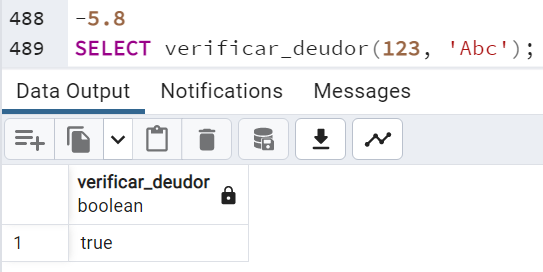


## Invocar y probar la función con datos que permitan mostrar resultado Falso y otros que permitan retornar resultado Verdadero

Para un deudor que existe en la tabla tenemos:

Sentencia SQL

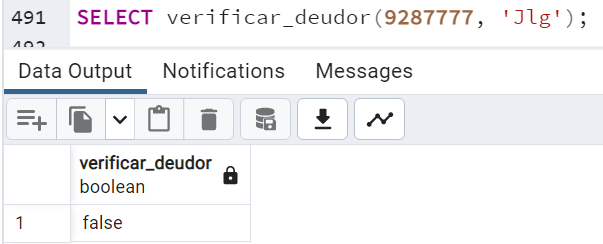
SELECT verificar\_deudor(123, 'Abc');



Para un deudor que NO existe en la tabla tenemos:

Sentencia SQL

SELECT verificar\_deudor(9287777, 'Jlg');



## Crear una función que reciba como parámetros la cédula de un deudor y devuelva un solo dato con el nombre del deudor, valor del último credito, fecha y estado del último crédito que tiene en el sistema. El resultado debe ser una cadena de texto en el siguiente formato: por ejemplo. NOMBRE: FULANITO DE TAL, FECHA: 12-04-2022, VALOR: $100000, ESTADO: ACTIVO

Sentencia SQL

CREATE OR REPLACE FUNCTION obtener\_informacion\_deudor(d\_cc INTEGER)

RETURNS TEXT AS $$

DECLARE

d\_nombre TEXT;

ultimo\_credito RECORD;

resultado TEXT;

BEGIN

-- Obtener el nombre del deudor

SELECT nombre INTO d\_nombre

FROM deudores

WHERE cc = d\_cc;

-- Obtener el último crédito del deudor

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

INTO ultimo\_credito

FROM creditos c

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

WHERE d.cc = d\_cc

ORDER BY c.fecha DESC

LIMIT 1;

-- Construir el resultado en el formato requerido

resultado := 'NOMBRE: ' || d\_nombre || ',

FECHA: ' || to\_char(ultimo\_credito.fecha, 'DD-MM-YYYY') || ',

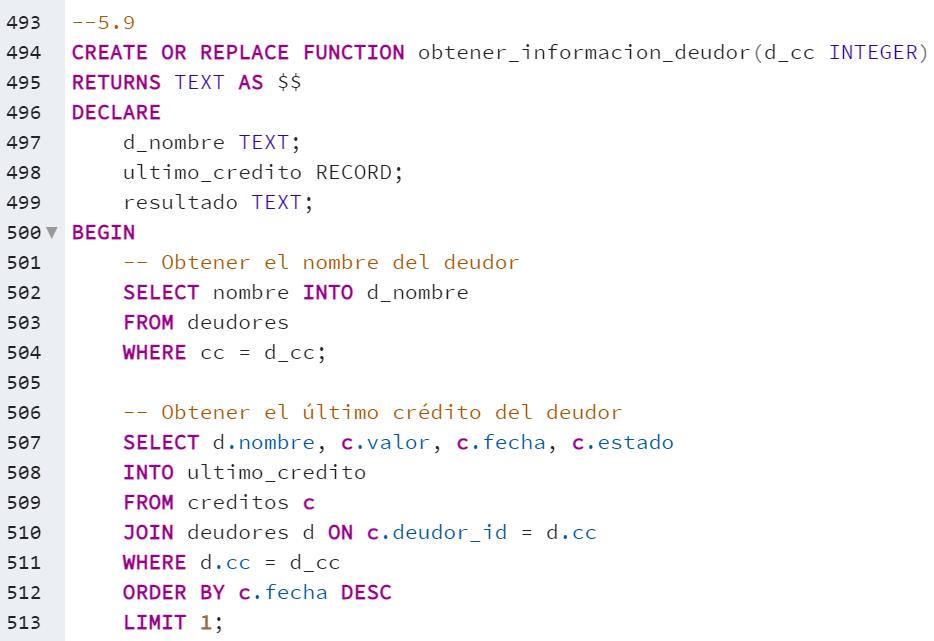
VALOR: $' || ultimo\_credito.valor || ',

