✏️ Flask CRUD: UPDATE y DELETE

Actualizar y Eliminar Datos desde la Base de Datos

Python Full Stack - Clase 26

¿Qué aprenderemos hoy?

- ✏️ Actualizar registros existentes con UPDATE y formularios de edición
- 🔍 Obtener un registro específico por ID para editarlo
- 📝 Crear formularios pre-llenados con datos existentes
- 🗑️ Eliminar registros de forma segura con DELETE
- 🔗 Agregar enlaces de acción (Ver, Editar, Borrar) en tablas
- 🎯 Completar todas las operaciones CRUD (Create, Read, Update, Delete)
- 🔄 Implementar rutas GET y POST para actualización
- ✅ Validar que los registros existen antes de modificar o eliminar

🔄 Recordando: ¿Qué es CRUD?

CRUD son las 4 operaciones básicas de cualquier aplicación:

Operación	Método HTTP	SQL	Propósito
Create	POST	INSERT	Crear nuevos registros
Read	GET	SELECT	Leer/consultar datos
Update	POST (con datos)	UPDATE	Modificar registros existentes
Delete	POST (o GET)	DELETE	Eliminar registros

En la clase anterior aprendimos Create y Read. Hoy completamos con Update y Delete. ✅

✏️ UPDATE: Actualizar Datos

El Flujo de Actualización

Para actualizar un registro necesitamos **2 pasos**:

```
1. Mostrar formulario con datos actuales (GET)
        ↓
2. Procesar la actualización (POST)
        ↓
3. Redirigir a la página principal
```

Es similar a crear, pero primero necesitamos obtener los datos existentes.

Paso 1: Método get_one() en el Modelo

```python
# Primero, necesitamos un método para obtener **un solo registro** por su ID:
class Usuario:
    # ... métodos anteriores (get_all, save) ...
    @classmethod
    def get_one(cls, usuario_id):
        """
        Obtiene un usuario específico por su ID.
        Retorna un objeto Usuario o None si no existe.
        """
        query = "SELECT * FROM usuarios WHERE id = %(id)s;"

        datos = {
            'id': usuario_id
        }

        resultado = connectToMySQL('tienda').query_db(query, datos)

        # Si encontramos el usuario, retornamos un objeto
        if resultado:
            return cls(resultado[0])
        return None  # Si no existe, retornamos None
```

```python
@classmethod
def update(cls, datos):
    """
    Actualiza un usuario existente en la base de datos.
    El diccionario 'datos' debe incluir el 'id' del usuario.
    """
    query = """
        UPDATE usuarios
        SET nombre = %(nombre)s,
            apellido = %(apellido)s,
            email = %(email)s,
            edad = %(edad)s,
            updated_at = NOW()
        WHERE id = %(id)s;
    """

    # UPDATE retorna el número de filas afectadas
    return connectToMySQL('tienda').query_db(query, datos)
```

Nota:

- `UPDATE` retorna el número de filas modificadas (normalmente 1)
- **SIEMPRE** incluye `WHERE id = %(id)s` para actualizar solo un registro

Paso 3: Ruta GET para Mostrar Formulario de Edición

En `server.py`, creamos la ruta que muestra el formulario con datos actuales

```
@app.route('/usuarios/editar/<int:usuario_id>')
def editar_usuario(usuario_id):
    """
    Muestra el formulario de edición con los datos del usuario.
    """
    # Obtenemos el usuario por su ID
    usuario = Usuario.get_one(usuario_id)

    # Si el usuario no existe, redirigimos
    if not usuario:
        return redirect('/')

    # Mostramos el formulario con los datos del usuario
    return render_template('editar_usuario.html', usuario=usuario)
```

Importante:

- `<int:usuario_id>` captura el ID de la URL
- Verificamos que el usuario existe antes de mostrar el formulario

