**PECL3**

**BBDD 2022/2023**

Adrián Rodríguez Hurtado – 09064004A

Víctor Sanavia Valdeolivas – 03202543T

# **ÍNDICE**

Contenido

[**ÍNDICE** 2](#_Toc123764224)

[**Librerías necesarias** 3](#_Toc123764225)

[**Creación de los disparadores que se necesiten para completar la lógica de negocio** 4](#_Toc123764226)

[**Creación de usuarios** 10](#_Toc123764227)

[**Conexión con programas externos y seguridad** 11](#_Toc123764228)

# **Librerías necesarias**

Todas las librerías utilizadas son propias de Python excepto una, “dotenv”, la cual es necesario descargarse. Para ello, hay que escribir en la terminal “pip install python-dotenv”.

# **Creación de los disparadores que se necesiten para completar la lógica de negocio**

**Trigger de auditoría**

Inicialmente, vamos a crear una tabla para almacenar las medias de todas las películas, y otra tabla de auditoría.

CREATE TABLE peliculas.nota\_media\_peliculas(

    titulo\_peliculas text,

    anno\_peliculas smallint,

    media integer

);

CREATE TABLE peliculas.auditoria(

    evento text,

    tabla name,

    usuario text,

    fecha timestamp

);

Hemos creado dos funciones auxiliares, una para dar permisos al crítico, y otra para quitárselos.

CREATE OR REPLACE FUNCTION peliculas.da\_permiso\_critico()

RETURNS void

SECURITY DEFINER

AS

$BODY$

BEGIN

GRANT INSERT ON peliculas.auditoria TO critico;

END;

$BODY$

LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION peliculas.quita\_permiso\_critico()

RETURNS void

SECURITY DEFINER

AS

$BODY$

BEGIN

REVOKE INSERT ON peliculas.auditoria FROM critico;

END;

$BODY$

LANGUAGE plpgsql;

El primer trigger es el trigger de auditoría, que contará con los eventos de inserción, modificación y borrado. Esta tabla registrará:

* La tabla en la que ha ocurrido el evento.
* El tipo de evento.
* El usuario que lo ha llevado a cabo.
* La fecha y la hora.

Vamos a crear el trigger:

CREATE OR REPLACE FUNCTION peliculas.fn\_auditoria() RETURNS TRIGGER AS $fn\_auditoria$

BEGIN

    PERFORM peliculas.da\_permiso\_critico();

    IF TG\_OP = 'INSERT' THEN INSERT INTO peliculas.auditoria(evento, tabla, usuario, fecha) VALUES ('INSERT', TG\_RELNAME, current\_user, now());

    ELSIF TG\_OP = 'UPDATE' THEN INSERT INTO peliculas.auditoria(evento, tabla, usuario, fecha) VALUES ('UPDATE', TG\_RELNAME, current\_user, now());

    ELSIF TG\_OP = 'DELETE' THEN INSERT INTO peliculas.auditoria(evento, tabla, usuario, fecha) VALUES ('DELETE', TG\_RELNAME, current\_user, now());

    END IF;

    PERFORM peliculas.quita\_permiso\_critico();

    RETURN NEW;

END;

$fn\_auditoria$ LANGUAGE plpgsql;

CREATE TRIGGER tg\_auditoria

    AFTER INSERT OR UPDATE OR DELETE

    ON peliculas.criticas

    FOR EACH ROW

    EXECUTE PROCEDURE peliculas.fn\_auditoria();

**Trigger de críticas**

El segundo trigger consiste está relacionado con las críticas. Debe comprobar si, al insertar una crítica, la página web se encuentra o no en la tabla. Si ésta no se encuentra, se añadirá a la tabla.

Primero, vamos a comprobar que el usuario ha introducido todos los atributos necesarios.