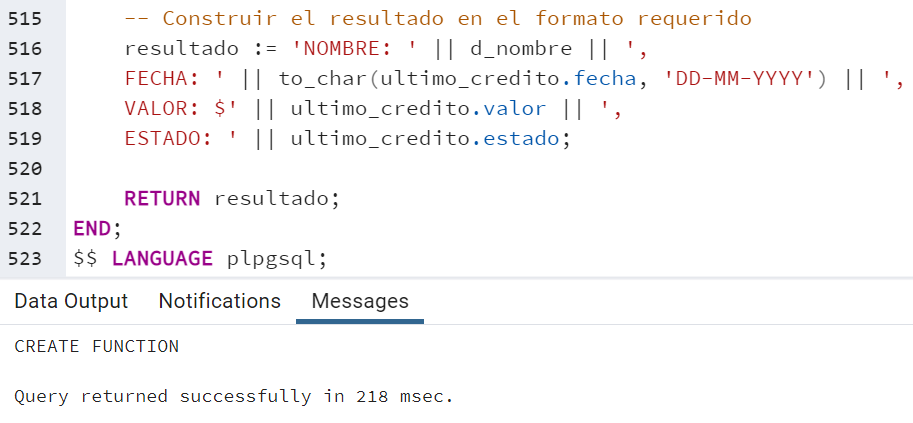
ESTADO: ' || ultimo\_credito.estado;

RETURN resultado;

END;

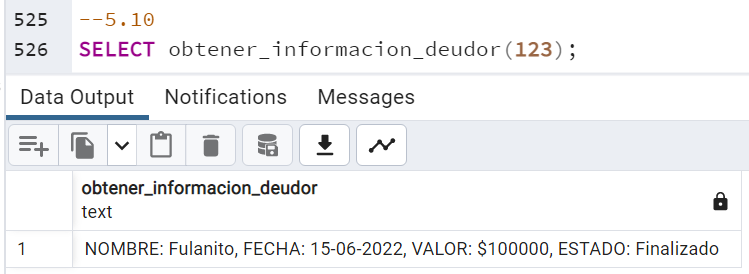
$$ LANGUAGE plpgsql;





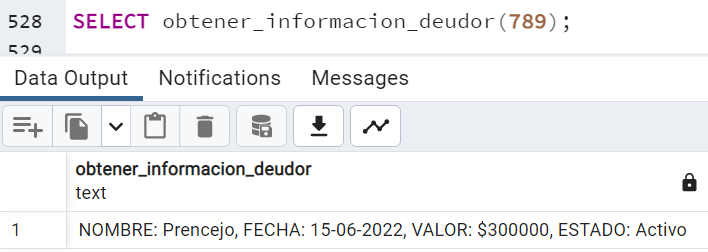
## Invocar y probar la función

SELECT obtener\_informacion\_deudor(123);



Se Prueba con otro deudor

SELECT obtener\_informacion\_deudor(789);



## Crear una que reciba 2 números y devuelva la sumatoria acumulativa entre ellos.

La siguiente función suma todos los números entre dos números dados.

