Bases de datos
Grado en Ingeniería de Computadores
Universidad de Alcalá

PECL3

Jose Fernández López

ÍNDICE

CREACIÓN DE TRIGGERS Y FUNCTIONS Y LÓGICA INTERNA

· AUDITORÍA

```
CREATE TABLE IF NOT EXISTS auditoria(
    tabla TEXT NOT NULL,
    evento TEXT NOT NULL,
    usuario TEXT NOT NULL,
    fecha TIMESTAMP NOT NULL DEFAULT NOW()
);
```

Este comando crea una nueva tabla llamada "registro_auditoria" si aún no existe. La tabla tiene cuatro columnas: "tabla", "evento", "usuario" y "fecha". La columna "tabla" almacena el nombre de la tabla en la que se ha realizado una operación, "evento" almacena el tipo de operación (inserción, eliminación o actualización), "usuario" almacena el nombre de usuario que ha realizado la operación y "fecha" almacena la fecha y hora en la que se realizó la operación.

La columna "fecha" tiene un valor predeterminado de NOW(), lo que significa que se insertará automáticamente la fecha y hora actual en cada fila creada.

```
CREATE OR REPLACE FUNCTION fn_auditoria() RETURNS TRIGGER AS $fn_auditoria$
DECLARE
BEGIN
    INSERT INTO registro_auditoria VALUES(TG_TABLE_NAME, TG_OP, current_user);
    RETURN NULL;
END;
$fn_auditoria$ LANGUAGE plpgsql;
```

Este comando crea una función llamada "fn_auditoria" si aún no existe, o la reemplaza si ya existe. La función es de tipo TRIGGER y se utiliza para realizar inserciones en la tabla "registro_auditoria".

La función obtiene el nombre de la tabla en la que se ha realizado una operación de "TG_TABLE_NAME", el tipo de operación de "TG_OP" y el nombre de usuario actual de "current_user" y los inserta en la tabla "registro_auditoria".

```
CREATE OR REPLACE TRIGGER tg_auditoria AFTER INSERT OR DELETE OR UPDATE ON peliculas.actor
FOR EACH STATEMENT
EXECUTE PROCEDURE fn_auditoria();

CREATE OR REPLACE TRIGGER tg_auditoria AFTER INSERT OR DELETE OR UPDATE ON peliculas.actua
FOR EACH STATEMENT
EXECUTE PROCEDURE fn_auditoria();

CREATE OR REPLACE TRIGGER tg_auditoria AFTER INSERT OR DELETE OR UPDATE ON peliculas.caratulas
FOR EACH STATEMENT
EXECUTE PROCEDURE fn_auditoria();

CREATE OR REPLACE TRIGGER tg_auditoria AFTER INSERT OR DELETE OR UPDATE ON peliculas.criticas
FOR EACH STATEMENT
EXECUTE PROCEDURE fn_auditoria();

CREATE OR REPLACE TRIGGER tg_auditoria AFTER INSERT OR DELETE OR UPDATE ON peliculas.director
FOR EACH STATEMENT
EXECUTE PROCEDURE fn_auditoria();

CREATE OR REPLACE TRIGGER tg_auditoria AFTER INSERT OR DELETE OR UPDATE ON peliculas.generos
FOR EACH STATEMENT
EXECUTE PROCEDURE fn_auditoria();

CREATE OR REPLACE TRIGGER tg_auditoria AFTER INSERT OR DELETE OR UPDATE ON peliculas.pag_web
FOR EACH STATEMENT
EXECUTE PROCEDURE fn_auditoria();

CREATE OR REPLACE TRIGGER tg_auditoria AFTER INSERT OR DELETE OR UPDATE ON peliculas.peliculas
FOR EACH STATEMENT
EXECUTE PROCEDURE fn_auditoria();

CREATE OR REPLACE TRIGGER tg_auditoria AFTER INSERT OR DELETE OR UPDATE ON peliculas.personal
FOR EACH STATEMENT
EXECUTE PROCEDURE fn_auditoria();
```

Este comando crea un trigger llamado "tg_auditoria" si aún no existe o lo reemplaza si ya existe. El trigger se activa después de realizar operaciones de inserción, eliminación o actualización en la tabla "actor" del esquema "peliculas".