CREATE OR REPLACE FUNCTION peliculas.fn\_inserta\_critica() RETURNS TRIGGER SECURITY DEFINER AS $fn\_inserta\_critica$

BEGIN

    IF NEW.critico ISNULL THEN

        RAISE EXCEPTION 'El nombre del critico no puede ser nulo, debes imprimir uno';

    END IF;

    IF NEW.anno\_peliculas ISNULL THEN

        RAISE EXCEPTION 'El anno de la pelicula no puede ser nulo, debes aportar uno';

    END IF;

    IF NEW.titulo\_peliculas ISNULL THEN

        RAISE EXCEPTION 'El titulo de la pelicula no puede tener valor nulo, debes aportar uno';

    END IF;

    IF NEW.nombre\_pag\_web ISNULL THEN

        RAISE EXCEPTION 'El nombre o url de la pagina web no puede ser nulo, debes aportar uno';

    END IF;

Si el usuario ha introducido todos los datos, se procederá a comprobar si la página web se encuentra o no en la tabla.

IF NEW.nombre\_pag\_web NOT IN (SELECT nombre FROM peliculas.pag\_web) THEN

        INSERT INTO peliculas.pag\_web(nombre) VALUES (NEW.nombre\_pag\_web);

    END IF;

    RETURN NEW;

END;

$fn\_inserta\_critica$ LANGUAGE plpgsql;

Por último, creamos el trigger:

CREATE TRIGGER tg\_inserta\_critica

    BEFORE INSERT

    ON peliculas.criticas

    FOR EACH ROW

    EXECUTE FUNCTION peliculas.fn\_inserta\_critica();

**Trigger de media**

Para este tercer y último trigger se creará una nueva tabla para almacenar las medias de las películas. La tabla deberá de ser actualizada cada vez que se introduzca una nueva media.

Primero, actualizaremos la media de las películas a las que se la haya añadido una nueva calificación.

CREATE OR REPLACE FUNCTION peliculas.fn\_actualiza\_media\_peliculas() RETURNS TRIGGER SECURITY DEFINER AS $fn\_actualiza\_media\_peliculas$

BEGIN

    UPDATE peliculas.nota\_media\_peliculas SET (media) =

    (SELECT puntuacion\_media

    FROM peliculas.media\_peliculas

    WHERE (peliculas.media\_peliculas.titulo\_peliculas = NEW.titulo\_peliculas) and (peliculas.media\_peliculas.anno\_peliculas = NEW.anno\_peliculas));

    RETURN NEW;

END;

Se crea el trigger que se disparará para actualizar la media de las películas.

CREATE TRIGGER tg\_actualiza\_medias\_peliculas

    AFTER INSERT

    ON peliculas.criticas

    FOR EACH ROW

    EXECUTE FUNCTION peliculas.fn\_actualiza\_media\_peliculas();

Se consigue una vista auxiliar de las medias de todas las películas.

CREATE VIEW peliculas.media\_peliculas as

(SELECT titulo\_peliculas, anno\_peliculas, avg(puntuacion) as puntuacion\_media

FROM

peliculas.criticas

GROUP BY

titulo\_peliculas, anno\_peliculas

ORDER BY

avg(puntuacion));

Utilizamos la vista auxiliar para poblar la tabla de las medias.

INSERT INTO peliculas.nota\_media\_peliculas(titulo\_peliculas, anno\_peliculas, media)

SELECT titulo\_peliculas, anno\_peliculas, puntuacion\_media FROM peliculas.media\_peliculas;

# **Creación de usuarios**

El programa constará de 4 tipos distintos de usuarios:

* Administrador -> Cualquier operación.
* Gestor -> Inserción, modificación, borrado y consulta.
* Crítico -> Consulta de todas las tablas e inserción en la tabla “críticas”.
* Cliente -> Consulta.

Primero, crearemos los usuarios.

CREATE USER admin PASSWORD 'admin';

CREATE USER gestor PASSWORD 'gestor';

CREATE USER critico PASSWORD 'critico';

CREATE USER cliente PASSWORD 'cliente';

Una vez creados, les asignamos los permisos en función del rol.

