

FUNCIONES Y TRIGGERS:

ORACLE

Autor: Alberto Alejandro Morales Caballero

Fecha: 9 de abril del 2025

Objetivo: Crear funciones y procedimientos en Oracle.

FUNCIÓN QUE GENERA CADENA DE TEXTO ALEATORIA . 'U' INDICA QUE SOLO SERÁN LETRAS MAYÚSCULAS CON UNA LONGITUD DE 4 CARACTERES

```
SELECT DBMS_RANDOM.string('U',4) FROM DUAL;
```

```
SQL> SELECT DBMS_RANDOM.string('U',4) FROM DUAL;
```

```
DBMS_RANDOM.STRING('U',4)
```

```
-----
```

```
-----
```

```
LVNT
```

FUNCIÓN QUE GENERA CADENA DE TEXTO ALEATORIA . 'L' INDICA QUE SOLO SERÁN LETRAS MINÚSCULAS Y TENDRÁ UNA LONGITUD DE 5 CARACTERES

```
SELECT DBMS_RANDOM.string('L',5) FROM DUAL;
```

```
SQL> SELECT DBMS_RANDOM.string('L',5) FROM DUAL;
```

```
DBMS_RANDOM.STRING('L',5)
```

```
-----
```

```
-----
```

```
shbrb
```

FUNCIÓN QUE DEVUELVE UN NÚMERO ALEATORIO DECIMAL ENTRE EL 0-9 PARA DESPUÉS TRUNCARLO A LA PARTE ENTERA. SE USA DUAL PARA NO HACER REFERENCIA A UNA TABLA REAL

```
SELECT TRUNC(DBMS_RANDOM.value(0,9)) FROM DUAL;
```

```
SQL> SELECT TRUNC(DBMS_RANDOM.value(0,9)) FROM DUAL;
```

```
TRUNC(DBMS_RANDOM.VALUE(0,9))
```

```
-----
```

FUNCIÓN QUE DEVUELVE UN CARACTER ALEATORIO SOLO EN MAYUSCULAS DE 1 CARÁCTER

```
SELECT DBMS_RANDOM.string('U',1) FROM DUAL;
```

```
SQL> SELECT DBMS_RANDOM.string('U',1) FROM DUAL;
```

```
DBMS_RANDOM.STRING('U',1)
```

```
-----  
-----
```

```
W
```

Este comando seleccionará un carácter aleatorio de la cadena "(+\${#()-!}{[])" usando un índice aleatorio generado por DBMS_RANDOM.VALUE(1, 14). EL ULTIMO PARAMETRO (1) INDICA QUE SOLO SE MOSTRará UN CARACTER.

```
SELECT SUBSTR('(+${#()-!}{[])',DBMS_RANDOM.value(1,14),1) FROM DUAL;
```

```
SQL> SELECT SUBSTR('(+${#()-!}{[])',DBMS_RANDOM.value  
(1,14),1) FROM DUAL;
```

```
SUBS
```

```
----
```

```
}
```

Este comando devolverá una cadena aleatoria de 100 caracteres que puede contener letras

```
SELECT DBMS_RANDOM.string('A',100) FROM DUAL;
```

```
SQL> SELECT DBMS_RANDOM.string('A',100) FROM DUAL;
```

```
DBMS_RANDOM.STRING('A',100)
```

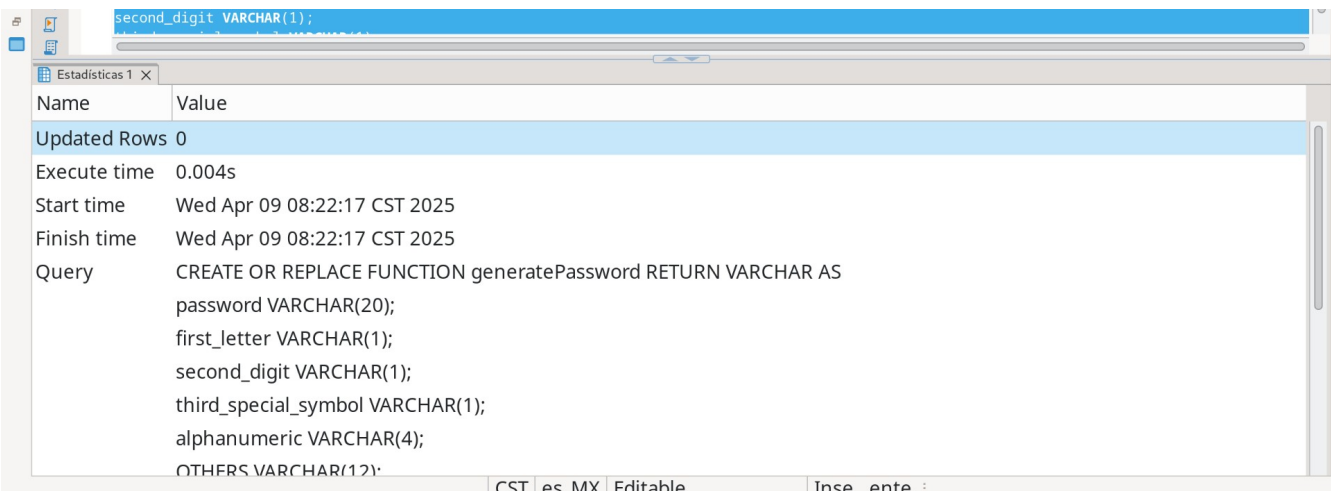
```
-----  
-----
```

```
nBpkVhJsCFGjyvYvhOQpCZnKZcPBDHdYxcVoEmMqIsFpWLfwSzsxj  
gvMeFLYomQzicYDhllqHWeZlTGr  
cIynnYlchGNsYcpqUesE
```

Esta función genera una contraseña aleatoria compuesta por:

1. Una letra mayúscula aleatoria.
2. Un dígito aleatorio (entre 0 y 9).
3. Un símbolo especial aleatorio de una lista predefinida de símbolos.
4. Una cadena alfanumérica aleatoria de 4 caracteres.
5. Una cadena alfanumérica adicional aleatoria de longitud variable entre 1 y 12 caracteres.

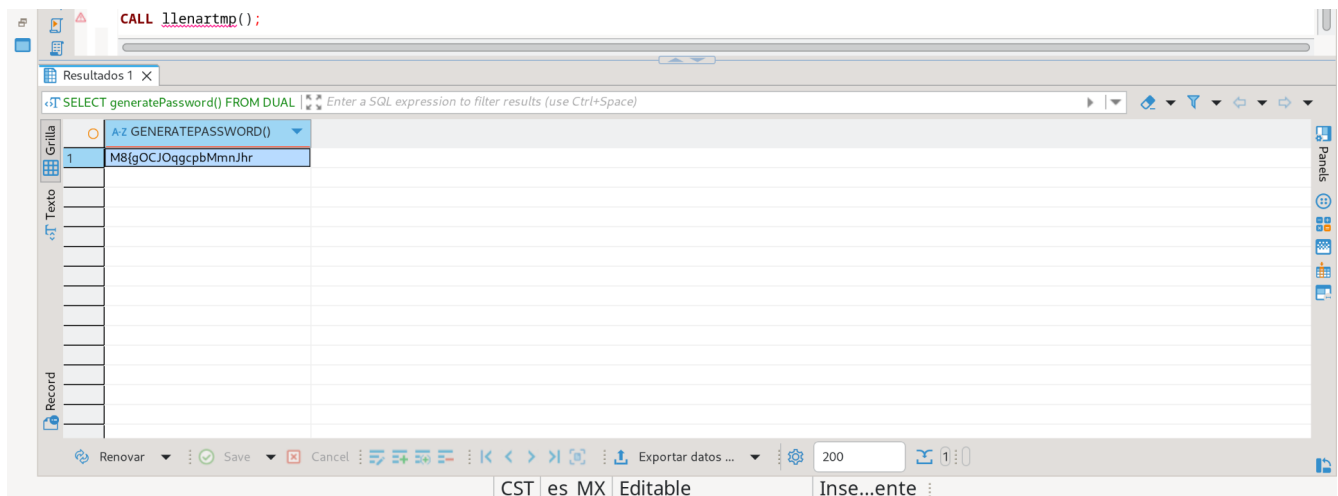
```
CREATE OR REPLACE FUNCTION generatePassword RETURN VARCHAR AS
password VARCHAR(20);
first_letter VARCHAR(1);
second_digit VARCHAR(1);
third_special_symbol VARCHAR(1);
alphanumeric VARCHAR(4);
OTHERS VARCHAR(12);
BEGIN
    SELECT DBMS_RANDOM.string('U',1) INTO first_letter FROM DUAL;
    SELECT TRUNC(DBMS_RANDOM.value(0,9)) INTO second_digit FROM DUAL;
    SELECT SUBSTR('+$#()-!{}[]]', TRUNC(DBMS_RANDOM.VALUE(1, 14)), 1) INTO
third_special_symbol FROM DUAL;
    SELECT DBMS_RANDOM.string('A',4) INTO alphanumeric FROM DUAL;
    SELECT DBMS_RANDOM.string('A',TRUNC(DBMS_RANDOM.value(1,12))) INTO OTHERS
FROM DUAL;
    password := first_letter || second_digit || third_special_symbol ||
alphanumeric || OTHERS;
    RETURN password;
END;
```



Name	Value
Updated Rows	0
Execute time	0.004s
Start time	Wed Apr 09 08:22:17 CST 2025
Finish time	Wed Apr 09 08:22:17 CST 2025
Query	CREATE OR REPLACE FUNCTION generatePassword RETURN VARCHAR AS password VARCHAR(20); first_letter VARCHAR(1); second_digit VARCHAR(1); third_special_symbol VARCHAR(1); alphanumeric VARCHAR(4); OTHERS VARCHAR(12);

Prueba de funcionamiento, llamando al metodo que generamos: generatePassword().

```
SELECT generatePassword() FROM DUAL;
```



El procedimiento `llenartmp` inserta 1 millón de registros en la tabla `tmp`. Cada registro tiene un valor único en la columna `ID` (que es el valor de la variable `i`), y la columna `campox` contiene una cadena aleatoria de 100 caracteres generada por `DBMS_RANDOM.STRING('X', 100)`.

```
CREATE TABLE tmp(ID INTEGER, campox VARCHAR2(100));
```

```
CREATE OR REPLACE PROCEDURE llenartmp AS
```

```
i INTEGER;
```

```
BEGIN
```

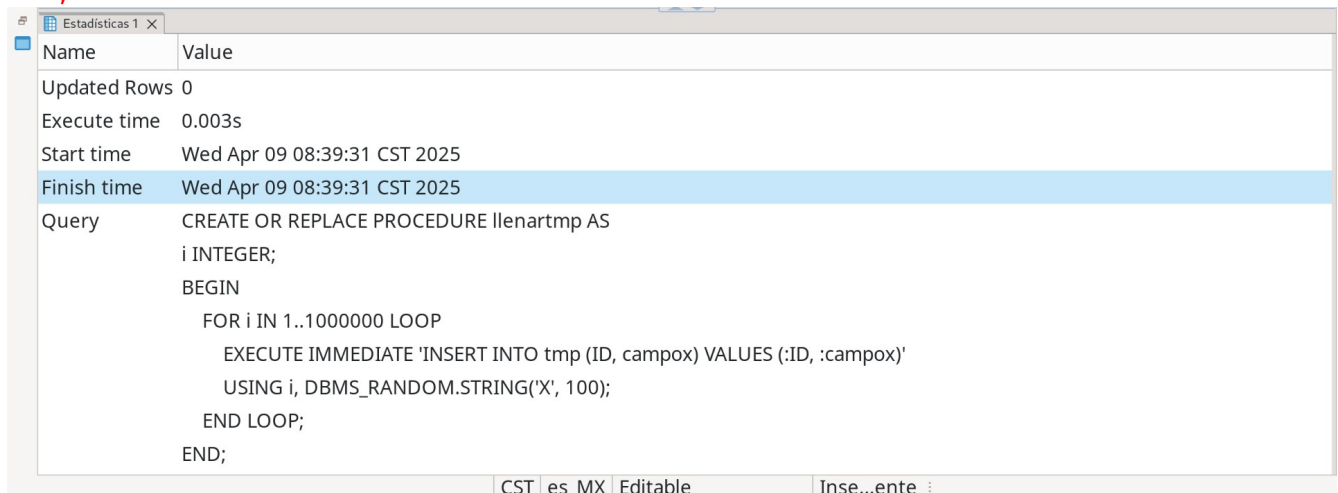
```
    FOR i IN 1..1000000 LOOP
```

```
        EXECUTE IMMEDIATE 'INSERT INTO tmp (ID, campox) VALUES (:ID, :campox)'
```

```
        USING i, DBMS_RANDOM.STRING('X', 100);
```

```
    END LOOP;
```

```
END;
```



Debido a que es un proceso tardado puede llegar a demorar dependiendo de la potencia de la maquina que la este ejecutando. En caso de que el usuario no pueda ejecutar la consulta es necesario ejecutar

```
ALTER USER SYSDG QUOTA UNLIMITED ON USERS;
```

Help: <https://docs.oracle.com/error-help/db/ora-00911/>

```
SQL> ALTER USER SYSDG QUOTA UNLIMITED ON USERS;
```

User altered.

```
SQL> █
```

o asignarle una cuota suficiente para ejecutar la consulta

```
CALL llenartmp();
```

Estadísticas 1 X	
Name	Value
Updated Rows	-1
Execute time	1m 53s
Start time	Wed Apr 09 08:36:14 CST 2025
Finish time	Wed Apr 09 08:38:07 CST 2025
Query	CALL llenartmp()

Podemos saber si el procedimiento fue exitoso seleccionando la tabla y viendo si se agregaron las filas.

```
SELECT COUNT(*) FROM tmp;
```