El trigger se ejecuta después de cada sentencia en esta tabla y ejecuta la función "fn_auditoria", lo que provoca que se registre la información en la tabla "registro_auditoria”.

· INSERCIÓN DE PÁGINAS WEB

```
CREATE OR REPLACE FUNCTION fn_comprobarweb() RETURNS TRIGGER AS $fn_comprobarweb$
DECLARE
BEGIN
    IF (NEW.nombre_pag_web NOT IN (SELECT nombre FROM peliculas.pag_web)) THEN
        INSERT INTO peliculas.pag_web VALUES(NEW.nombre_pag_web, NULL, NULL);
    END IF;
    RETURN NULL;
END;
$fn_comprobarweb$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER tg_comprobarweb BEFORE INSERT ON peliculas.criticas
FOR EACH ROW
EXECUTE PROCEDURE fn_comprobarweb();
```

Esta función se utiliza como un desencadenador para la tabla "peliculas.criticas".

La función se ejecuta antes de que se ejecute una instrucción INSERT en la tabla "peliculas.criticas" y comprueba si el campo "nombre_pag_web" de la nueva fila ya está presente en la tabla "peliculas.pag_web".

Si no está presente, se agrega una nueva fila a la tabla "peliculas.pag_web" con el valor del campo "nombre_pag_web", y los otros campos se establecen en NULL.

También se crea un trigger llamado "tg_comprobarweb" que se ejecuta antes de insertar una nueva fila en la tabla "peliculas.criticas" y ejecuta la función fn_comprobarweb.

· PUNTUACIÓN MEDIA

```sql
CREATE TABLE IF NOT EXISTS puntuacion_media(
    puntuacion SMALLINT NOT NULL,
    año_pelicula SMALLINT,
    nombre_pelicula character varying,
    FOREIGN KEY (año_pelicula, nombre_pelicula) REFERENCES peliculas.peliculas(año, titulo)
);

INSERT INTO puntuacion_media
SELECT DISTINCT AVG(puntuacion), año_peliculas, titulo_peliculas
FROM peliculas.criticas
GROUP BY año_peliculas, titulo_peliculas
EXCEPT
SELECT puntuacion, año_pelicula, nombre_pelicula
FROM puntuacion_media;

CREATE OR REPLACE FUNCTION fn_puntuacion() RETURNS TRIGGER AS $fn_puntuacion$
DECLARE
BEGIN
    IF TG_OP = 'INSERT' THEN
        INSERT INTO puntuacion_media VALUES(AVG(NEW.puntuacion), año_peliculas, titulo_peliculas);
    ELSE
        WITH avg_puntuacion AS (
            SELECT AVG(puntuacion) AS rating, año_peliculas, titulo_peliculas
            FROM peliculas.criticas
            WHERE año_peliculas = NEW.año_peliculas AND titulo_peliculas = NEW.titulo_peliculas
        )

        INSERT INTO peliculas.rating_average (rating, año_peliculas, titulo_peliculas)
        SELECT rating, año_peliculas, titulo_peliculas
        FROM avg_puntuacion
        ON CONFLICT (año_peliculas, titulo_peliculas) DO UPDATE
        SET rating = avg_puntuacion.rating;
    END IF;
    RETURN NEW;
END;
$fn_puntuacion$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER tg_puntuacion AFTER INSERT OR UPDATE ON peliculas.criticas
FOR EACH STATEMENT
EXECUTE PROCEDURE fn_puntuacion();
```

Este código está creando una tabla llamada "puntuacion_media" si aún no existe. La tabla tiene tres columnas: "puntuacion" (un entero pequeño), "año_pelicula" (también un entero pequeño) y "nombre_pelicula" (un caracter variable). La columna "año_pelicula" y "nombre_pelicula" tienen una clave externa que hace referencia a la tabla "peliculas" en las columnas "año" y "titulo" respectivamente.