REVOKE ALL PRIVILEGES ON ALL TABLES IN SCHEMA peliculas FROM gestor, critico, cliente;

GRANT USAGE ON SCHEMA peliculas TO admin, gestor, critico, cliente;

GRANT SELECT ON ALL TABLES IN SCHEMA peliculas TO critico;

GRANT INSERT ON peliculas.criticas TO critico;

GRANT SELECT ON ALL TABLES IN SCHEMA peliculas TO cliente;

GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA peliculas TO admin WITH GRANT OPTION;

GRANT INSERT, UPDATE, DELETE, SELECT ON ALL TABLES IN SCHEMA peliculas TO gestor;

# **Conexión con programas externos y seguridad**

Se ha creado un programa en Python para que cualquier usuario creado previamente pueda conectarse a la base de datos. El programa solicitará el usuario, su respectiva contraseña y la consulta a resolver o la crítica a insertar.

Primero, se llevará a cabo la elección del usuario, el cual tendrá que elegir uno de los 4 usuarios creados previamente, e introducir la contraseña asignada a cada uno. Si la contraseña es correcta, el programa mostrará el texto 'Contrasena correcta!'. Si ésta fuese incorrecta, no se podría asignar el usuario correspondiente.

def user\_choice() -> str:

    global user

    correct\_choice = False

    while (not correct\_choice):

        try:

            print('\t\t-----------------=[ELECCION DE USUARIO]=-----------------\n\n\t1. Admin\n\t2. Gestor\n\t3. Critico\n\t4. Cliente\n\n\t\t-----------------=[ELECCION DE USUARIO]=-----------------\n\n')

            choice = int(input('Elige el usuario con el que deseas conectarte a la base de datos (numero): '))

            if (choice.\_\_eq\_\_(1)):

                password = str(input('Introduce la contrasena del usuario admin: '))

                if (password.\_\_eq\_\_(os.getenv('admin\_password'))):

                    user = 'admin'

                    correct\_choice = True

                else:

                    print('Contrasena incorrecta para el usuario admin')

                    time.sleep(2)

            elif (choice.\_\_eq\_\_(2)):

                password = str(input('Introduce la contrasena del usuario gestor: '))

                if (password.\_\_eq\_\_(os.getenv('gestor\_password'))):

                    user = 'gestor'

                    correct\_choice = True

                else:

                    print('Contrasena incorrecta para el usuario gestor')

                    time.sleep(2)

            elif (choice.\_\_eq\_\_(3)):

                password = str(input('Introduce la contrasena del usuario critico: '))

                if (password.\_\_eq\_\_(os.getenv('critico\_password'))):

                    user = 'critico'

                    correct\_choice = True

                else:

                    print('Contrasena incorrecta para el usuario critico')

                    time.sleep(2)

            elif (choice.\_\_eq\_\_(4)):

                password = str(input('Introduce la contrasena del usuario cliente: '))

                if (password.\_\_eq\_\_(os.getenv('cliente\_password'))):

                    user = 'cliente'

                    correct\_choice = True

                else:

                    print('Contrasena incorrecta para el usuario cliente')

                    time.sleep(2)

            else:

                print('Introduce una opcion correcta (numero del 1 al 4)')

                time.sleep(2)

        except ValueError:

            print('Introduce una opcion valida (numero del 1 al 4)')

        os.system('cls')

    print('Contrasena correcta!, estableciendo conexion...')

    time.sleep(2)

    user\_info = os.getenv(user + '\_user')

    return user\_info

Se crearán dos funciones para establecer conexión con la base de datos y otra para cerrar la conexión

def connection\_establishment(user\_info: str):

    global connection

    global cursor

    connection = psycopg2.connect(user\_info)

    cursor = connection.cursor()

    print('\nConexion establecida correctamente!\n')

    time.sleep(1)

def connection\_termination():

    global connection

    global cursor

    cursor.close()

    connection.close()

    print('Conexion cerrada.')