Paso 4: Plantilla del Formulario de Edición

```
<!DOCTYPE html>
<html>
<head>
    <title>Editar Usuario</title>
</head>
<body>
    <h1>Editar Usuario</h1>

    <form action="/usuarios/actualizar" method="POST">
        <!-- Campo oculto con el ID del usuario -->
        <input type="hidden" name="id" value="{{ usuario.id }}">

        <label>Nombre:</label>
        <input type="text" name="nombre" value="{{ usuario.nombre }}"><br>

        <label>Apellido:</label>
        <input type="text" name="apellido" value="{{ usuario.apellido }}"><br>

        <label>Email:</label>
        <input type="email" name="email" value="{{ usuario.email }}"><br>

        <label>Edad:</label>
        <input type="number" name="edad" value="{{ usuario.edad }}"><br>

        <input type="submit" value="Actualizar Usuario">
    </form>

    <a href="/">Volver</a>
</body>
</html>
```

- El formulario debe estar **pre-llenado** con los datos actuales:
- **Clave**: `value="{{ usuario.nombre }}"` pre-llena el campo con el valor actual.

Desglosando el Formulario de Edición

Componentes importantes:

1. **Campo oculto con ID**:

```
<input type="hidden" name="id" value="{{ usuario.id }}">
```

   - Necesitamos el ID para saber qué usuario actualizar
   - `hidden` significa que no se ve pero se envía en el formulario

2. **Campos pre-llenados**:

```
<input type="text" name="nombre" value="{{ usuario.nombre }}">
```

   - `value` muestra el valor actual del usuario
   - El usuario puede modificar estos valores

3. **Action diferente**:

```
<form action="/usuarios/actualizar" method="POST">
```

   - Va a una ruta diferente que la de creación

Paso 5: Ruta POST para Procesar la Actualización

```python
@app.route('/usuarios/actualizar', methods=['POST'])
def actualizar_usuario():
    """
    Procesa la actualización del usuario.
    Recibe los datos del formulario y actualiza en la BD.
    """
    # Creamos diccionario con los datos del formulario
    datos = {
        "id": int(request.form['id']),  # ID del usuario a actualizar
        "nombre": request.form['nombre'],
        "apellido": request.form['apellido'],
        "email": request.form['email'],
        "edad": int(request.form['edad']) if request.form.get('edad') else None
    }

    # Actualizamos en la base de datos
    Usuario.update(datos)

    # Redirigimos a la página principal
    return redirect('/')
```

Importante:

- El ID viene del campo oculto del formulario
- Redirigimos después de actualizar (patrón POST-Redirect-GET)

```
    resultado = connectToMySQL('tienda').query_db(query, datos)
    if resultado:
        return cls(resultado[0])
    return None

@classmethod
def update(cls, datos):
    query = """
        UPDATE usuarios
        SET nombre = %(nombre)s, apellido = %(apellido)s,
            email = %(email)s, edad = %(edad)s, updated_at = NOW()
        WHERE id = %(id)s;
    """
    return connectToMySQL('tienda').query_db(query, datos)
```

Controlador (server.py):

```
@app.route('/usuarios/editar/<int:usuario_id>')
def editar_usuario(usuario_id):
    usuario = Usuario.get_one(usuario_id)
    if not usuario:
        return redirect('/')
    return render_template('editar_usuario.html', usuario=usuario)
```

💪 Ejercicio: Implementar UPDATE

¡Hora de practicar!

Extiende tu aplicación de usuarios con funcionalidad de actualización:

1. Agrega método `get_one()` al modelo Usuario
2. Agrega método `update()` al modelo Usuario
3. Crea ruta GET `/usuarios/editar/<int:usuario_id>` que muestre el formulario
4. Crea plantilla `editar_usuario.html` con formulario pre-llenado
5. Crea ruta POST `/usuarios/actualizar` que procese la actualización

Pistas:

- Usa sentencias preparadas en ambos métodos
- El formulario debe tener un campo oculto con el ID
- Pre-llena los campos con `value="{{ usuario.campo }}"`
- Verifica que el usuario existe antes de mostrar el formulario
- Redirige después de actualizar

⏱️ **Tiempo:** 20 minutos

🗑️ DELETE: Eliminar Datos

El Flujo de Eliminación

Para eliminar un registro necesitamos:

```
1. Confirmar que queremos eliminar (opcional pero recomendado)
        ↓
2. Procesar la eliminación (POST)
        ↓
3. Redirigir a la página principal
```

⚠️ **IMPORTANTE**: La eliminación es **permanente**. Siempre verifica antes de eliminar.