Sentencia SQL

CREATE OR REPLACE FUNCTION sumatoria\_acumulativa(n\_1 INTEGER, n\_2 INTEGER)

RETURNS INTEGER AS $$

DECLARE

v\_suma INTEGER := 0;

v\_min INTEGER;

v\_max INTEGER;

BEGIN

IF n\_1 < n\_2 THEN

v\_min := n\_1;

v\_max := n\_2;

ELSE

v\_min := n\_2;

v\_max := n\_1;

END IF;

FOR i IN v\_min..v\_max LOOP

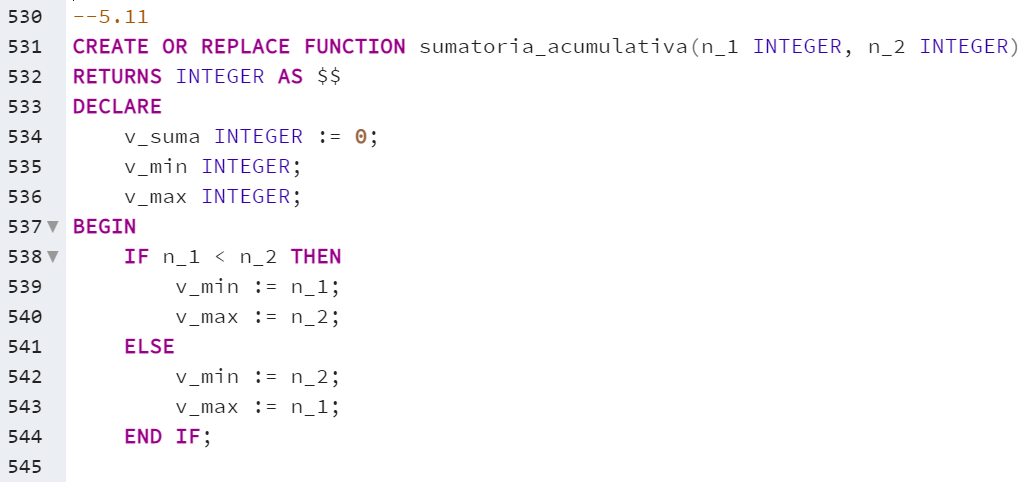
v\_suma := v\_suma + i;

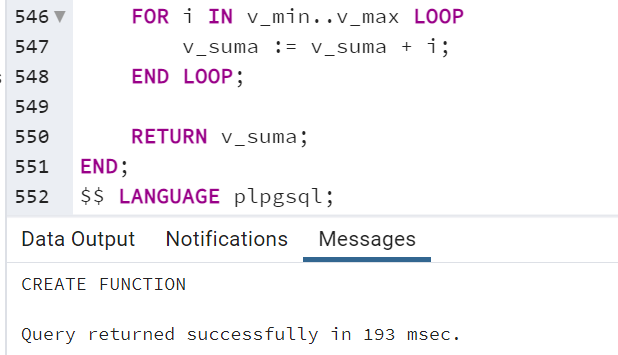
END LOOP;

RETURN v\_suma;

END;

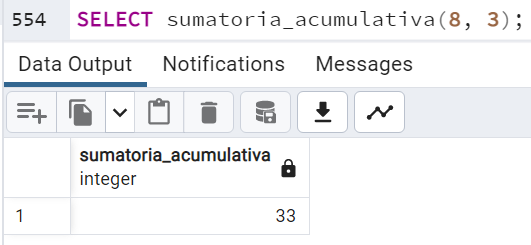
$$ LANGUAGE plpgsql;



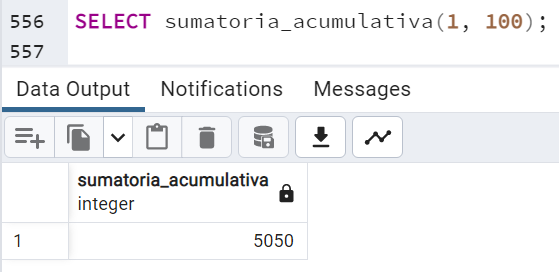


Al darle parámetros nos queda:

SELECT sumatoria\_acumulativa(8, 3);



SELECT sumatoria\_acumulativa(1, 100);



## Crear un procedimiento almacenado con las siguientes características:

* 1. debe tener un parámetro de **entrada** que reciba el código de un deudor
  2. debe tener un parámetro de **salida** que devuelva la cantidad (cuantos) de créditos Activos que tiene ese deudor

Sentencia SQL

CREATE OR REPLACE PROCEDURE creditos\_activos(

d\_cc INTEGER,

OUT creditos\_activos INTEGER

)