Resultados 1 X

SQL: `SELECT COUNT(*) FROM tmp`

Grilla	123 COUNT(*)
1	1,000,000

1 row(s) fetched - 0.138s, on 2025-04-09 at 08:41:44

Resultados 1 X

SQL: `SELECT * FROM tmp WHERE ROWNUM <= 10`

Grilla	123 ID	AZ CAMPOX
1	133	ERD19X3SLU1BN5OPIE2VWY2NBMTQXTO9F5QHMYU4GBHUKJ0703R7R5YQD686RS10MST93ITYPKQGH3VCQKXOZNSYZNDQV423A4O
2	134	2U6QNI098T23R6UQFTTLRHNUJEY1TUV5KX12D2MX8H2WQ5I46WYOURSJG57JIDTT6V5HUS5Z3QBXZRI8WQOMNO0P9MWGFQEC7EKY
3	135	A34000SRJKSPY850IUW4REA8IEAA4WWAV5BLQGE79TF7U2HYKBHYSU7TIEATWO2FQ3BOOK7SNBCLGR73P59WR34EKMFM4GM0WQWI
4	136	MPQ7MUK8MFV76KQNR6TAPQBT0USJ1842GXBMT5DIOLQ65AWMGBMG7CMJRN3PB3C8JTDVIA7FTWVLPNA36HGGPOG3ZPJGHV95KK
5	137	1B7TVICIT2AULX5J604SBJJC16GKMGWXXH6GOT9JZVZ159GLOK03HDTDAGY8QZS12MZSP0ZCOSMS3U09PX9QARFN2ZIRW4N
6	138	42EGKJWCV36X8BQOWIG9JPH9257SWZZTU09OEXMK1GNJ2D3KKG5BMLX6UL8HIW5Q4GLM8C9YY4SM2KTN9Z3IKXOPMI5XURB6W95B
7	139	QCULKY05RU4LCZ3T5FRYQJ34K4UDYYTWC6UK27QOUT8V400XQXUZFUFUWNEZJXH230Y5F6UHZ9ZUVKR2A2PDNDNTZKT40PTRUSEB
8	140	BV6XZXPV97W7P3D7B5W2LS9FWN9U1D57FY0AE9XKKOP06BBUPUGKNR8JFS1BV3P5WIPIM5WQXSAZLRD3BL1WFYESBAACNQHIE7
9	141	TG7LMMQJ9IZWJ4MCSJOJPT05ZSM7G6HS1YCQZ5AFEVXXYN2WB97CEZT8TT2T8CBHBW96SN7YKGM5NUV4VFSKHLA553JAHMXQS73
10	142	QB9UVQT42UPDP8VDPYLAHZTG995OCOQIY4P0M9XMBAOIDYXZF77751K0KEI8KN0ZFDH50189SUYGGPHCW4PAJJ880TKNAPITV

10 rows fetched - 0.138s, on 2025-04-09 at 08:41:44

Figura 1: También podemos consultar la información de algunas columnas. `SELECT * FROM tmp WHERE ROWNUM <= 10;`

EMPLEADOS

Como yo no tenía la tabla de empleados la cree con

```
CREATE TABLE empleados (
```

```
    id_empleado NUMBER PRIMARY KEY,
```

```
    nombre VARCHAR2(100),
```

```
    salario NUMBER
```

```
);
```

Ahora, si es posible iniciar con las funciones y procedimientos.

Agregar empleados

```
CREATE OR REPLACE PROCEDURE agregar_empleado(
```

```
    p_id IN NUMBER,
```

```
    p_nombre IN VARCHAR2,
```

```

    p_salario IN NUMBER
) IS
BEGIN
    INSERT INTO empleados (id_empleado, nombre, salario)
    VALUES (p_id, p_nombre, p_salario);
    COMMIT;
END agregar_empleado;

```

```

SQL> CREATE OR REPLACE PROCEDURE agregar_empleado( p_id IN NUMBER,
    p_nombre IN VARCHAR2, p_salario IN NUMBER ) IS BEGIN INSERT INTO
empleados (id_empleado, nombre, salario) VALUES (p_id, p_nombre, p
_salario); COMMIT; END agregar_empleado;
2 /

```

Procedure created.

Ahora que el procedimiento está creado, podemos confirmar que funciona haciendo un CALL O un EXEC

Ejemplo: Agregamos como primer empleado a Pepe con un salario de \$5000

```
EXEC agregar_empleado(1, 'Pepe', 5000);
```

```
SQL> EXEC agregar_empleado(1, 'Pepe', 5000);
```

PL/SQL procedure successfully completed.

```
SQL> SELECT * FROM EMPLEADOS;
```

ID_EMPLEADO

NOMBRE

SALARIO

1

Pepe

5000

Figura 2: Como vemos fue posible agregar al empleado Pepe y sus atributos son: ID: 1, NOMBRE: Pepe, SALARIO: 5000

Agregar empleados pero ahora con condiciones

```

CREATE OR REPLACE PROCEDURE agregar_empleado(
    p_id IN NUMBER,
    p_nombre IN VARCHAR2,

```

```

    p_salario IN NUMBER
) IS
BEGIN
    IF p_salario >= 5000 AND p_salario <= 30000 THEN
        INSERT INTO empleados (id_empleado, nombre, salario)
        VALUES (p_id, p_nombre, p_salario);
        COMMIT;
        DBMS_OUTPUT.PUT_LINE('Empleado agregado correctamente');
    ELSE
        DBMS_OUTPUT.PUT_LINE('No se pudo agregar al empleado: salario fuera de rango');
    END IF;
END agregar_empleado;
/

```

```

SQL> CREATE OR REPLACE PROCEDURE agregar_empleado( p_id IN NUMBER,
  p_nombre IN VARCHAR2, p_salario IN NUMBER ) IS BEGIN IF p_salario
  >= 5000 AND p_salario <= 30000 THEN INSERT INTO empleados (id_emp
leado, nombre, salario) VALUES (p_id, p_nombre, p_salario); COMMIT
; DBMS_OUTPUT.PUT_LINE('Empleado agregado correctamente'); ELSE DB
MS_OUTPUT.PUT_LINE('No se pudo agregar al empleado: salario fuera
de rango'); END IF; END agregar_empleado;
  2  /

Procedure created.

```

Ejemplo: Agregamos a un empleado con un salario menor al establecido por la condición, para que funcione debemos habilitar el SERVEROUTPUT con:

```

SET SERVEROUTPUT ON

CALL agregar_empleado(2, 'Carlos', 3000);

```



```

SQL> set serveroutput on
SP2-0265: serveroutput must be set ON or OFF
Help: https://docs.oracle.com/error-help/db/sp2-0265/
SQL> SET SERVEROUTPUT ON
SQL> CALL agregar_empleado(2, 'Carlos', 3000);
No se pudo agregar al empleado: salario fuera de rango

Call completed.

SQL> █

```

Figura 3: El mensaje se mostro correctamente, " No se pudo agregar al empleado: salario fuera de rango"

Al consultar los datos de la tabla de empleados podemos ver que no se registro en la BD

```
SELECT * FROM EMPLEADOS;
```

```

SQL> select * from empleados;

ID_EMPLEADO
-----
NOMBRE
-----
SALARIO
-----
          1
Pepe
      5000

SQL> █

```

Por otra parte si se agrega un empleado dentro de la condición aparece como agregado y se puede visualizar, nuevamente seleccionando los registros

```
SELECT * FROM EMPLEADOS;
```

```
SQL> EXEC agregar_empleado(2, 'Carlos', 9000);
Empleado agregado correctamente

PL/SQL procedure successfully completed.
```

Figura 4: Obsérvese que se muestra la leyenda de "Empleado agregado correctamente"
Aqui se agrega, pero ahora lo consultamos.

```
SQL> SELECT * FROM EMPLEADOS;

ID_EMPLEADO
-----
NOMBRE
-----
SALARIO
-----
1
Pepe
5000
2
Carlos
9000
```

Obtener salario de empleado

```
CREATE OR REPLACE FUNCTION obtener_salario(
    p_id IN NUMBER
) RETURN NUMBER IS
    v_salario empleados.salario%TYPE;
BEGIN
    SELECT salario INTO v_salario
    FROM empleados
    WHERE id_empleado = p_id;

    RETURN v_salario;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN NULL;
```

```
END obtener_salario;
```

```
Procedure created.
```

```
SQL> CREATE OR REPLACE FUNCTION obtener_salario( p_id IN NUMBER )  
RETURN NUMBER IS v_salario empleados.salario%TYPE; BEGIN SELECT sa  
lario INTO v_salario FROM empleados WHERE id_empleado = p_id; RETU  
RN v_salario; EXCEPTION WHEN NO_DATA_FOUND THEN RETURN NULL; END o  
btener_salario;  
2 /
```

```
Function created.
```

Para usar este procedimiento ejecutamos el siguiente comando. Es posible también usar CALL pero debido a la naturaleza de las funciones debemos ser más explícitos y es por eso que se crean variables:

Ejemplo

Alternativa call:

```
VARIABLE v_salario NUMBER;  
  
CALL obtener_salario(1) INTO :v_salario;  
  
PRINT v_salario;
```

Alternativa más sencilla:

```
SELECT obtener_salario(1) FROM DUAL;
```

```
SQL> SELECT obtener_salario(1) FROM DUAL;
```

```
OBTENER_SALARIO(1)  
-----  
5000
```

```
SQL> SELECT * FROM EMPLEADOS;
```

```
ID_EMPLEADO  
-----  
NOMBRE  
-----  
SALARIO  
-----  
1  
Pepe  
5000
```

Figura 5: Devuelve de DUAL (una tabla temporal) la información del salario de el empleado Pepe

Listar empleados

```
CREATE OR REPLACE PROCEDURE listar_empleados IS
```

```

CURSOR c_empleados IS
    SELECT id_empleado, nombre FROM empleados;
v_id empleados.id_empleado%TYPE;
v_nombre empleados.nombre%TYPE;
BEGIN
    OPEN c_empleados;
    LOOP
        FETCH c_empleados INTO v_id, v_nombre;
        EXIT WHEN c_empleados%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('ID: ' || v_id || ' Nombre: ' || v_nombre);
    END LOOP;
    CLOSE c_empleados;
END listar_empleados;
/

```

Ejemplo: Listar los empleados registrados hasta este momento.

Como hace uso del server output para devolver los atributos del Empleado en la linea

DBMS_OUTPUT.PUT_LINE('ID: ' || v_id || ' Nombre: ' || v_nombre); es necesaria habilitarlos nuevamente.

```
SET SERVEROUTPUT ON;
```

```
EXEC listar_empleados;
```

```
SQL> EXEC listar_empleados;
```

```
ID: 1  Nombre: Pepe
```

```
PL/SQL procedure successfully completed.
```

Suponiendo que se agregarán más empleados, también se podrían listar.

```
SQL> EXEC listar_empleados;
```

```
ID: 1  Nombre: Pepe
```

```
ID: 2  Nombre: Carlos
```

```
ID: 3  Nombre: Enrique
```

```
PL/SQL procedure successfully completed.
```

```
SQL>
```

Procedimiento para auditar empleados

```

CREATE OR REPLACE PROCEDURE auditar_empleado(
    p_id_empleado IN NUMBER,
    p_accion IN VARCHAR2

```

```

) IS
BEGIN
    INSERT INTO auditoria_empleados(id_empleado, accion, usuario)
    VALUES (p_id_empleado, p_accion, USER);

    COMMIT;
END auditar_empleado;
/

```

Como yo no tengo la tabla auditoria_empleados, voy a crearla con:

```

CREATE TABLE auditoria_empleados (
    id_empleado NUMBER NOT NULL,
    accion VARCHAR2(200) NOT NULL,
    usuario VARCHAR2(30) NOT NULL,
    fecha_hora TIMESTAMP DEFAULT SYSTIMESTAMP
);

```

Ejemplo: Podemos llamar al procedimiento para evaluar el desempeño del empleado con id 1, y ponerle un excelente desempeño.

```
EXEC auditar_empleado(1, 'REVISIÓN COMPLETA, EXCELENTE DESEMPEÑO');
```

```

SQL> EXEC auditar_empleado(1, 'REVISIÓN COMPLETA, EXCELENTE DESEMPEÑO');

PL/SQL procedure successfully completed.

SQL> select * from auditoria_empleados;

ID_EMPLEADO
-----
ACCION
-----
USUARIO
-----
FECHA_HORA
-----
1
REVISIÓN COMPLETA, EXCELENTE DESEMPEÑO
ALEX
01-MAY-25 07.09.22.800188 PM

```

Figura 6: Como vemos fue posible evaluar el desempeño del empleado 1

TRIGGERS

Trigger que se activa al insertar un salario menor a 0

```

CREATE OR REPLACE TRIGGER validar_salario
BEFORE INSERT ON empleados
FOR EACH ROW
BEGIN
    IF :NEW.salario < 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'El salario no puede ser negativo');
    END IF;
END validar_salario;
/

```

Ejemplo: Vamos a insertar en la tabla empleados con los atributos ID_empleado, nombre y salario. Los nuevos valores serán 5, Juan, -5000.

```
INSERT INTO empleados (id_empleado, nombre, salario)
VALUES (5, 'Juan', -5000);
```

```
SQL> INSERT INTO empleados (id_empleado, nombre, salario) VALUES (5, 'Juan', -5000);
INSERT INTO empleados (id_empleado, nombre, salario) VALUES (5, 'Juan', -5000)
      *
ERROR at line 1:
ORA-20001: El salario no puede ser negativo
ORA-06512: at "ALEX.VALIDAR_SALARIO", line 1
ORA-04088: error during execution of trigger 'ALEX.VALIDAR_SALARIO'
```

Figura 7: Ahora muestra el mensaje "EL SALARIO NO PUEDE SER NEGATIVO", el trigger funcionó.