Paso 1: Método delete() en el Modelo

Creamos el método para eliminar:

```python
@classmethod
def delete(cls, usuario_id):
    """
    Elimina un usuario de la base de datos por su ID.
    Retorna el número de filas eliminadas (normalmente 1).
    """
    query = "DELETE FROM usuarios WHERE id = %(id)s;"

    datos = {
        'id': usuario_id
    }

    # DELETE retorna el número de filas eliminadas
    return connectToMySQL('tienda').query_db(query, datos)
```

⚠️ **CRÍTICO**:

- **SIEMPRE** usa `WHERE id = %(id)s`
- Sin WHERE, eliminarías TODOS los registros
- La eliminación es permanente e irreversible

Paso 2: Ruta POST para Eliminar

En `server.py`, creamos la ruta que procesa la eliminación:

```
@app.route('/usuarios/eliminar/<int:usuario_id>', methods=['POST'])
def eliminar_usuario(usuario_id):
    """
    Elimina un usuario de la base de datos.
    """
    # Verificamos que el usuario existe antes de eliminar
    usuario = Usuario.get_one(usuario_id)

    if usuario:
        # Eliminamos el usuario
        Usuario.delete(usuario_id)

    # Redirigimos a la página principal
    return redirect('/')
```

Nota:

- Usamos `methods=['POST']` para evitar eliminaciones accidentales
- Verificamos que el usuario existe antes de eliminar
- Redirigimos después de eliminar

```
<table>
    <thead>
        <tr>
            <th>ID</th>
            <th>Nombre</th>
            <th>Apellido</th>
            <th>Email</th>
            <th>Acciones</th>
        </tr>
    </thead>
    <tbody>
        {% for usuario in usuarios %}
        <tr>
            <td>{{ usuario.id }}</td>
            <td>{{ usuario.nombre }}</td>
            <td>{{ usuario.apellido }}</td>
            <td>{{ usuario.email }}</td>
            <td>
                <!-- Formulario para eliminar -->
                <form action="/usuarios/eliminar/{{ usuario.id }}" method="POST" style="display: inline;">
                    <input type="submit" value="Eliminar" onclick="return confirm('¿Estás seguro?');">
                </form>
            </td>
        </tr>
        {% endfor %}
    </tbody>
</table>
```

Características:

- Formulario con `method="POST"`

Confirmación de Eliminación con JavaScript

El `confirm()` muestra un diálogo de confirmación:

```
onclick="return confirm('¿Estás seguro de eliminar este usuario?');"
```

¿Cómo funciona?

- Si el usuario hace clic en "Aceptar" → retorna `true` → se envía el formulario
- Si el usuario hace clic en "Cancelar" → retorna `false` → NO se envía el formulario

Esto previene eliminaciones accidentales. ✅

Alternativa: Ruta GET para Eliminar (Menos Segura)

También puedes usar GET, pero es **menos seguro**:

```
@app.route('/usuarios/eliminar/<int:usuario_id>')
def eliminar_usuario(usuario_id):
    Usuario.delete(usuario_id)
    return redirect('/')
```

Problemas con GET:

- ❌ Los bots pueden seguir enlaces y eliminar registros
- ❌ Se puede eliminar accidentalmente al seguir un enlace
- ❌ No hay confirmación fácil

Recomendación: Usa POST para operaciones destructivas como DELETE.

El Flujo Completo de Eliminación

Modelo (usuario.py):

```
@classmethod
def delete(cls, usuario_id):
    query = "DELETE FROM usuarios WHERE id = %(id)s;"
    datos = {'id': usuario_id}
    return connectToMySQL('tienda').query_db(query, datos)
```

Controlador (server.py):

```
@app.route('/usuarios/eliminar/<int:usuario_id>', methods=['POST'])
def eliminar_usuario(usuario_id):
    usuario = Usuario.get_one(usuario_id)
    if usuario:
        Usuario.delete(usuario_id)
    return redirect('/')
```

Vista (index.html):

```
<form action="/usuarios/eliminar/{{ usuario.id }}" method="POST" style="display: inline;">
    <input type="submit" value="Eliminar" onclick="return confirm('¿Estás seguro?');">
</form>
```

🧩 Ejercicio: Implementar DELETE

¡Hora de practicar!