LANGUAGE plpgsql

AS $$

BEGIN

SELECT COUNT(\*) INTO creditos\_activos

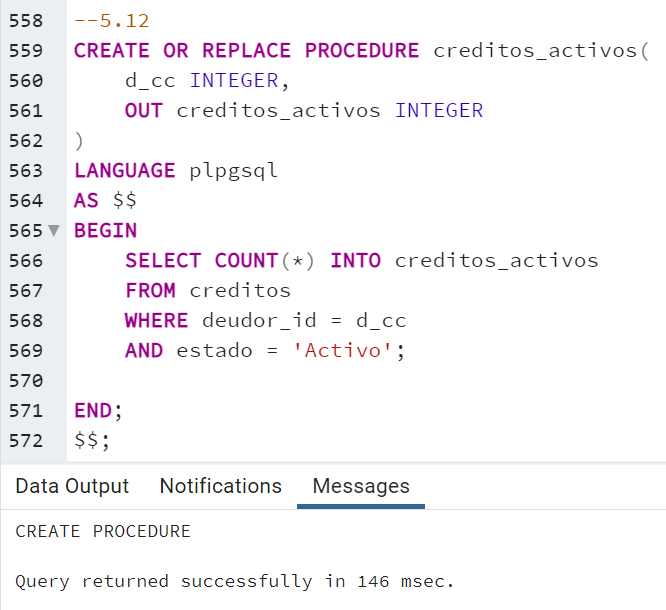
FROM creditos

WHERE deudor\_id = d\_cc

AND estado = 'Activo';

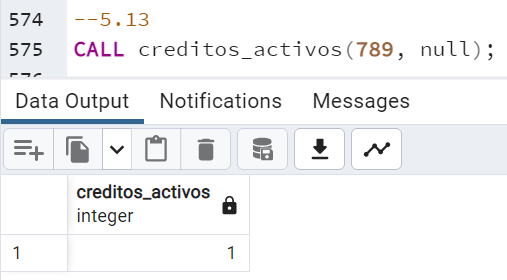
END;

$$;

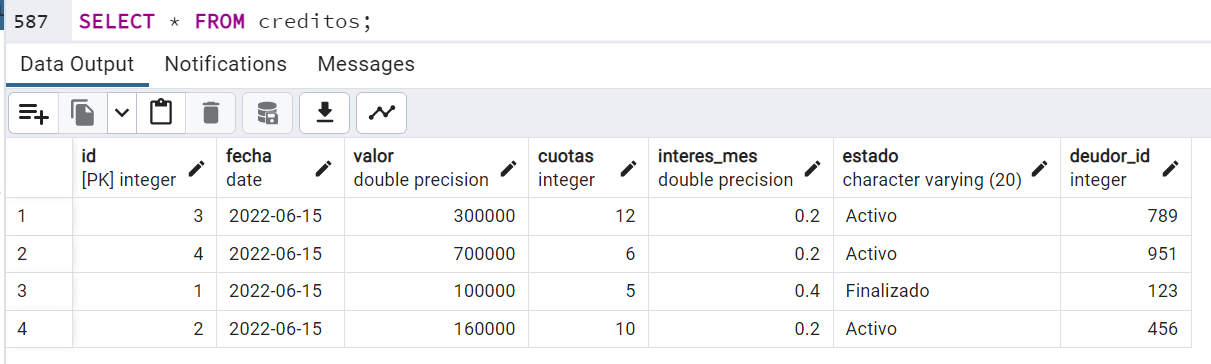


## Invocar y probar el procedimiento almacenado.

Sentencia SQL

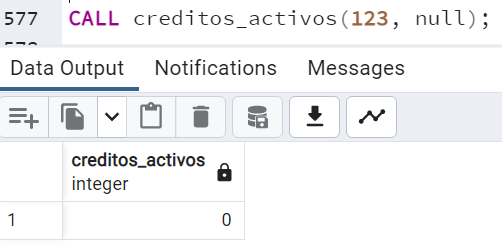


Se observa la tabla créditos para verificar.



Se prueba con el deudor de cc = 123

CALL creditos\_activos(123, null);



.

# 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/

*https://www.postgresql.org*. (02 de 07 de 2023). Obtenido de https://www.postgresql.org: https://www.postgresql.org/docs/

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

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

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/

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