Trigger para auditar la inserción de empleados.

```
CREATE OR REPLACE TRIGGER auditar_insercion_empleado
AFTER INSERT ON empleados
FOR EACH ROW
BEGIN
```

```
    INSERT INTO auditoria_empleados (id_empleado, accion, usuario)
    VALUES (:NEW.id_empleado, 'INSERCIÓN', USER);
```

```
END auditar_insercion_empleado;
```

Ejemplo: Agregar a un nuevo empleado con el ID: 5, Nombre: María López y un salario de 9000

```
INSERT INTO empleados (id_empleado, nombre, salario) VALUES (5, 'María López', 9000);
```

El trigger se ejecuta correctamente, pues esta acción se registra en la tabla auditoria_empleados.

```

SQL> INSERT INTO empleados (id_empleado, nombre, salario) VALUES (5, 'María López', 9000);

1 row created.

SQL> select * from auditoria_empleados
  2  ;

ID_EMPLEADO
-----
ACCION
-----
USUARIO
-----
FECHA_HORA
-----
1
REVISIÓN COMPLETA, EXCELENTE DESEMPEÑO
ALEX
01-MAY-25 07.09.22.800188 PM

ID_EMPLEADO
-----
ACCION
-----
USUARIO
-----
FECHA_HORA
-----
5
INSERCIÓN
ALEX
01-MAY-25 07.17.37.758768 PM

SQL>

```

Figura 8: El usuario Alex hizo una inserción

Trigger para auditar un cambio de salario.

```

CREATE OR REPLACE TRIGGER auditar_cambio_salario
BEFORE UPDATE OF salario ON empleados
FOR EACH ROW
BEGIN
    INSERT INTO auditoria_empleados (id_empleado, accion, usuario)
    VALUES (:OLD.id_empleado, 'CAMBIO SALARIO DE ' || :OLD.salario || ' A ' || :NEW.salario,
    USER);
END auditar_cambio_salario;

```

Ejemplo: Vamos a cambiar el salario del empleado con ID 1 a una cantidad mayor, 15000.

```
UPDATE empleados SET salario = 15000 WHERE id_empleado = 1;
```

Al consultar la tabla auditoria_empleados, podemos ver que efectivamente se muestra la actualización de salario.

```
ID_EMPLEADO
-----
ACCION
-----
USUARIO
-----
FECHA_HORA
-----
1
CAMBIO SALARIO DE 15000 A 15000
ALEX
01-MAY-25 07.21.24.291085 PM

SQL>
```

Trigger para prevenir borrado de jefe

```
CREATE OR REPLACE TRIGGER prevenir_borrado_jefe
```

```
BEFORE DELETE ON empleados
```

```
FOR EACH ROW
```

```
BEGIN
```

```
  IF :OLD.nombre = 'Director General' THEN
```

```
    RAISE_APPLICATION_ERROR(-20002, 'No se puede borrar el Director General');
```

```
  END IF;
```

```
END prevenir_borrado_jefe;
```

Ejemplo: Intentemos borrar un director general que previamente crearemos.

Se creara con:

```
INSERT INTO empleados (id_empleado, nombre, salario) VALUES (6, 'Director General', 50000);
```

```
SQL> INSERT INTO empleados (id_empleado, nombre, salario) VALUES (6, 'Director General', 50000);
```

```
1 row created.
```

```
SQL> select * from empleados;
```

```
ID_EMPLEADO
```

```
NOMBRE
```

```
SALARIO
```

```
6
```

```
Director General
```

```
50000
```

```
1
```

```
Pepe
```

```
15000
```

Ahora intentamos borrarlo.

```
DELETE FROM empleados WHERE id_empleado = 10;
```



```
SQL> DELETE FROM empleados WHERE id_empleado = 6;
DELETE FROM empleados WHERE id_empleado = 6
      *
ERROR at line 1:
ORA-20002: No se puede borrar el Director General
ORA-06512: at "ALEX.PREVENIR_BORRADO_JEFE", line 1
ORA-04088: error during execution of trigger 'ALEX.PREVENIR_BORRADO_JEFE'
```

Figura 9: No se puede borrar por el trigger que creamos

TRIGGER a nivel Statement: Solo se ejecuta una vez en lugar de fila por fila

Triger para bloqueo de tabla

```
CREATE OR REPLACE TRIGGER trigger_bloqueo_tabla
BEFORE UPDATE ON empleados
BEGIN
```

```
    DBMS_OUTPUT.PUT_LINE('Se esta intentando actualizar la tabla EMPLEADOS');
```

Ejemplo: Si quisieramos actualizar el salario de un empleado, digamos el empleado con id 2. Al ejecutar la sentencia:

```
UPDATE empleados SET salario = salario * 1.05 WHERE id_empleado = 2;
```

```
SQL> UPDATE empleados SET salario = salario * 1.05 WHERE id_empleado = 2;
Se esta intentando actualizar la tabla EMPLEADOS
```

Aparece el mensaje de “Se esta intentando actualizar la tabla EMPLEADOS”

MARIADB

FUNCIONES Y PROCEDIMIENTOS

Función para crear contraseña con caracteres especiales.

Crear una función que genere una contraseña aleatoria con una longitud mínima de 8 y máxima de 20, con la primera letra mayúscula, segundo debe ser un dígito, el tercero un símbolo especial (+\$#()-!{}{[]}), los siguientes debe ser alfanumerico

```
DELIMITER //
```

```
CREATE FUNCTION generatePassword()
```

```
RETURNS VARCHAR(20)
```

```
NOT DETERMINISTIC
```

```
BEGIN
```

```
    DECLARE generated_password VARCHAR(20) DEFAULT "";
```

```
    DECLARE first_letter CHAR(1) DEFAULT "";
```

```
    DECLARE second_digit CHAR(1) DEFAULT "";
```

```
    DECLARE third_special_symbol CHAR(1) DEFAULT "";
```

```
    DECLARE alphanumeric VARCHAR(4) DEFAULT "";
```

```
    DECLARE others VARCHAR(12) DEFAULT "";
```

```
    DECLARE symbols VARCHAR(20) DEFAULT '($#()-!{}{[])';
```

```
    DECLARE i INT DEFAULT 0;
```

```
    DECLARE len INT DEFAULT 0;
```

```
    -- Letra mayúscula (A-Z)
```

```
    SET first_letter = CHAR(FLOOR(RAND() * 26) + 65);
```

```
    -- Dígito (0-9)
```

```
    SET second_digit = CHAR(FLOOR(RAND() * 10) + 48);
```

```
    -- Símbolo especial
```

```
SET i = FLOOR(RAND() * CHAR_LENGTH(symbols)) + 1;
```

```
SET third_special_symbol = SUBSTRING(symbols, i, 1);
```

```
-- 4 letras minúsculas aleatorias
```

```
SET alphanumeric = "";
```

```
SET i = 0;
```

```
WHILE i < 4 DO
```

```
    SET alphanumeric = CONCAT(alphanumeric, CHAR(FLOOR(RAND() * 26) + 97));
```

```
    SET i = i + 1;
```

```
END WHILE;
```

```
-- De 1 a 12 letras minúsculas aleatorias
```

```
SET others = "";
```

```
SET len = FLOOR(RAND() * 12) + 1;
```

```
SET i = 0;
```

```
WHILE i < len DO
```

```
    SET others = CONCAT(others, CHAR(FLOOR(RAND() * 26) + 97));
```

```
    SET i = i + 1;
```

```
END WHILE;
```

```
-- Concatenar componentes paso a paso
```

```
SET generated_password = CONCAT(first_letter, second_digit);
```

```
SET generated_password = CONCAT(generated_password, third_special_symbol);
```

```
SET generated_password = CONCAT(generated_password, alphanumeric);
```

```
SET generated_password = CONCAT(generated_password, others);
```

```
RETURN generated_password;
```

```
END;
```

```
//
```

```
DELIMITER ;
```

1. Como yo usaré Dbeaver, la función no necesita delimitadores pues Dbeaver los maneja implícitamente.

```
CREATE FUNCTION generatePassword()  
RETURNS VARCHAR(20)  
NOT DETERMINISTIC  
BEGIN  
    DECLARE generated_password VARCHAR(20) DEFAULT '';  
    DECLARE first_letter CHAR(1) DEFAULT '';  
    DECLARE second_digit CHAR(1) DEFAULT '';  
    DECLARE third_special_symbol CHAR(1) DEFAULT '';  
    DECLARE alphanumeric VARCHAR(4) DEFAULT '';  
    DECLARE others VARCHAR(12) DEFAULT '';  
    DECLARE symbols VARCHAR(20) DEFAULT '+$#@!()-';  
    DECLARE i INT DEFAULT 0;  
    DECLARE len INT DEFAULT 0;  
  
    SET first_letter = CHAR(FLOOR(RAND() * 26) + 65);  
    SET second_digit = CHAR(FLOOR(RAND() * 10) + 48);  
  
    SET i = FLOOR(RAND() * CHAR_LENGTH(symbols)) + 1;  
    SET third_special_symbol = SUBSTRING(symbols, i, 1);  
  
    SET alphanumeric = '';  
    SET i = 0;  
    WHILE i < 4 DO  
        SET alphanumeric = CONCAT(alphanumeric, CHAR(FLOOR(RAND() * 26) + 97));  
        SET i = i + 1;  
    END WHILE;  
  
    SET others = '';  
    SET len = FLOOR(RAND() * 12) + 1;  
    SET i = 0;  
    WHILE i < len DO  
        SET others = CONCAT(others, CHAR(FLOOR(RAND() * 26) + 97));  
        SET i = i + 1;  
    END WHILE;  
  
    SET generated_password = CONCAT(first_letter, second_digit, third_special_symbol, alphanumeric,  
others);  
  
    RETURN generated_password;  
END;
```

2. Una vez que se creo la función la podemos llamar con el comando

```
SELECT generatePassword();
```

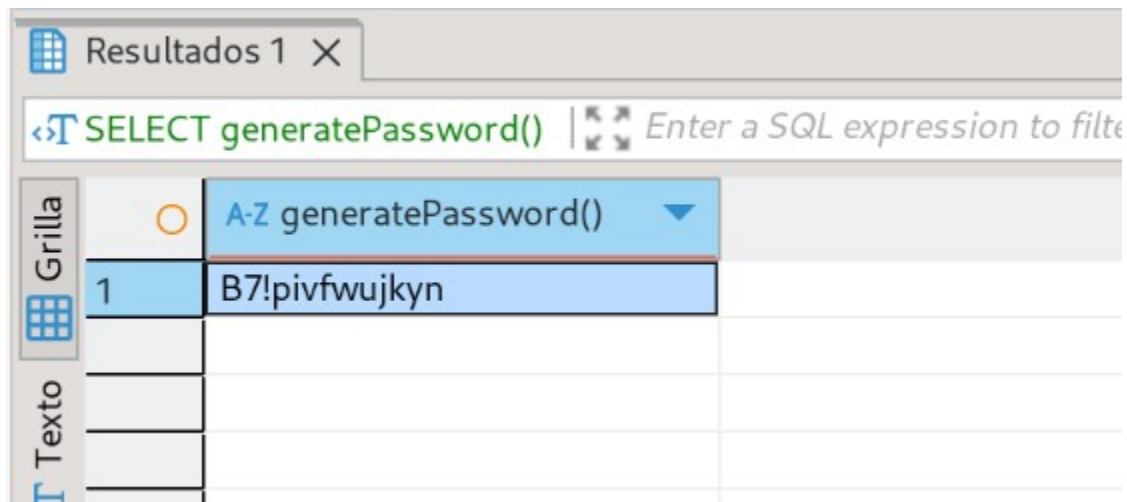


Figura 10: Esta fue la contraseña generada

2. Crear base de datos rh.

CREATE DATABASE IF NOT EXISTS rh;

```
| rh |
| sys |
+-----+
9 rows in set (0.008 sec)

MariaDB [(none)]> use rh;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with 'nohints'

Database changed
MariaDB [rh]>
```

USE rh;

3. Crear tabla empleados.

CREATE TABLE empleados (id_empleado INT PRIMARY KEY, nombre VARCHAR(100), salario DOUBLE);

3.1 Crear tabla auditorio_empleados

CREATE TABLE IF NOT EXISTS auditoria_empleados (
id_empleado INT NOT NULL,
accion VARCHAR(200) NOT NULL,
usuario VARCHAR(30) NOT NULL,
fecha_hora TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

Procedimiento para agregar empleados

4. Crear un procedimiento para agregar empleados.

```
DELIMITER //  

CREATE PROCEDURE agregar_empleado(  

IN p_id INT,
```

```

IN p_nombre VARCHAR(255),
IN p_salario DECIMAL(10,2)
)
BEGIN
INSERT INTO empleados (id_empleado, nombre, salario)
VALUES (p_id, p_nombre, p_salario);

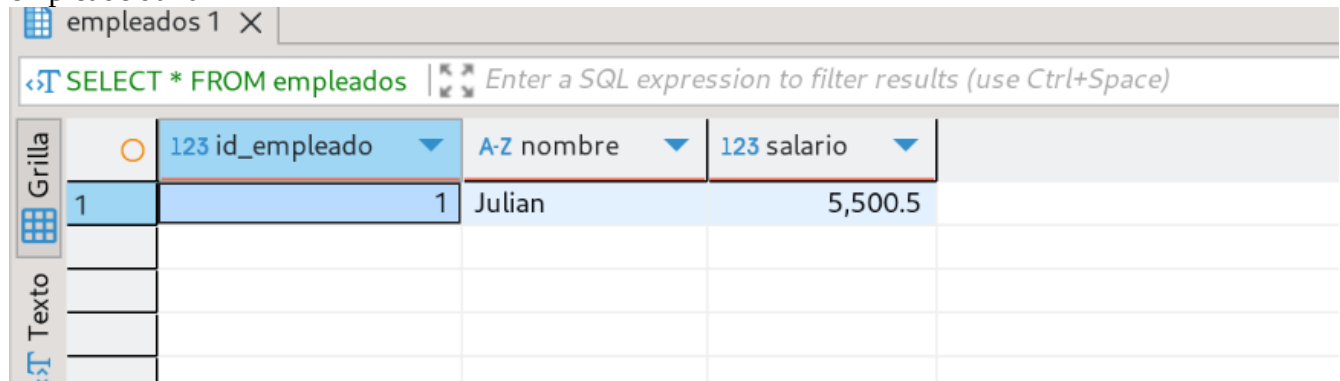
COMMIT;
END;
//
DELIMITER ;

