Administración de Bases de Datos

Arenas Deseado Luis Eduardo

Integridad, Confiabilidad y Calidad de Datos en un DBMS

Diego Pacheco Valdez

12 de Junio del 2024

Introducción

En esta tarea se busca mejorar nuestro conocimiento de las restricciones de integridad que se pueden tener dentro de una base de datos.

Desarrollo

- **Planteamiento del problema:**

  Comprender y aplicar restricciones de integridad adicionales en una tabla de base de datos.

```
# actividad del lunes 10 de junio
import psycopg2

conn = psycopg2.connect(
    dbname = "integridad",
    user = "postgres",
    password = "1234",
    host = "localhost"
)

cur = conn.cursor()

# Crear la tabla de Cursos
cur.execute("""
CREATE TABLE IF NOT EXISTS Cursos (
        id SERIAL PRIMARY KEY,
        nombre VARCHAR(50) NOT NULL
)
""")

# Crear la tabla de Estudiantes
cur.execute("""
CREATE TABLE IF NOT EXISTS Estudiantes (
        id SERIAL PRIMARY KEY,
        nombre VARCHAR(50) NOT NULL,
        edad INT CHECK (edad > 0),
        curso_id INT,
        FOREIGN KEY (curso_id) REFERENCES Cursos (id)
)
""")

# Crear una función para validar la edad de los estudiantes

cur.execute("""
```

```
    CREATE OR REPLACE FUNCTION validar_edad() RETURNS TRIGGER
as $$
            BEGIN
                IF NEW.edad <= 0 THEN
                    RAISE EXCEPTION 'La edad debe de ser
mayor a 0';
                END IF;
                RETURN NEW;
END;
$$ LANGUAGE plpgsql
""")

# Eliminar trigger que ya existe
cur.execute("""
    DROP TRIGGER IF EXISTS trigger_validar_edad ON
Estudiantes;
""")

# Crear un trigger que utilice la función validar edad()
cur.execute("""
    CREATE TRIGGER trigger_validar_edad
    BEFORE INSERT OR UPDATE ON Estudiantes
    FOR EACH ROW EXECUTE FUNCTION validar_edad();
""")

# Actualizar cambios
conn.commit()

# actividad TAREA 3
```

- Añadir una nueva columna email a la tabla Estudiantes con la restricción de que debe ser única y no nula.

```
# Ejecutar el comando para añadir la columna email.
# Not Null asegura no pueda estar vacio
# Unique asegura que no se repita
'''cur.execute("""
    ALTER TABLE Estudiantes
    ADD COLUMN email VARCHAR(100) NOT NULL UNIQUE;
""")'''
```

- Crear una función y un trigger para validar que el email tenga un formato válido.

```
# Crear una función validar formato del email
```

```
cur.execute("""
    CREATE OR REPLACE FUNCTION validar_email() RETURNS
TRIGGER AS $$
            BEGIN
                IF NEW.email !~* '^[A-Za-z0-9._%+-]+@[A-Za-
z0-9.-]+\.[A-Z|a-z]{2,}$' THEN
                    RAISE EXCEPTION 'Formato de email no
válido: %', NEW.email;
                END IF;
                RETURN NEW;
END;
$$ LANGUAGE plpgsql;
""")


# Crear un trigger para agregar email una vez validado
cur.execute("""
CREATE OR REPLACE TRIGGER validar_email_trigger
BEFORE INSERT OR UPDATE ON Estudiantes
FOR EACH ROW
EXECUTE FUNCTION validar_email();
""")

# Actualizar cambios
conn.commit()

def mostrar_menu():
    while True:
        print()
        print("~.~.~.~.~.~.~ Base ~.~.~.~.~.~.~")
        print("1. Insertar datos")
        print("2. Leer datos")
        print("3. Actualizar datos")
        print("4. Eliminar datos")
        print("5. Salir")
        print("~.~.~.~.~.~.~.~.~.~.~.~.~.~.~")
        opcion = int(input("Inserte una opción: "))
        if opcion == 1:
            insertar_datos()
        elif opcion == 2:
            leer_datos()
        elif opcion == 3:
            actualizar_datos()
        elif opcion == 4:
            eliminar_datos()
        elif opcion == 5:
```

```
            break
        else:
            print("Favor de insertar una opción válida.")
```

- Modificar el código Python para insertar, leer, actualizar y eliminar datos con las nuevas restricciones.

```
def insertar_datos():
    nombre_curso = input("Ingrese el nombre del curso: ")
    cur.execute("INSERT INTO Cursos (nombre) VALUES (%s) 
RETURNING id", (nombre_curso,))
    curso_id = cur.fetchone()[0]

    nombre_estudiante = input("Ingrese el nombre del 
estudiante: ")
    edad_estudiante = int(input("Ingrese la edad del 
estudiante: "))
    email_estudiante = input("Ingrese el email del 
estudiante: ")

    cur.execute("INSERT INTO Estudiantes (nombre, edad, 
curso_id, email) VALUES (%s, %s, %s, %s)",
                (nombre_estudiante, edad_estudiante, 
curso_id, email_estudiante))

    conn.commit()
    print("Datos insertados correctamente.")

def leer_datos():
    cur.execute("SELECT * FROM Estudiantes")
    estudiantes = cur.fetchall()
    print("\nEstudiantes:")
    for estudiante in estudiantes:
        print(estudiante)

    cur.execute("SELECT * FROM Cursos")
    cursos = cur.fetchall()
    print("\nCursos:")
    for curso in cursos:
        print(curso)

def actualizar_datos():
    nombre_estudiante = input("Ingrese el nombre del 
estudiante: ")
```

```
    nueva_edad = int(input("Ingrese la nueva edad del
estudiante: "))
    nuevo_email = input("Ingrese el nuevo email del
estudiante: ")

    cur.execute("UPDATE Estudiantes SET edad = %s, email = %s
WHERE nombre = %s",
                (nueva_edad, nuevo_email, nombre_estudiante))
    conn.commit()
    print("Datos actualizados!")

def eliminar_datos():
    nombre_estudiante = input("Inserte el nombre del
estudiante: ")
    cur.execute("SELECT curso_id FROM Estudiantes WHERE
nombre = %s", (nombre_estudiante,))
    curso_id = cur.fetchone()[0]
    cur.execute("DELETE FROM Estudiantes WHERE nombre = %s",
(nombre_estudiante,))
    cur.execute("DELETE FROM Cursos WHERE id = %s",
(curso_id,))
    conn.commit()
    print("Datos eliminados exitosamente :)")

mostrar_menu()

# Cerrar conexión
cur.close()
conn.close()
```

Conclusiones

Esta tarea nos permito terminar de entender los conceptos de triggers y funciones así como aplicar las expresiones regulares (regex).