




## Guía: Implementación del método actualizar en PersonaDAO (DAO) con Python y PostgreSQL

### Introducción

En esta lección aprenderás cómo implementar y probar el método `actualizar` dentro de la clase `PersonaDAO`, aplicando el patrón DAO para actualizar registros en la base de datos PostgreSQL. Reutilizaremos el código y buenas prácticas previamente vistas, y realizaremos pruebas para confirmar que la actualización se realiza correctamente.

---

### Paso 1: Ubicación del archivo trabajado

 **Ruta del archivo:** `capa_datos_persona/persona_dao.py`

Este archivo contiene la definición de la clase `PersonaDAO` con los métodos CRUD.

---

### Paso 2: Agregar el método `actualizar`

#### Descripción

Dentro de la clase `PersonaDAO`, se agrega el método `actualizar` como un método de clase. Este método recibe un objeto `Persona`, construye una tupla con sus atributos, y ejecuta la sentencia SQL de actualización.

### Código trabajado (`capa_datos_persona/persona_dao.py`):

```
@classmethod
def actualizar(cls, persona):
    with Conexion.obtener_conexion() as conexion:
        with conexion.cursor() as cursor:
            valores = (persona.nombre, persona.apellido, persona.email, persona.id_persona)
            cursor.execute(cls._ACTUALIZAR, valores)
            log.debug(f'Persona actualizada: {persona}')
            return cursor.rowcount
```

### Explicación:

- Se abre una conexión y un cursor usando `with` para manejar automáticamente los recursos.
- Se arma una tupla `valores` con el nombre, apellido, email y `id_persona`.
- Se ejecuta la sentencia SQL de actualización con esos valores.
- Se registra en el log el objeto actualizado.
- Finalmente, se devuelve el número de filas modificadas con `cursor.rowcount`.

## Paso 3: Probar el método `actualizar`

### Descripción

Agregamos una prueba al final del archivo para actualizar un registro ya existente en la base de datos.

### Código trabajado (`capa_datos_persona/persona_dao.py`):

```
if __name__ == '__main__':
    # Actualizar un registro
    persona1 = Persona(1, 'Juan Carlos', 'Juarez', 'cjjuarez@mail.com')
    personas_actualizadas = PersonaDAO.actualizar(persona1)
    log.debug(f'Personas actualizadas: {personas_actualizadas}')


    # Seleccionar objetos
    personas = PersonaDAO.seleccionar()
    for persona in personas:
        log.debug(persona)
```


### Explicación:

- Creamos un objeto `Persona` con ID 1 y nuevos valores para nombre, apellido y email.
- Llamamos a `PersonaDAO.actualizar()` para modificar el registro en la base de datos.
- Registramos en el log cuántas personas fueron actualizadas.
- Finalmente, usamos `seleccionar()` para mostrar todos los registros y verificar que el cambio se refleje correctamente.



## Sección final: Código completo de los archivos trabajados

 Aquí puedes agregar el código completo para el alumno:

 **Ruta y nombre del archivo:** `capa_datos_persona/persona_dao.py`

```
from conexion import Conexion

from persona import Persona
from logger_base import log

class PersonaDAO:
    """
    DAO (Data Access Object)
    CRUD (Create-Read-Update-Delete)
    """
    _SELECCIONAR = 'SELECT * FROM persona ORDER BY id_persona'
    _INSERTAR = 'INSERT INTO persona(nombre, apellido, email) VALUES(%s, %s, %s)'
    _ACTUALIZAR = 'UPDATE persona SET nombre=%s, apellido=%s, email=%s WHERE id_persona=%s'
    _ELIMINAR = 'DELETE FROM persona WHERE id_persona=%s'

    @classmethod
    def seleccionar(cls):
        with Conexion.obtener_conexion() as conexion:
            with conexion.cursor() as cursor:
                cursor.execute(cls._SELECCIONAR)
                registros = cursor.fetchall()
                personas = []
                for registro in registros:
                    persona = Persona(registro[0], registro[1], registro[2], registro[3])
                    personas.append(persona)
                return personas

    @classmethod
    def insertar(cls, persona):
        with Conexion.obtener_conexion() as conexion:
            with conexion.cursor() as cursor:
                valores = (persona.nombre, persona.apellido, persona.email)
                cursor.execute(cls._INSERTAR, valores)
```

```
        log.debug(f'Persona insertada: {persona}')
        return cursor.rowcount

    @classmethod
    def actualizar(cls, persona):
        with Conexion.obtener_conexion() as conexion:
            with conexion.cursor() as cursor:
                valores = (persona.nombre, persona.apellido, persona.email, persona.id_persona)
                cursor.execute(cls._ACTUALIZAR, valores)
                log.debug(f'Persona actualizada: {persona}')
                return cursor.rowcount

if __name__ == '__main__':
    # Insertar un registro
    # persona1 = Persona(nombre='Pedro', apellido='Najera', email='pnajera@mail.com')
    # personas_insertadas = PersonaDAO.insertar(persona1)
    # log.debug(f'Personas insertadas: {personas_insertadas}')

    # Actualizar un registro
    persona1 = Persona(1, 'Juan Carlos', 'Juarez', 'cjjuarez@mail.com')
    personas_actualizadas = PersonaDAO.actualizar(persona1)
    log.debug(f'Personas actualizadas: {personas_actualizadas}')

    # Seleccionar objetos
    personas = PersonaDAO.seleccionar()
    for persona in personas:
        log.debug(persona)
```

---

## ✓ Conclusión

👏 En esta guía implementamos el método `actualizar` para modificar registros en la base de datos PostgreSQL usando Python y el patrón DAO. Verificamos su funcionamiento a través de una prueba real, observando cómo se modifica correctamente el registro especificado.

🚀 Esta funcionalidad permite mantener actualizados los datos en nuestra aplicación de forma profesional, preparando el camino para una arquitectura robusta y bien estructurada.

---

🌟 Sigue adelante con tu aprendizaje 🚀, ¡el esfuerzo vale la pena!

¡Saludos! 🙌

**Ing. Marcela Gamiño e Ing. Ubaldo Acosta**

**Fundadores de [GlobalMentoring.com.mx](https://www.globalmentoring.com.mx)**