```

Ejemplo: Agregar un empleado con el id 1, nombre Julian y salario 5,500.50

CALL agregar_empleado(1, 'Julian', 5500.50);

6. Una vez que se ejecuto el call, al seleccionar los datos de la tabla se puede observar que se registro el empleado Julian



	123 id_empleado	A-Z nombre	123 salario
1	1	Julian	5,500.5

Procedimiento de agregación de empleados con validación

```

DELIMITER //
CREATE OR REPLACE PROCEDURE agregar_empleado(
  IN p_id INT,
  IN p_nombre VARCHAR(255),
  IN p_salario DECIMAL(10,2)
)
BEGIN
  IF p_salario >= 5000 AND p_salario <= 30000 THEN
    INSERT INTO empleados (id_empleado, nombre, salario)
    VALUES (p_id, p_nombre, p_salario);
    COMMIT;
  ELSE
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'No se pudo agregar el empleado: salario fuera de
rango';
  END IF;
END;
//
DELIMITER ;

```

Ejemplo: Agregar un empleado cuyo salario sea menor a 0

CALL agregar_empleado(1, 'Alienigena', -5000);

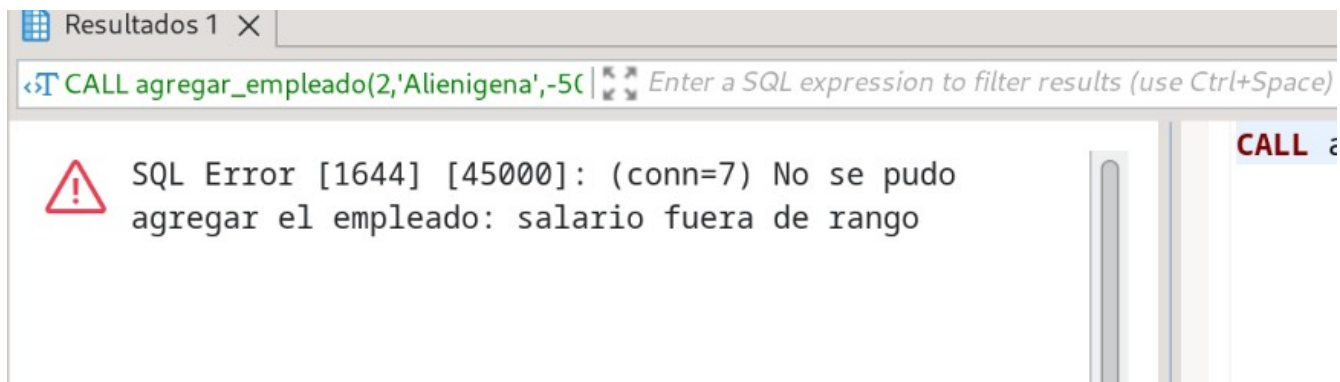


Figura 11: El empleado muestra que su salario esta fuera de rango, por lo que no se agrega a la tabla Comprobando

empleados 1 X

SELECT * FROM empleados Enter a SQL expression to filter results (use Ctrl+Space)

Grilla	123 id_empleado	A-Z nombre	123 salario
1	1	Julian	5,500.5

Figura 12: Al usar `SELECT * FROM empleados;` no se ve al empleado alienigena

Ejemplo: Agregar al siguiente empleado: ID: 2, Nombre: Julian y con salario de 15,000.

CALL agregar_empleado(2, 'Juan Climaco Rubio Cosme', 15000);

empleados 1 X

SELECT * FROM empleados Enter a SQL expression to filter results (use Ctrl+Space)

Grilla	123 id_empleado	A-Z nombre	123 salario
1	1	Julian	5,500.5
2	2	Juan Climaco Rubio Cosme	15,000

Name	Value
Updated Rows	1
Execute time	0.008s
Start time	Sun May 04 12:45:01 CST 2025
Finish time	Sun May 04 12:45:01 CST 2025
Query	CALL agregar_empleado(2,'Juan Climaco Rubio Cosme',15000)

En este caso si se agrega correctamente, pues su salario esta dentro del rango permitido.

Obtener el salario de un empleado.

```
CREATE FUNCTION obtener_salario(p_id INT)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    RETURN (
        SELECT salario
        FROM empleados
        WHERE id_empleado = p_id
        LIMIT 1
    );
END;
```

Ahora podemos usarla llamandola y especificando el id del empleado del cual queremos consultar el salario.

```
SELECT obtener_salario(2) AS salario_empleado;
```

Resultados 1	
SELECT obtener_salario(2) AS salario_em Enter a SQL expression to filter results (use Ctrl+S	
Grilla	123 salario_empleado
1	15,000

empleados 1			
SELECT * FROM empleados Enter a SQL expression to filter results (use Ctrl+Space)			
Grilla	123 id_empleado	A-Z nombre	123 salario
1	1	Julian	5,500.5
2	2	Juan Climaco Rubio Cosme	15,000

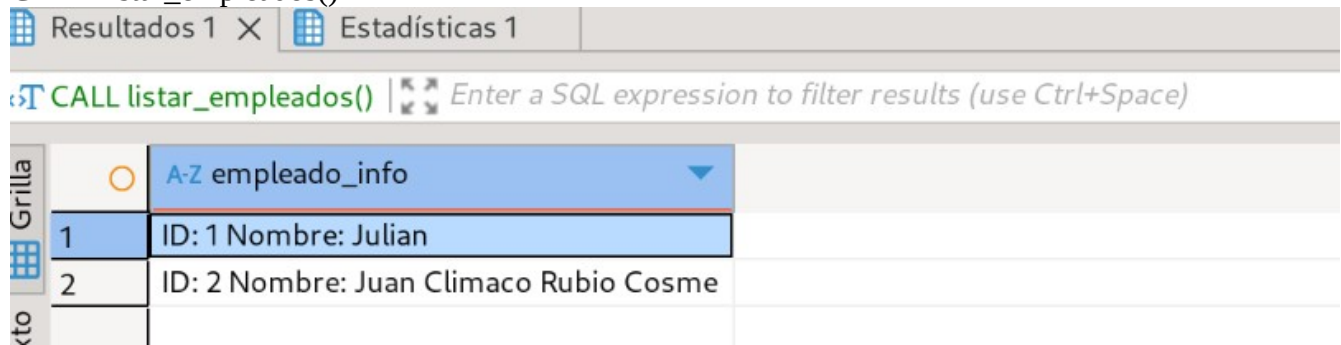
Listar empleados:

```
CREATE PROCEDURE listar_empleados()
BEGIN
    SELECT CONCAT('ID: ', id_empleado, ' Nombre: ', nombre) AS empleado_info
    FROM empleados;
END;
```

Este procedimiento llamado listar_empleados muestra una lista de todos los empleados de la tabla, donde para cada uno combina su ID y nombre en una sola línea de texto que se devuelve como resultado cuando se ejecuta.

Se llama con

CALL listar_empleados()



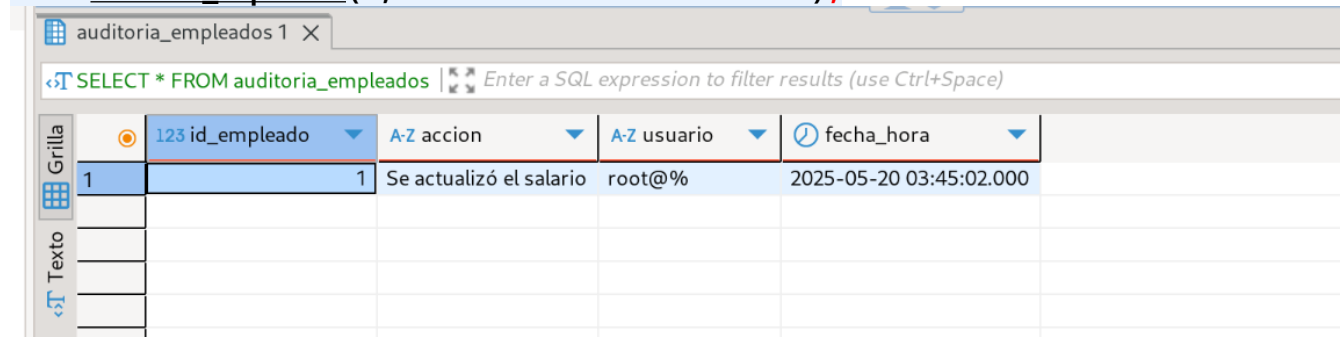
	A-Z empleado_info
1	ID: 1 Nombre: Julian
2	ID: 2 Nombre: Juan Climaco Rubio Cosme

Auditar empleados:

```
CREATE PROCEDURE auditar_empleado(
    IN p_id_empleado INT,
    IN p_accion VARCHAR(255)
)
BEGIN
    INSERT INTO auditoria_empleados(id_empleado, accion, usuario)
    VALUES (p_id_empleado, p_accion, CURRENT_USER());
    COMMIT;
END;
```

Ejemplo: Agregar a la base de datos de la auditoria el cambio en el salario de un empleado.

CALL auditar_empleado(1, 'Se actualizó el salario');



	123 id_empleado	A-Z accion	A-Z usuario	fecha_hora
1	1	Se actualizó el salario	root@%	2025-05-20 03:45:02.000

TRIGGERS

Trigger que se activa al insertar un salario menor a 0

```
CREATE TRIGGER validar_salario
BEFORE INSERT ON empleados
```

```

FOR EACH ROW
BEGIN
    IF NEW.salario < 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El salario no puede ser
negativo';
    END IF;
END;

```

Ejemplo: Insertar un empleado con un salario menor a 0
INSERT INTO empleados (id_empleado, nombre, salario)
VALUES (5, 'Juan', -5000);

Resultados 1 X

SQL Error [1644] [45000]: (conn=7) El salario no puede ser negativo

INSERT INTO empleados (id_e
VALUES (5, 'Juan', -5000)

Trigger para auditar la inserción de empleados.

```

CREATE TRIGGER auditar_insercion_empleado
AFTER INSERT ON empleados
FOR EACH ROW
BEGIN
    INSERT INTO auditoria_empleados (id_empleado, accion, usuario)
    VALUES (NEW.id_empleado, 'INSERCIÓN', CURRENT_USER());
END;

```

Ejemplo: Insertaremos a un empleado, después visualizaremos en la tabla auditoria_empleados si paso por ahí.

INSERT INTO empleados (id_empleado, nombre, salario) VALUES (5, 'María López', 9000);

empleados 1 X

select * from empleados

	123 id_empleado	A-Z nombre	123 salario
1	1	Julian	5,500.5
2	2	Juan Climaco Rubio Cosme	15,000
3	5	María López	9,000

Como se ve, si paso por la auditoria con la acción INSERCIÓN

auditoria_empleados 1 X

select * from auditoria_empleados

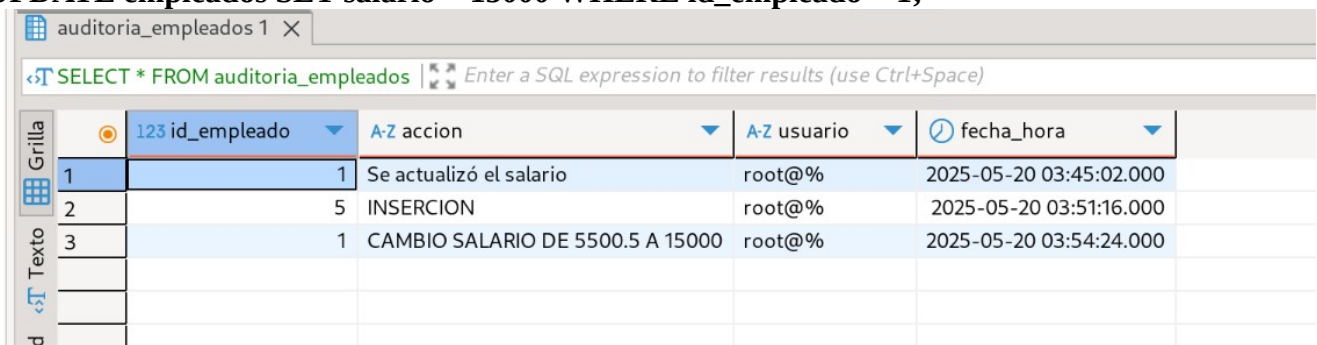
	123 id_empleado	A-Z accion	A-Z usuario	fecha_hora
1	1	Se actualizó el salario	root@%	2025-05-20 03:45:02.000
2	5	INSERCIÓN	root@%	2025-05-20 03:51:16.000

Trigger para auditar un cambio de salario.

```
CREATE TRIGGER auditar_cambio_salario
BEFORE UPDATE ON empleados
FOR EACH ROW
BEGIN
    IF OLD.salario <> NEW.salario THEN
        INSERT INTO auditoria_empleados (
            id_empleado,
            accion,
            usuario
        ) VALUES (
            OLD.id_empleado,
            CONCAT('CAMBIO SALARIO DE ', OLD.salario, ' A ', NEW.salario),
            CURRENT_USER()
        );
    END IF;
END;
```

Ejemplo: Cambiar el salario de un empleado y ver si se registra en la auditoria.