Se crea una función para elegir el tipo de consulta entre select e insert.

def query\_choice() -> int:

    possible\_results = [1, 2]

    print('\t\t-----------------=[ELECCION DE TIPO DE CONSULTA]=-----------------\n\n\t1. Consulta de tipo select\n\t2. Consulta de tipo insert\n\n')

    correct\_result = False

    while (not correct\_result):

        try:

            result = int(input('Elige la opción deseada (introduciendo el numero de la opcion): '))

            if (result in possible\_results):

                correct\_result = True

        except ValueError:

            print(f'Introduce un valor valido ({possible\_results})')

    return result

Se crea una función para la consulta select

def select\_query(sql\_command: str):

    global connection

    global cursor

    try:

        cursor.execute(sql\_command)

        rows = cursor.fetchall()

        print('\n\nConsulta realizada correctamente, mostrando resultados...\n\n')

        time.sleep(1)

        for row in rows:

            print(row)

    except (errors.UndefinedTable) as undefined\_table:

        print(f'\n\nLa tabla que has introducido no existe -> {undefined\_table}')

    except (errors.UndefinedColumn) as undefined\_column:

        print(f'\n\nLa columna introducida no existe -> {undefined\_column}')

    except (errors.InsufficientPrivilege) as permission\_error:

        print(f'\n\nEl usuario elegido {user} no tiene permisos para realizar esta accion -> {permission\_error}')

    except (errors.SyntaxError) as syntax\_error:

        print(f'\n\nError en la sintaxis de la consulta SQL -> {syntax\_error}')

Se crea una función para la consulta insert.

def insert\_query(sql\_command: str):

    global connection

    global cursor

    try:

        cursor.execute(sql\_command)

        connection.commit()

        print('\nConsulta realizada correctamente!')

    except (errors.UndefinedTable) as undefined\_table:

        print(f'\n\nLa tabla que has introducido no existe -> {undefined\_table}')

    except (errors.UndefinedColumn) as undefined\_column:

        print(f'\n\nLa columna introducida no existe -> {undefined\_column}')

    except (errors.InsufficientPrivilege) as permission\_error:

        print(f'\n\nEl usuario elegido {user} no tiene permisos para realizar esta accion -> {permission\_error}')

    except (errors.UniqueViolation) as unique\_violation:

        print(f'\n\nLa consulta viola una restriccion de unicidad (llave ya existente) -> {unique\_violation}')

    except (errors.ForeignKeyViolation) as foreign\_key\_violation:

        print(f'\n\nViolacion de clave foranea (no esta presente en la tabla) -> {foreign\_key\_violation}')

    except (errors.SyntaxError) as syntax\_error:

        print(f'\n\nError en la sintaxis de la consulta SQL -> {syntax\_error}')

Se crea una función para comprobar si el usuario va a querer hacer más consultas o quiere finalizar ya.

def more\_querys() -> bool:

    result: bool

    choice\_succeded = False

    choice = ''

    while (not choice\_succeded):

        try:

            choice = str(input('\n\n¿Desea hacer mas consultas? (si / no): '))

            if (choice.\_\_eq\_\_('si')):

                result = True

                choice\_succeded = True

            elif (choice.\_\_eq\_\_('no')):

                result = False

                choice\_succeded = True

        except ValueError:

            print('\n\nIntroduce una opcion valida (si / no)')

    return result

Por último, una función main que llama al resto de funciones creadas.

def main():

    try:

        os.system('cls')

        running = True

        while (running):

            try:

                connection\_establishment(user\_choice())

            except (psycopg2.OperationalError):

                print('Error en la conexion a la base de datos')

                running = False

                print('\n\n\t\t-----------------=[PROGRAMA FINALIZADO POR FALLO DE CONEXION]=-----------------\n\n')

                break

            query\_type = query\_choice()

            if (query\_type.\_\_eq\_\_(1)):

                sql\_command = str(input('Introduce la consulta a realizar (tipo select): '))

                select\_query(sql\_command)

            elif (query\_type.\_\_eq\_\_(2)):

                sql\_command = str(input('Introduce la consulta a realizar (tipo insert): '))

                insert\_query(sql\_command)

            if more\_querys().\_\_eq\_\_(False):

                print('Cerrando...')

                connection\_termination()

                time.sleep(0.5)

                print('\n\t\t-----------------=[PROGRAMA FINALIZADO]=-----------------\n\n')

                running = False

    except KeyboardInterrupt:

        print('\n\n\t\t-----------------=[PROGRAMA FINALIZADO POR TECLADO]=-----------------\n\n')

Por último, se llama a la función main.

main()