Luego, el código está insertando en la tabla "puntuacion_media" los valores promedio de puntuacion, año_peliculas y titulo_peliculas de la tabla "peliculas.criticas" , excepción hecha de los valores que ya están en la tabla "puntuacion_media"

Después se está creando una función llamada "fn_puntuacion()" que es un "trigger" (disparador) que se activa después de una inserción o actualización en la tabla "peliculas.criticas". La función calcula el promedio de puntuacion, año_peliculas y titulo_peliculas de la tabla "peliculas.criticas" y lo inserta en la tabla "puntuacion_media". Si ya existe una entrada con esos valores en la tabla "puntuacion_media", entonces actualiza el valor de "rating" con el nuevo promedio calculado.

Finalmente, se crea un trigger llamado "tg_puntuacion" que se activa después de una inserción o actualización en la tabla "peliculas.criticas" y ejecuta la función "fn_puntuacion()" cada vez que se activa.

CREACIÓN DE USUARIOS Y ESTABLECIMIENTO DE PRIVILEGIOS

```
CREATE USER admin WITH LOGIN SUPERUSER PASSWORD '1234';

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

CREATE USER critico;
GRANT SELECT ON ALL TABLES IN SCHEMA peliculas TO critico;
GRANT INSERT ON peliculas.criticas TO critico;

CREATE USER cliente;
GRANT SELECT ON ALL TABLES IN SCHEMA peliculas TO cliente;
```

Este código crea cuatro usuarios diferentes en una base de datos, cada uno con diferentes permisos y roles.

El primer comando "CREATE USER admin WITH LOGIN SUPERUSER PASSWORD '1234';" crea un usuario llamado "admin" con el rol de superusuario y una contraseña establecida como "1234".

El segundo comando "CREATE USER gestor" crea un usuario llamado "gestor", y posteriormente se le otorgan permisos de selección, inserción, actualización y eliminación en todas las tablas en el esquema "películas"

El tercer comando "CREATE USER critico" crea un usuario llamado "critico" con permisos de selección en todas las tablas y permisos de inserción en la tabla "criticas" en el esquema "peliculas" al usuario "critico".

El cuarto comando "CREATE USER cliente;" crea un usuario llamado "cliente" y posteriormente se le otorgan permisos de selección en todas las tablas en el esquema "peliculas".

LOGIN Y CONSULTAS DESDE PYTHON CON AUTENTICACIÓN

```python
import psycopg2

def main():

    conn_string = 'host=localhost port=5434 user=admin password=1234 dbname=peliculas'
    conn = psycopg2.connect(conn_string)

    cursor = conn.cursor()

    print("--------------------------------")
    print(" CONSULTA 1")
    print("--------------------------------")

    cursor.execute("SELECT a.nombre_personal\n"
        + "FROM peliculas.actor AS a\n"
        + "INNER JOIN peliculas.director AS d\n"
        + "ON a.nombre_personal = d.nombre_personal")
    for nombre in cursor.fetchall():
        print(nombre)

    print("--------------------------------")
    print(" CONSULTA 2")
    print("--------------------------------")

    cursor.execute("SELECT g.genero, count(g.titulo_peliculas)\n"
        + "FROM peliculas.generos AS g\n"
        + "GROUP BY g.genero\n"
        + "ORDER BY count(g.titulo_peliculas) DESC")
    for genero, n in cursor.fetchall():
        print(genero, n)

    cursor.close
    conn.close

main()
```

Este script se conecta a nuestra base de datos utilizando la biblioteca psycopg2. Utiliza una cadena de conexión para especificar los detalles de la conexión, como el host, el puerto, el nombre de usuario, la contraseña y el nombre de la base de datos. En este caso usaremos nuestra base de datos alojada en localhost y nos identificaremos con los credenciales generados en el apartado 2.

Luego, crea un cursor y utiliza el método execute para ejecutar dos consultas SQL.

La primera consulta selecciona el nombre de todas las personas que son tanto actores como directores.

La segunda consulta cuenta el número de películas por género y los ordena de manera descendente.

Finalmente, imprime los resultados de cada consulta.

Al finalizar, cierra el cursor y la conexión a la base de datos.

Aquí un ejemplo de la ejecución del programa:

```
PS C:\Users\Jose\Desktop> python3 3-conexion.py
------------------------------
 CONSULTA 1
------------------------------
('John Smith',)
('Jane Doe',)
------------------------------
 CONSULTA 2
------------------------------
Drama 10
Comedia 5
Acción 3
Terror 2
```