UPDATE empleados SET salario = 15000 WHERE id_empleado = 1;



	123 id_empleado	A-Z accion	A-Z usuario	fecha_hora
1	1	Se actualizó el salario	root@%	2025-05-20 03:45:02.000
2	5	INSERCIÓN	root@%	2025-05-20 03:51:16.000
3	1	CAMBIO SALARIO DE 5500.5 A 15000	root@%	2025-05-20 03:54:24.000

Como se ve, el cambio se refleja en auditoria_empleados.

Trigger para prevenir borrado de jefe

```
CREATE TRIGGER prevenir_borrado_jefe
BEFORE DELETE ON empleados
FOR EACH ROW
BEGIN
    IF OLD.nombre = 'Director General' THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No se puede borrar el Director General';
    END IF;
END;
```

Primero tenemos que insertar a un director general.

```
INSERT INTO empleados (id_empleado, nombre, salario) VALUES (6, 'Director General', 50000);
```

Ahora si podemos proceder a intentar eliminarlo.

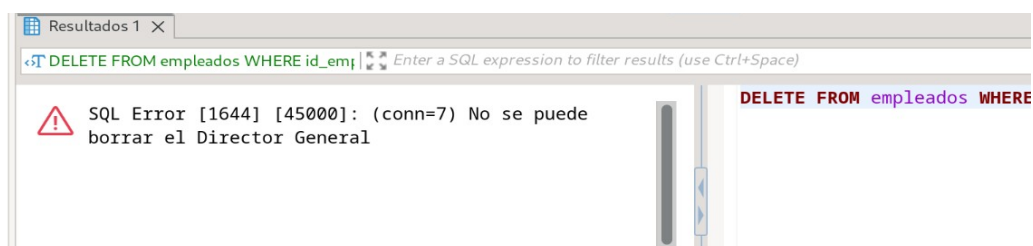


Figura 13: SQL Error [1644] [45000]: (conn=7) No se puede borrar el Director General

TRIGGER a nivel Statement: Solo se ejecuta una vez en lugar de fila por fila

Trigger para bloqueo de tabla

```
CREATE TRIGGER trigger_bloqueo_tabla
```

```
BEFORE UPDATE ON empleados
```

```
FOR EACH ROW
```

```
BEGIN
```

```
    INSERT INTO auditoria_empleados (id_empleado, accion, usuario)
```

```
    VALUES (NEW.id_empleado, 'Se está intentando actualizar la tabla EMPLEADOS',
```

```
    CURRENT_USER());
```

```
END;
```

auditoria_empleados 1 X				
SELECT * FROM auditoria_empleados Enter a SQL expression to filter results (use Ctrl+Space)				
Grilla	123 id_empleado	A-Z accion	A-Z usuario	fecha_hora
4	6	INSERTION	root@%	2025-05-20 03:55:41.000
5	2	CAMBIO SALARIO DE 15000 A 15750	root@%	2025-05-20 04:01:59.000
6	2	Se está intentando actualizar la tabla EMPLEADOS	root@%	2025-05-20 04:01:59.000
7	2	CAMBIO SALARIO DE 15750 A 16537.5	root@%	2025-05-20 04:02:03.000
8	2	Se está intentando actualizar la tabla EMPLEADOS	root@%	2025-05-20 04:02:03.000
9	2	CAMBIO SALARIO DE 16537.5 A 17364.375	root@%	2025-05-20 04:02:11.000
10	2	Se está intentando actualizar la tabla EMPLEADOS	root@%	2025-05-20 04:02:11.000

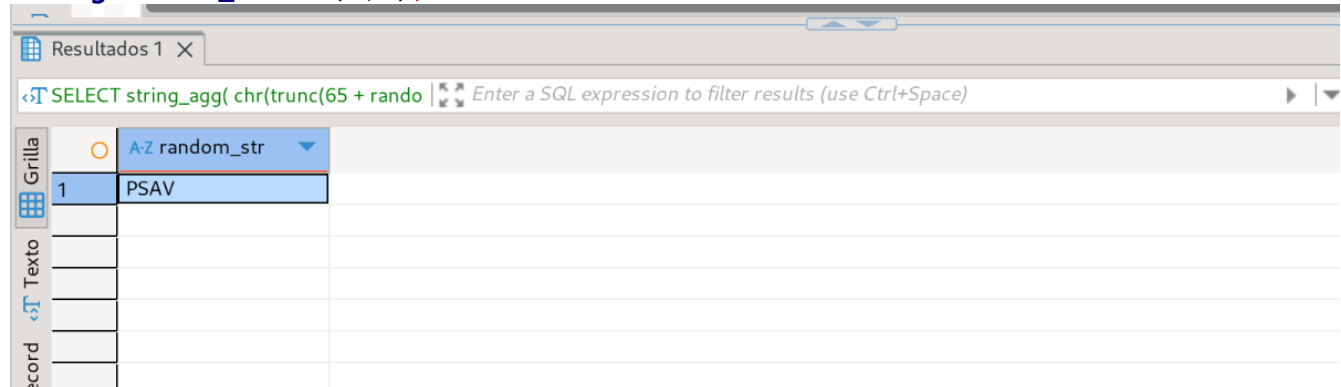
Se registra en la tabla auditoria_empleados los cambios como intento de actualizar la tabla de empleados.

POSTGRES

Funciones y triggers.

FUNCIÓN QUE GENERA CADENA DE TEXTO ALEATORIA . 'U' INDICA QUE SOLO SERÁN LETRAS MAYÚSCULAS CON UNA LONGITUD DE 4 CARACTERES

```
SELECT string_agg(  
    chr(trunc(65 + random() * 26)::int), ''  
    ) AS random_str  
FROM generate_series(1,4);
```



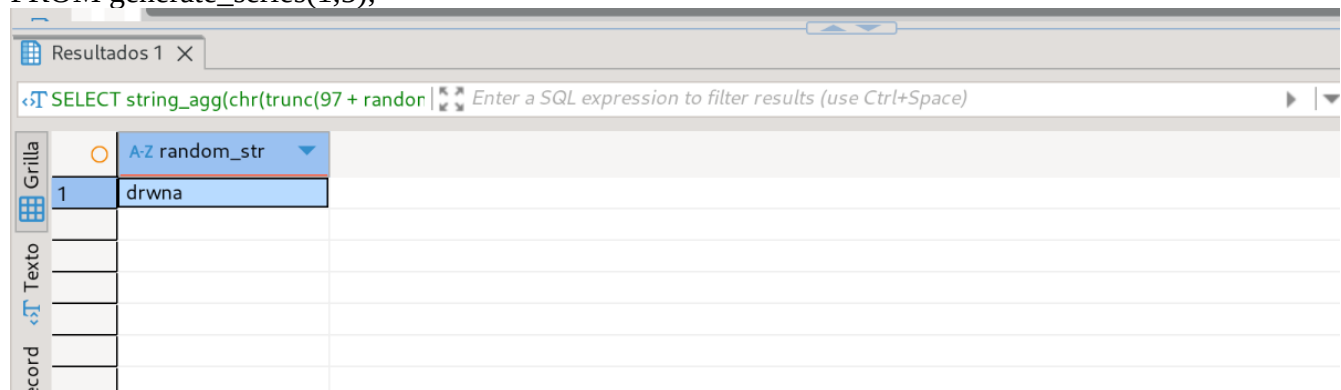
Resultados 1 X

SELECT string_agg(chr(trunc(65 + random() * 26)::int), '') AS random_str

Grilla	A-Z random_str
1	PSAV

Función que genera cadena aleatoria de letras minúsculas de longitud 5

```
SELECT string_agg(chr(trunc(97 + random() * 26)::int), '') AS random_str  
FROM generate_series(1,5);
```



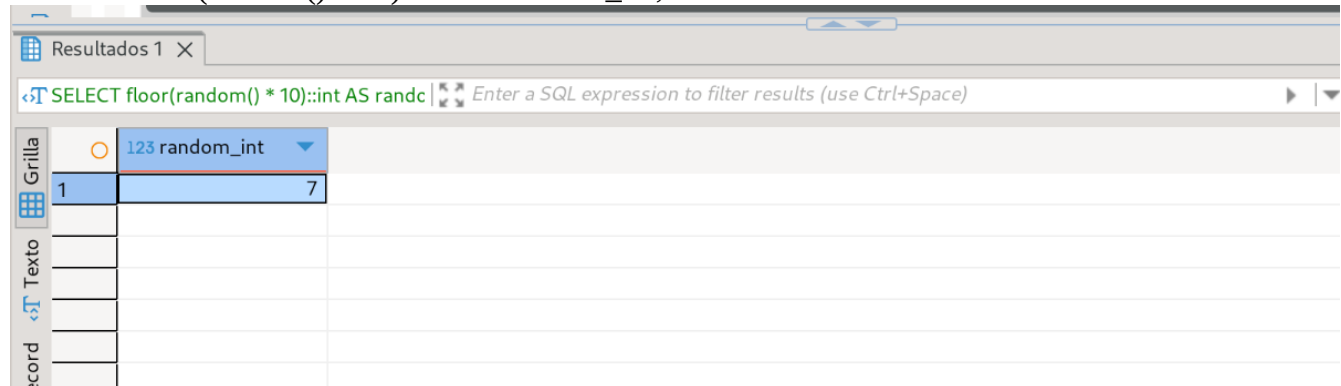
Resultados 1 X

SELECT string_agg(chr(trunc(97 + random() * 26)::int), '') AS random_str

Grilla	A-Z random_str
1	drwna

Función que devuelve un número decimal aleatorio entre 0 y 9 truncado a entero

```
SELECT floor(random() * 10)::int AS random_int;
```



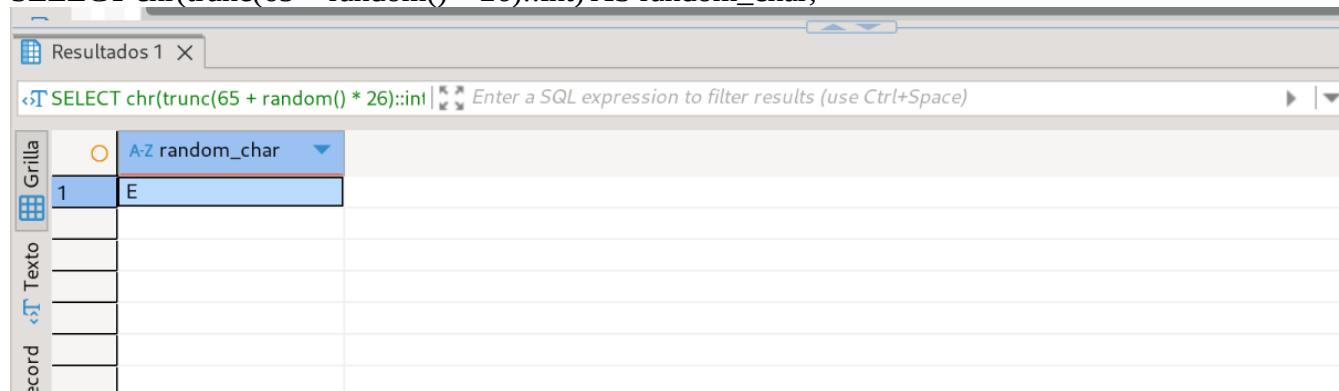
Resultados 1 X

SELECT floor(random() * 10)::int AS random_int

Grilla	123 random_int
1	7

Función que devuelve un caracter aleatorio mayúscula de 1 carácter

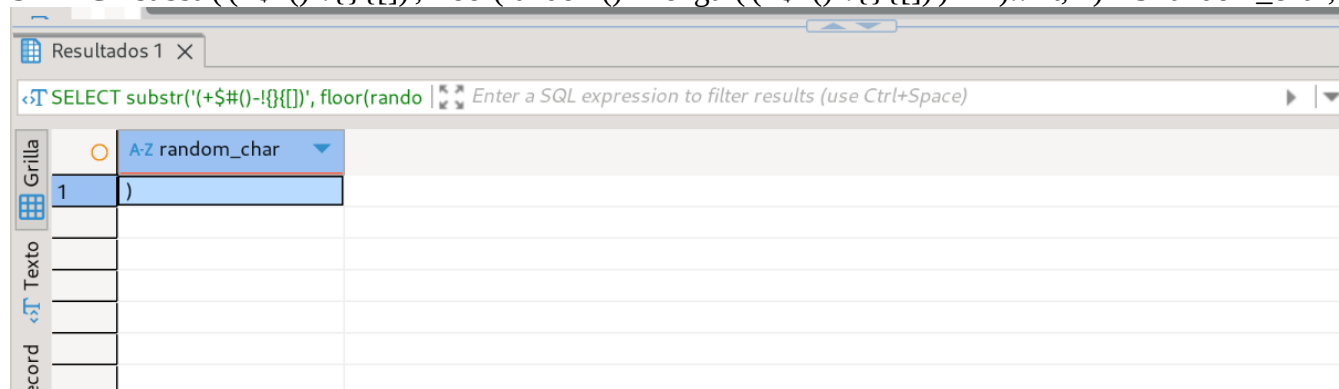
SELECT chr(trunc(65 + random() * 26)::int) AS random_char;



Grilla	A-Z random_char
1	E

Función que selecciona un carácter aleatorio de la cadena (+\$#()-!{}[]):

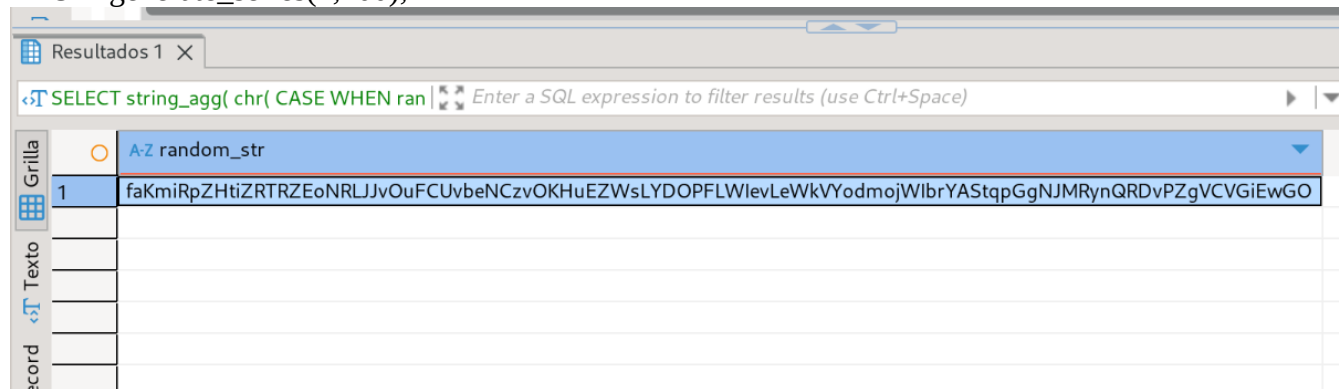
SELECT substr('(\$#()-!{}[])', floor(random() * length('(\$#()-!{}[])' + 1)::int, 1) AS random_char;



Grilla	A-Z random_char
1)

Generar cadena aleatoria de 100 caracteres que contenga letras mayúsculas y minúsculas

```
SELECT string_agg(  
    chr(  
        CASE WHEN random() < 0.5 THEN  
            trunc(65 + random() * 26)::int -- Mayúscula A-Z  
        ELSE  
            trunc(97 + random() * 26)::int -- Minúscula a-z  
        END  
    ), ""  
    ) AS random_str  
FROM generate_series(1,100);
```



Grilla	A-Z random_str
1	faKmiRpZHtiZRTRZEoNRLJvOuFCUvbeNCzvOKHuEZWslYDOPFLWlEvLeWkVYodmojWlbrYASqpGgNJMRynQRDvPZgVCVGiEwGO

Esta función genera una contraseña aleatoria compuesta por: 1. Una letra mayúscula aleatoria. 2. Un dígito aleatorio (entre 0 y 9). 3. Un símbolo especial aleatorio de una lista predefinida de símbolos. 4. Una cadena alfanumérica aleatoria de 4 caracteres. 5. Una cadena alfanumérica adicional aleatoria de longitud variable entre 1 y 12 caracteres.

```
CREATE OR REPLACE FUNCTION generatePassword()
RETURNS VARCHAR AS $$
DECLARE
    first_letter CHAR(1);
    second_digit CHAR(1);
    third_special_symbol CHAR(1);
    alphanumeric VARCHAR(4);
    others VARCHAR(12);
    symbols CONSTANT TEXT := '($#()-!{}{[]})';
    password VARCHAR;
    others_length INT;
BEGIN
    -- 1. Letra mayúscula aleatoria
    first_letter := chr(trunc(65 + random() * 26)::int);

    -- 2. Dígito aleatorio 0-9
    second_digit := chr(trunc(48 + random() * 10)::int); -- 48 es '0'

    -- 3. Símbolo especial aleatorio de la lista
    third_special_symbol := substr(symbols, floor(random() * length(symbols) + 1)::int, 1);

    -- 4. Cadena alfanumérica de 4 caracteres (mayúsculas y minúsculas)
    alphanumeric := (
        SELECT string_agg(
            chr(
                CASE WHEN random() < 0.5 THEN
                    trunc(65 + random() * 26)::int -- A-Z
                ELSE
                    trunc(97 + random() * 26)::int -- a-z
                END
            ), ''
        ) FROM generate_series(1,4)
    );

    -- 5. Longitud variable entre 1 y 12 para la cadena adicional
    others_length := floor(random() * 12 + 1)::int;

    others := (
        SELECT string_agg(
            chr(
                CASE WHEN random() < 0.5 THEN
                    trunc(65 + random() * 26)::int -- A-Z
                ELSE
                    trunc(97 + random() * 26)::int -- a-z
                END
            ), ''
        ) FROM generate_series(1, others_length)
    );

    password := first_letter || second_digit || third_special_symbol || alphanumeric || others;
END;
```

```

), "
) FROM generate_series(1, others_length)
);

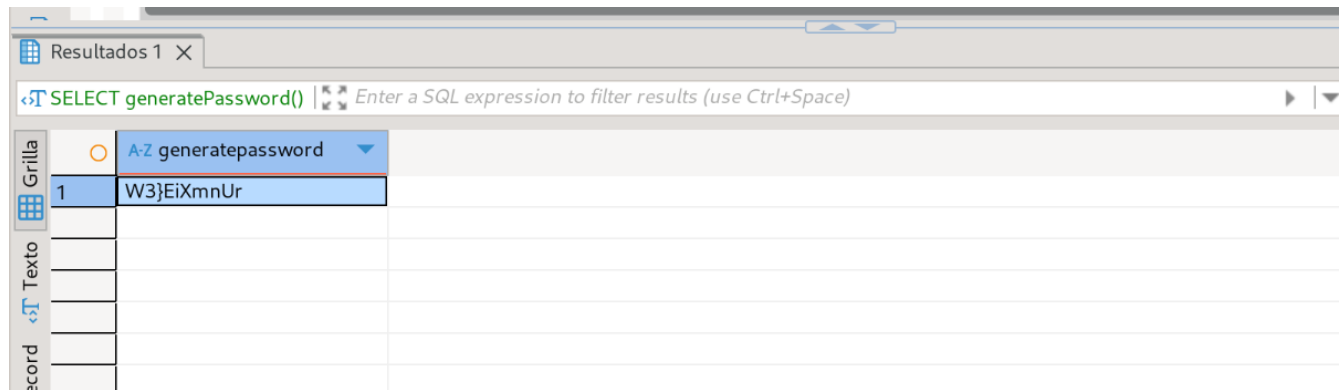
```

```
password := first_letter || second_digit || third_special_symbol || alphanumeric || others;
```

```
RETURN password;
```

```
END;
```

```
$$ LANGUAGE plpgsql;
```



Resultados 1 X	
SELECT generatePassword() Enter a SQL expression to filter results (use Ctrl+Space)	
Grilla	A-Z generatepassword
1	W3}EiXmnUr
Texto	
Record	

Figura 14: Para usar la función se le puede llamar con `SELECT generatePassword()`;

Procedimiento para llenar una tabla con 1 millón de registros con cadenas aleatorias de 100 caracteres

Primero se crea la tabla temporal.

```

CREATE TABLE tmp (
  id SERIAL PRIMARY KEY,
  campox VARCHAR(100)
);

```

Ahora se crea el procedimiento.

```

CREATE OR REPLACE PROCEDURE llenartmp()
LANGUAGE plpgsql

```

```
AS $$
```

```
DECLARE
```

```
  i INT := 0;
```

```
BEGIN
```

```
  FOR i IN 1..1000000 LOOP
```

```
    INSERT INTO tmp (campox)
```

```
    VALUES (
```

```
      (
```

```
        SELECT string_agg(
```

```
          chr(
```

```
            CASE WHEN random() < 0.5 THEN
```

```
              trunc(65 + random() * 26)::int -- A-Z
```

```
            ELSE
```

```
              trunc(97 + random() * 26)::int -- a-z
```

```
            END
```

```
          ), "
        )
      )
    )

```



```

        FROM generate_series(1, 100)
    )
);
END LOOP;
END;
$$;

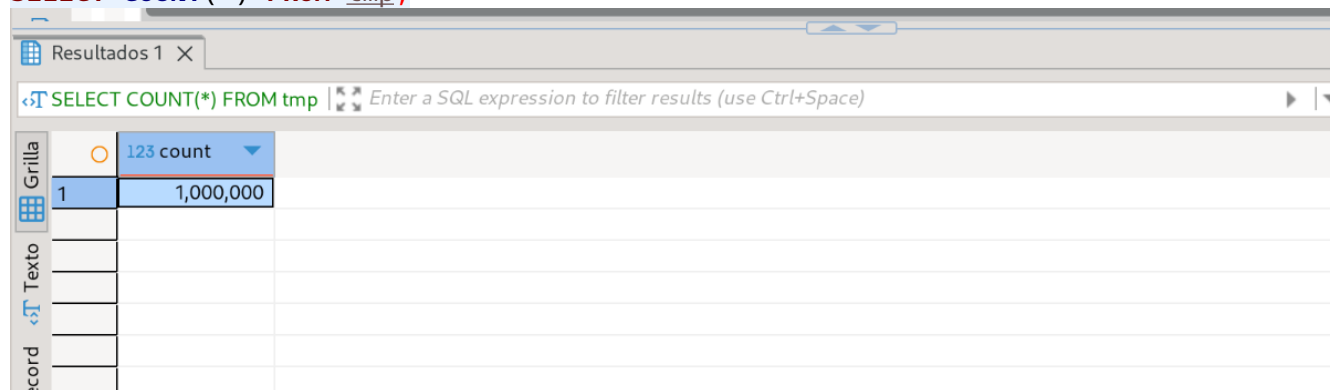
```

Y finalmente se ejecuta llamando con:

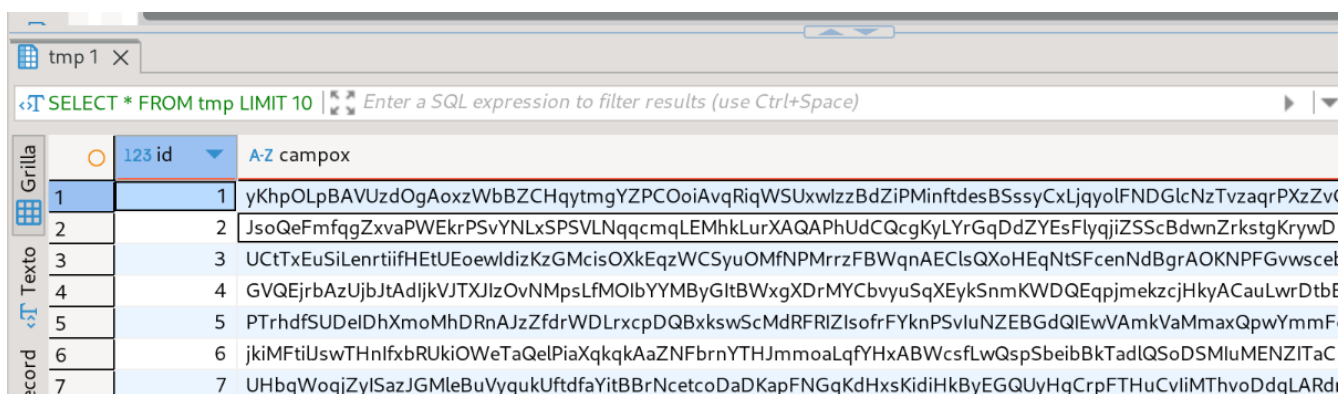
CALL llenartmp();

Para comprobar si realmente sirvió el procedimiento que hicimos podemos ejecutar:

```
SELECT COUNT(*) FROM tmp;
```



Resultados 1 X	
SELECT COUNT(*) FROM tmp	
123 count	1,000,000



tmp 1 X	
SELECT * FROM tmp LIMIT 10	
id	campox
1	yKhpOLpBAVUzdOgAoxzWbBZCHqytmgYZPCOoiAvqRiqWSUxwizzBdZiPMinfdesBSssyCxLjqyolFNDGlcNzTvzaqrPXzZv
2	JsoQeFmfqgZxvaPWEkrPSvYNLxSPSVLNqqcmqLEMhkLurXAQAPhUdCQcgKyLYrGqDdZYEsflyqjiZSScBdwnZrkstgKrywD
3	UCtTxEuSiLenrtiifHEtUEoewldizKzGMcisOXkEqzWCSyuOMfNPMrrzFBWqnAEClsQXoHEqNtSFcenNdBgrAOKNPFgvwscet
4	GVQEjrbAzUjbJtAdljkVJTXJlZOVNMpsLfMOlBYMBYgltBWxgXDrMYCbyyuSqXEykSnmKWDQEppmekzcjHkyACauLwrDtbE
5	PTrhdfSUDeIDhXmoMhDRnAJzZfdrWDLrxcPQDBxkswScMdRFRIZIsifrFYknPSvluNZEBGdQIEwVAmkVaMmaxQpwYmmF
6	jkiMFtiUswTHnlfxbRUKiOWeTaQelPiaXqkqkAaZNFbrnYTHJmmoaLqfYHxABWcsfLwQspSbeibBkTadIQSoDSMluMENZITaC
7	UHBqWoqjZylSazJGMeBuVyqukUftdfaYitBBRncetcoDaDKapFNGqKdHxsKidiHkByEGQUyHgCrpFTHuCVliMThvoDdqLARdi

Figura 15: Esto te mostrará los primeros 10 registros de la tabla para que no te aparezca toda la tabla si tiene muchos datos.

Empleados

Antes que nada crearemos las tablas que usaremos para las funciones y procedimientos.

```

CREATE TABLE empleados (
    id_empleado INT PRIMARY KEY,
    nombre VARCHAR(100),
    salario NUMERIC(10,2)
);

```

```

CREATE TABLE auditoria_empleados (
    id_empleado INT NOT NULL,
    accion VARCHAR(200) NOT NULL,

```

```
usuario VARCHAR(30) NOT NULL,  
fecha_hora TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

```
);
```

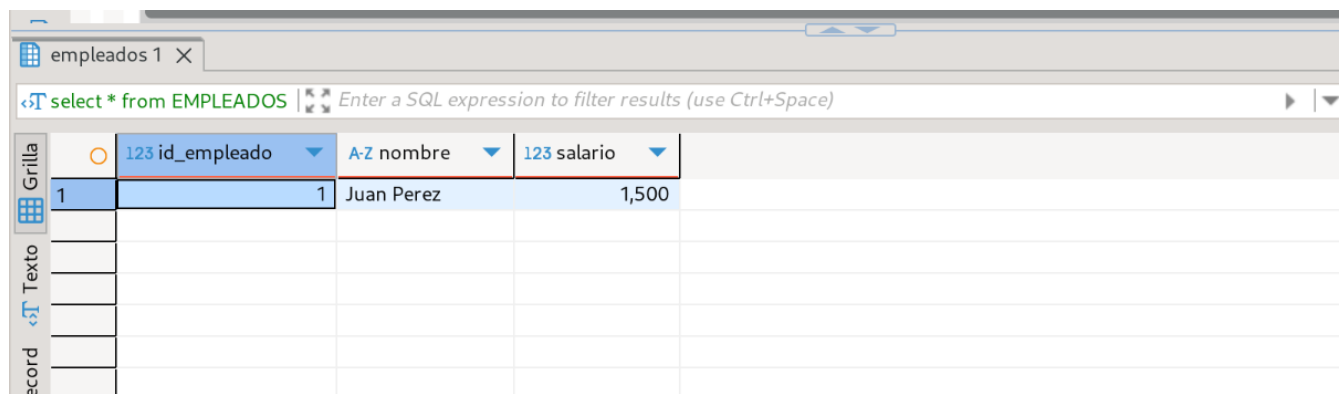
Ahora, si es posible iniciar con las funciones y procedimientos.

Agregar empleados

```
CREATE OR REPLACE FUNCTION agregar_empleado(  
    p_id INTEGER,  
    p_nombre VARCHAR,  
    p_salario NUMERIC  
) RETURNS VOID AS $$  
BEGIN  
    INSERT INTO empleados (id_empleado, nombre, salario)  
    VALUES (p_id, p_nombre, p_salario);  
END;  
$$ LANGUAGE plpgsql;
```

Para agregar empleados debemos llamar a la función con los parametros que nos pide.

```
SELECT agregar_empleado(1, 'Juan Perez', 1500);
```



The screenshot shows a database application window titled 'empleados 1'. The SQL query 'select * from EMPLEADOS' is entered in the query bar. The results are displayed in a table with columns 'id_empleado', 'nombre', and 'salario'. The first row shows '1', 'Juan Perez', and '1,500'.

	id_empleado	nombre	salario
1	1	Juan Perez	1,500

Figura 16: select * from EMPLEADOS;

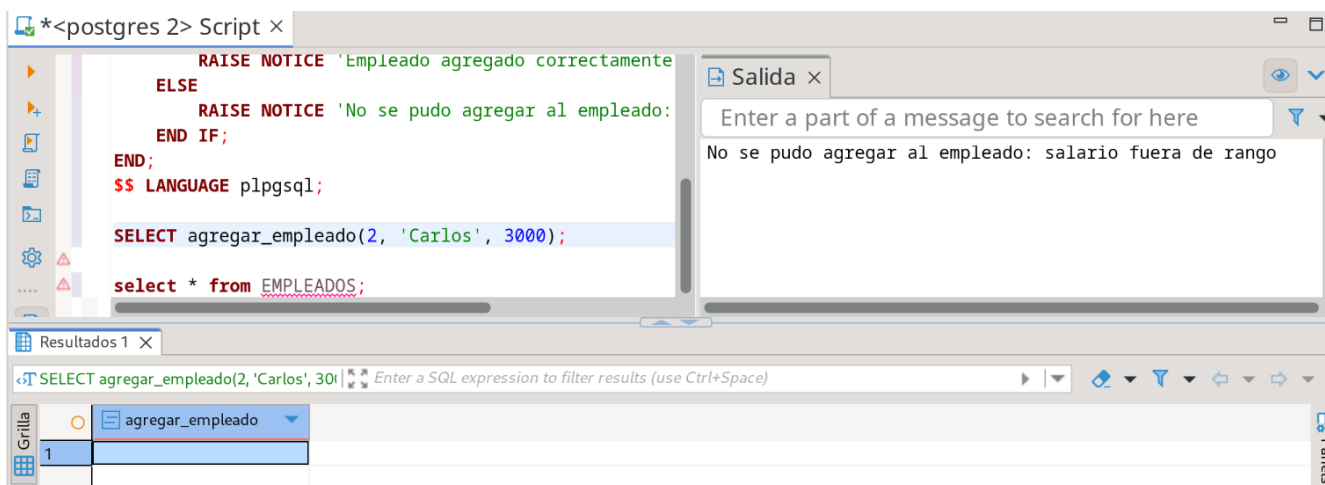
Procedimiento para agregar empleado con validación de salario

```
CREATE OR REPLACE FUNCTION agregar_empleado(  
    p_id INT,  
    p_nombre VARCHAR,  
    p_salario NUMERIC  
) RETURNS VOID AS $$  
BEGIN  
    IF p_salario >= 5000 AND p_salario <= 30000 THEN  
        INSERT INTO empleados (id_empleado, nombre, salario)  
        VALUES (p_id, p_nombre, p_salario);  
        RAISE NOTICE 'Empleado agregado correctamente';  
    ELSE  
        RAISE NOTICE 'No se pudo agregar al empleado: salario fuera de rango';  
    END IF;  
END;  
$$ LANGUAGE plpgsql;
```

Si quisieramos agregar a un empleado fuera de rango como:

```
SELECT agregar_empleado(2, 'Carlos', 3000);
```

No se podría debido a la condición.



Función para obtener salario de un empleado

```

CREATE OR REPLACE FUNCTION obtener_salario(p_id INT) RETURNS NUMERIC AS $$
DECLARE
    v_salario NUMERIC;
BEGIN
    SELECT salario INTO v_salario FROM empleados WHERE id_empleado = p_id;
    RETURN v_salario;
EXCEPTION WHEN NO_DATA_FOUND THEN
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

```

Para comprobar su efectividad podemos hacer una consulta de prueba.

`SELECT obtener_salario(1);`

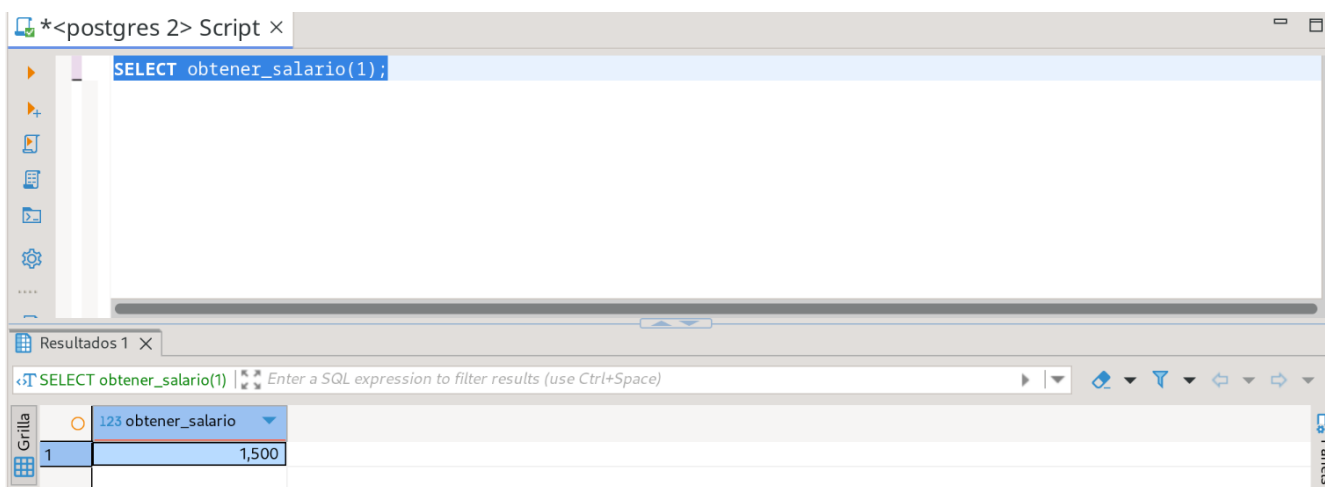


Figura 17: Nos muestra correctamente el salario que le asignamos al empleado Juan Perez, lo creamos antes de las condiciones.

Procedimiento para listar empleados

```

CREATE OR REPLACE FUNCTION listar_empleados() RETURNS VOID AS $$
DECLARE
    rec RECORD;
BEGIN
    FOR rec IN SELECT id_empleado, nombre FROM empleados LOOP
        RAISE NOTICE 'ID: %, Nombre: %', rec.id_empleado, rec.nombre;
    END LOOP;
END;
$$ LANGUAGE plpgsql;

```

```

END LOOP;
END;
$$ LANGUAGE plpgsql;

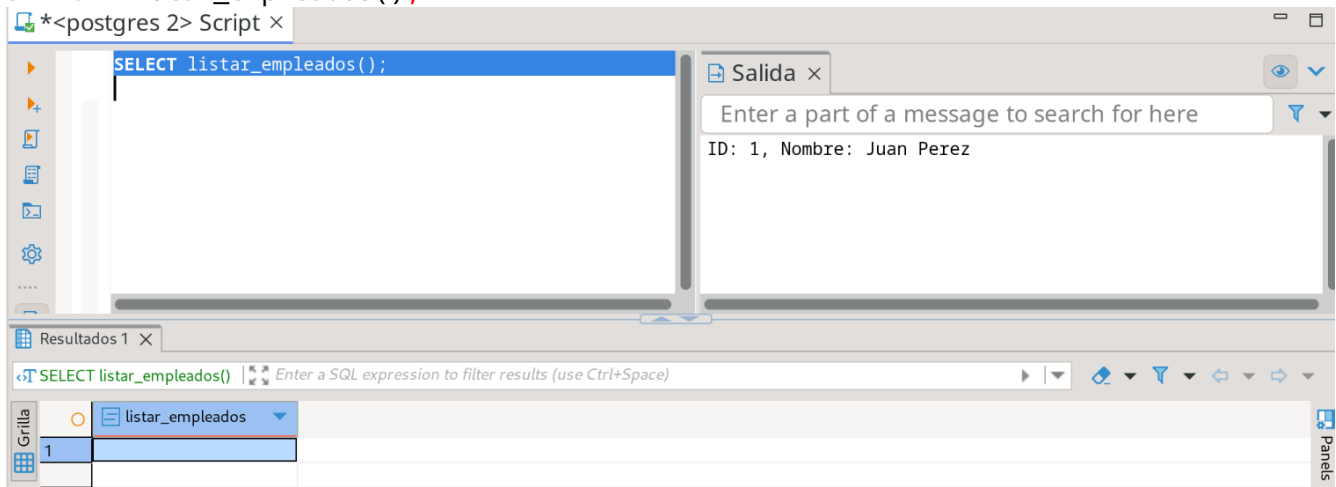
```

Para llamar a ese procedimiento se hace uso de

```

SELECT listar_empleados();

```



Procedimiento para auditar empleado

```

CREATE OR REPLACE FUNCTION auditar_empleado(
    p_id_empleado INT,
    p_accion VARCHAR
) RETURNS VOID AS $$
BEGIN
    INSERT INTO auditoria_empleados(id_empleado, accion, usuario)
    VALUES (p_id_empleado, p_accion, current_user);
END;
$$ LANGUAGE plpgsql;

```

Ejemplo: Podemos llamar al procedimiento para evaluar el desempeño del empleado con id 1, y ponerle un excelente desempeño.

```

SELECT auditar_empleado(1, 'REVISIÓN COMPLETA, EXCELENTE DESEMPEÑO');

```

	id_empleado	accion	usuario	fecha_hora
1	1	REVISIÓN COMPLETA, EXCELENTE DESEMPEÑO	postgres	2025-05-19 23:39:40.221
2	1	REVISIÓN COMPLETA, EXCELENTE DESEMPEÑO	postgres	2025-05-19 23:41:07.016
3	1	REVISIÓN COMPLETA, EXCELENTE DESEMPEÑO	postgres	2025-05-20 00:40:47.725
4	1	REVISIÓN COMPLETA, EXCELENTE DESEMPEÑO	postgres	2025-05-20 00:41:02.948

Figura 18: A mi me salen más porque ejecute el comando repetidas veces, pero si se modifica la tabla

TRIGGERS

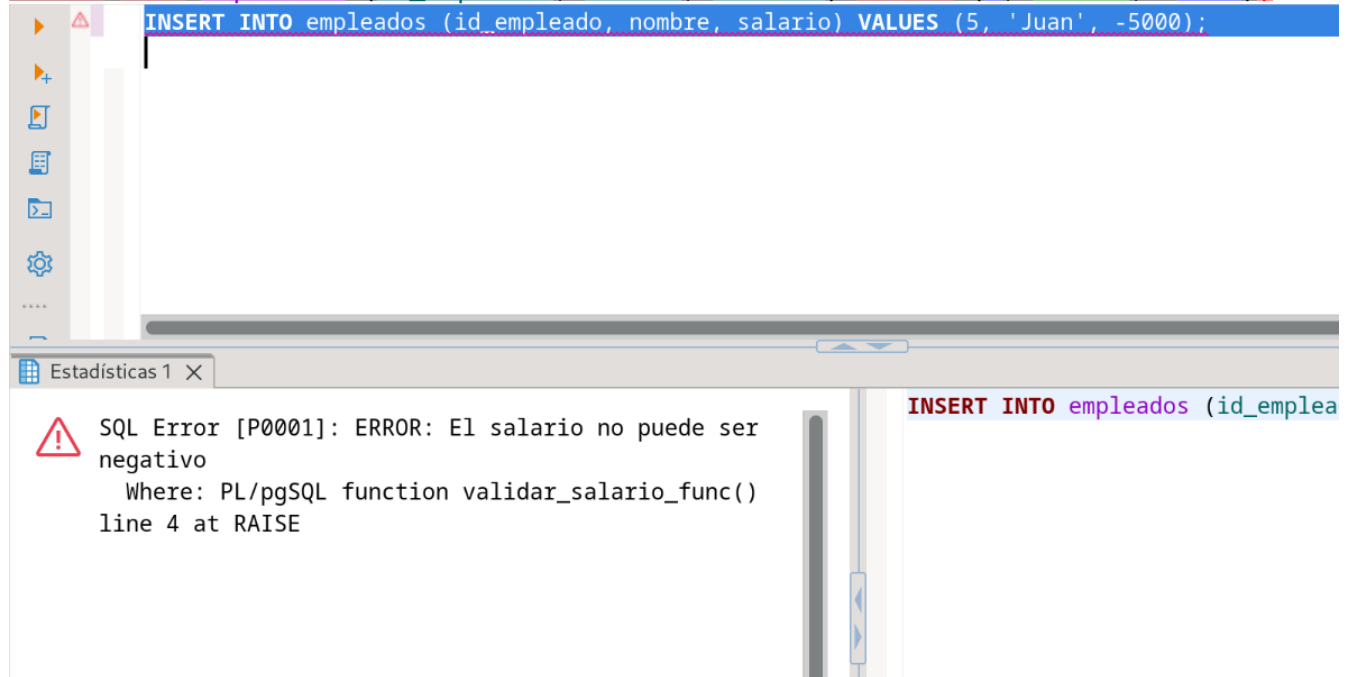
Trigger para validar que el salario no sea negativo antes de insertar

```
CREATE OR REPLACE FUNCTION validar_salario_func() RETURNS TRIGGER AS $$
BEGIN
    IF NEW.salario < 0 THEN
        RAISE EXCEPTION 'El salario no puede ser negativo';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER validar_salario
BEFORE INSERT ON empleados
FOR EACH ROW EXECUTE FUNCTION validar_salario_func();
```

Ejemplo: Vamos a insertar en la tabla empleados con los atributos ID_empleado, nombre y salario. Los nuevos valores serán 5, Juan, -5000.

```
INSERT INTO empleados (id_empleado, nombre, salario) VALUES (5, 'Juan', -5000);
```

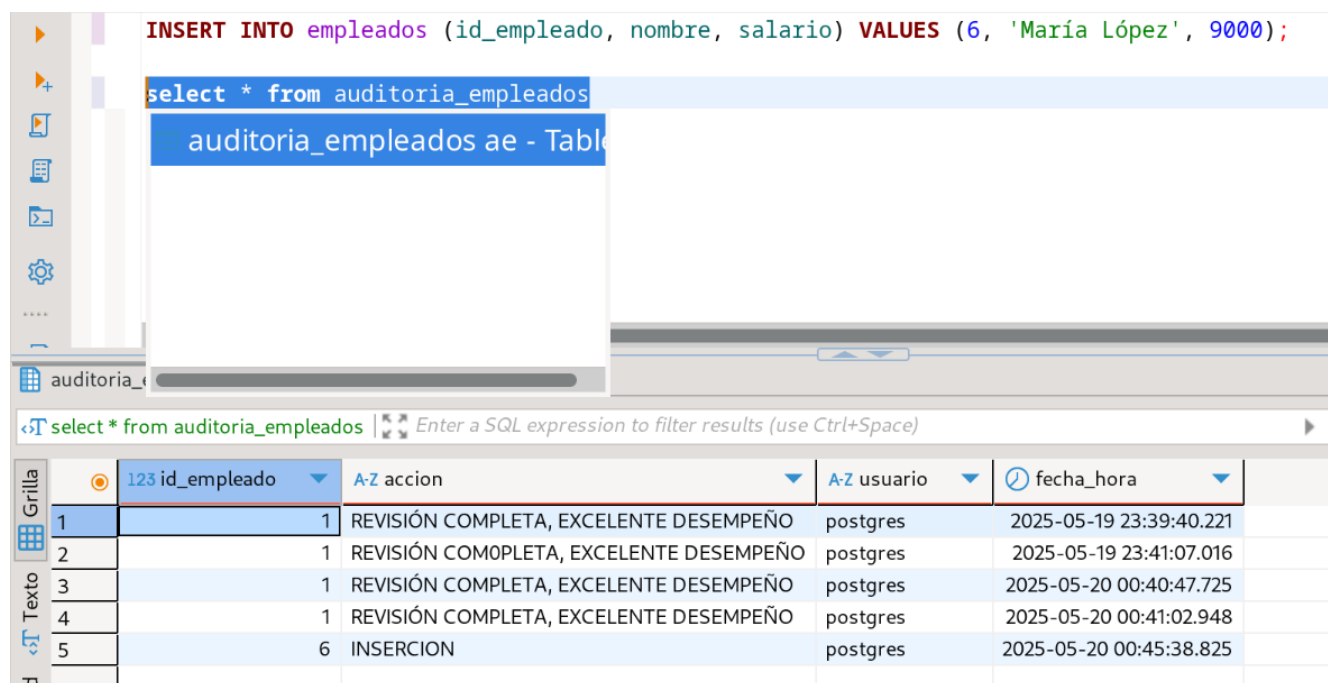


Trigger para auditar inserciones en empleados

```
CREATE OR REPLACE FUNCTION auditar_insercion_empleado_func() RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO auditoria_empleados(id_empleado, accion, usuario)
    VALUES (NEW.id_empleado, 'INSERCIÓN', current_user);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER auditar_insercion_empleado
AFTER INSERT ON empleados
FOR EACH ROW EXECUTE FUNCTION auditar_insercion_empleado_func();
```

Ejemplo: Agregar a un nuevo empleado con el ID: 5, Nombre: María López y un salario de 9000
 INSERT INTO empleados (id_empleado, nombre, salario) VALUES (5, 'María López', 9000);
INSERT INTO empleados (id_empleado, nombre, salario) VALUES (6, 'María López', 9000);



The screenshot shows a database management interface. At the top, a SQL query is entered: `INSERT INTO empleados (id_empleado, nombre, salario) VALUES (6, 'María López', 9000);`. Below the query, a dropdown menu shows `select * from auditoria_empleados`. The main area displays a table grid with the following data:

	123 id_empleado	A-Z accion	A-Z usuario	🕒 fecha_hora
1	1	REVISIÓN COMPLETA, EXCELENTE DESEMPEÑO	postgres	2025-05-19 23:39:40.221
2	1	REVISIÓN COMPLETA, EXCELENTE DESEMPEÑO	postgres	2025-05-19 23:41:07.016
3	1	REVISIÓN COMPLETA, EXCELENTE DESEMPEÑO	postgres	2025-05-20 00:40:47.725
4	1	REVISIÓN COMPLETA, EXCELENTE DESEMPEÑO	postgres	2025-05-20 00:41:02.948
5	6	INSERCIÓN	postgres	2025-05-20 00:45:38.825

El trigger se ejecuta correctamente, pues esta acción se registra en la tabla auditoria_empleados.

Trigger para auditar cambio de salario

```
CREATE OR REPLACE FUNCTION auditar_cambio_salario_func() RETURNS TRIGGER AS $$
BEGIN
    IF OLD.salario <> NEW.salario THEN
        INSERT INTO auditoria_empleados(id_empleado, accion, usuario)
        VALUES (OLD.id_empleado,
            'CAMBIO SALARIO DE ' || OLD.salario || ' A ' || NEW.salario,
            current_user);
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER auditar_cambio_salario
BEFORE UPDATE ON empleados
FOR EACH ROW EXECUTE FUNCTION auditar_cambio_salario_func();
```

Ejemplo: Vamos a cambiar el salario del empleado con ID 1 a una cantidad mayor, 15000. `UPDATE empleados SET salario = 15000 WHERE id_empleado = 1;`
`UPDATE empleados SET salario = 15000 WHERE id_empleado = 1;`

The screenshot shows a PostgreSQL script editor with a query window and a results window. The query window contains the SQL statement: `select * from auditoria_empleados;`. The results window displays the output of this query as a table with 6 rows and 5 columns: `id_empleado`, `accion`, `usuario`, and `fecha_hora`. The first row is highlighted in blue.

	id_empleado	accion	usuario	fecha_hora
1	1	REVISIÓN COMPLETA, EXCELENTE DESEMPEÑO	postgres	2025-05-19 23:39:40.221
2	1	REVISIÓN COMPLETA, EXCELENTE DESEMPEÑO	postgres	2025-05-19 23:41:07.016
3	1	REVISIÓN COMPLETA, EXCELENTE DESEMPEÑO	postgres	2025-05-20 00:40:47.725
4	1	REVISIÓN COMPLETA, EXCELENTE DESEMPEÑO	postgres	2025-05-20 00:41:02.948
5	6	INSERCIÓN	postgres	2025-05-20 00:45:38.825
6	1	CAMBIO SALARIO DE 1500.00 A 15000.00	postgres	2025-05-20 00:47:53.912

Figura 19: Se registra la acción en `auditoria_empleados`

Trigger para prevenir borrado de jefe ("Director General")

```
CREATE OR REPLACE FUNCTION prevenir_borrado_jefe_func() RETURNS TRIGGER AS $$
BEGIN
    IF OLD.nombre = 'Director General' THEN
        RAISE EXCEPTION 'No se puede borrar el Director General';
    END IF;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER prevenir_borrado_jefe
BEFORE DELETE ON empleados
FOR EACH ROW EXECUTE FUNCTION prevenir_borrado_jefe_func();
```

Ejemplo: Agregar e intentar eliminar un Director General.

```
INSERT INTO empleados (id_empleado, nombre, salario) VALUES (7, 'Director General', 50000);
```

*<postgres 2> Script x

```
INSERT INTO empleados (id_empleado, nombre, salario) VALUES (7, 'Director General', 50000);
select * from EMPLEADOS;
```

empleados 1 x

select * from EMPLEADOS | Enter a SQL expression to filter results (use Ctrl+Space)

	123 id_empleado	A-Z nombre	123 salario
1	6	María López	9,000
2	1	Juan Perez	15,000
3	7	Director General	50,000

Figura 20: Aqui se visualiza que en la tabla empleados ahora aparece el jefe

Si lo intentamos borrar.

DELETE FROM empleados WHERE id_empleado = 7;

*<postgres 2> Script x

```
INSERT INTO empleados (id_empleado, nombre, salario) VALUES (7, 'Director General', 50000);
select * from EMPLEADOS;
DELETE FROM empleados WHERE id_empleado = 7;
```

empleados 1 x

DELETE FROM empleados WHERE id_empleado = 7; | Enter a SQL expression to filter results (use Ctrl+Space)

SQL Error [P0001]: ERROR: No se puede borrar el Director General
Where: PL/pgSQL function prevenir_borrado_jefe_func() line 4 at RAISE

Detalles ↓

DELETE FROM empleados WHERE id

Figura 21: Error: No se puede borrar el Director General

Trigger a nivel statement para mostrar mensaje antes de update

```
CREATE OR REPLACE FUNCTION trigger_bloqueo_tabla_func() RETURNS TRIGGER AS $$  
BEGIN
```

```
    RAISE NOTICE 'Se está intentando actualizar la tabla EMPLEADOS';
```

```
    RETURN NULL;
```

```
END;
```

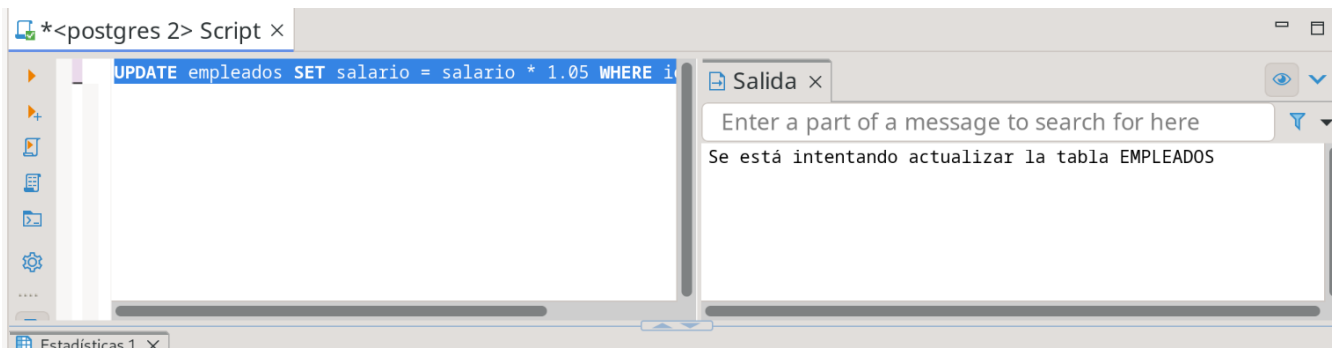
```
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER trigger_bloqueo_tabla
```

```
BEFORE UPDATE ON empleados
```

```
FOR EACH STATEMENT EXECUTE FUNCTION trigger_bloqueo_tabla_func();
```

Ejemplo: Si quisieramos actualizar el salario de un empleado, digamos el empleado con id 2. Al ejecutar la sentencia: `UPDATE empleados SET salario = salario * 1.05 WHERE id_empleado = 2;`



The screenshot shows a PostgreSQL IDE interface. The top pane, titled '*<postgres 2> Script x', contains the SQL query: `UPDATE empleados SET salario = salario * 1.05 WHERE id_empleado = 2;`. The bottom pane, titled 'Salida x', displays the output message: 'Se está intentando actualizar la tabla EMPLEADOS'. Below the main editor, a statistics window titled 'Estadísticas 1 x' provides execution details.

Name	Value
Updated Rows	0
Execute time	0.004s
Start time	Tue May 20 00:51:03 CST 2025
Finish time	Tue May 20 00:51:03 CST 2025
Query	UPDATE empleados SET salario = salario * 1.05 WHERE id_empleado = 2

Figura 22: Aparecerá en consola: Se está intentando actualizar la tabla EMPLEADOS