Agrega funcionalidad de eliminación a tu aplicación:

1. Agrega método `delete()` al modelo Usuario
2. Crea ruta POST `/usuarios/eliminar/<int:usuario_id>` que procese la eliminación
3. Agrega columna "Acciones" en la tabla de usuarios
4. Agrega botón "Eliminar" con confirmación JavaScript

Pistas:

- Usa sentencias preparadas en el método delete()
- Verifica que el usuario existe antes de eliminar
- Agrega confirmación con `confirm()` antes de eliminar
- Usa POST para la ruta de eliminación
- Redirige después de eliminar

🎯 CRUD Completo: Sistema de Usuarios

Ahora que tenemos todas las operaciones, podemos crear un sistema completo:

Operaciones disponibles:

- ✅ **Create**: Crear nuevos usuarios
- ✅ **Read**: Ver todos los usuarios y ver detalles de uno
- ✅ **Update**: Editar usuarios existentes
- ✅ **Delete**: Eliminar usuarios

Rutas necesarias:

- `GET /` → Mostrar todos los usuarios
- `GET /usuarios/nuevo` → Formulario para crear
- `POST /usuarios/crear` → Procesar creación
- `GET /usuarios/<id>` → Ver detalles de un usuario
- `GET /usuarios/editar/<id>` → Formulario de edición
- `POST /usuarios/actualizar` → Procesar actualización

Tabla con Acciones Completas

```
<table>
    <thead>
        <tr>
            <th>ID</th>
            <th>Nombre</th>
            <th>Apellido</th>
            <th>Email</th>
            <th>Acciones</th>
        </tr>
    </thead>
    <tbody>
        {% for usuario in usuarios %}
        <tr>
            <td>{{ usuario.id }}</td>
            <td>{{ usuario.nombre }}</td>
            <td>{{ usuario.apellido }}</td>
            <td>{{ usuario.email }}</td>
            <td> <!-- "Ver" y "Editar" son enlaces GET, "Eliminar" es un formulario POST. -->
                <a href="/usuarios/{{ usuario.id }}">Ver</a> |
                <a href="/usuarios/editar/{{ usuario.id }}">Editar</a> |
                <form action="/usuarios/eliminar/{{ usuario.id }}" method="POST" style="display: inline;">
                    <input type="submit" value="Eliminar" onclick="return confirm('¿Eliminar?');">
                </form>
            </td>
        </tr>
        {% endfor %}
    </tbody>
</table>
```

Ruta para Ver Detalles de un Usuario

También podemos agregar una página para ver los detalles completos:

```
@app.route('/usuarios/<int:usuario_id>')
def mostrar_usuario(usuario_id):
    """
    Muestra los detalles completos de un usuario.
    """
    usuario = Usuario.get_one(usuario_id)

    if not usuario:
        return redirect('/')

    return render_template('mostrar_usuario.html', usuario=usuario)
```

Plantilla mostrar_usuario.html:

```
<h1>Detalles del Usuario</h1>
<p><strong>ID:</strong> {{ usuario.id }}</p>
<p><strong>Nombre:</strong> {{ usuario.nombre }}</p>
<p><strong>Apellido:</strong> {{ usuario.apellido }}</p>
<p><strong>Email:</strong> {{ usuario.email }}</p>
<p><strong>Edad:</strong> {{ usuario.edad }}</p>
<p><strong>Creado:</strong> {{ usuario.created_at }}</p>
<p><strong>Actualizado:</strong> {{ usuario.updated_at }}</p>

<a href="/usuarios/editar/{{ usuario.id }}">Editar</a> |
<a href="/">Volver</a>
```

Modelo Usuario Completo

Aquí está el modelo completo con todas las operaciones CRUD:

```python
class Usuario:
    def __init__(self, data):
        self.id = data['id']
        self.nombre = data['nombre']
        self.apellido = data['apellido']
        self.email = data['email']
        self.edad = data.get('edad')
        self.created_at = data['created_at']
        self.updated_at = data['updated_at']

    @classmethod
    def get_all(cls):
        query = "SELECT * FROM usuarios ORDER BY id DESC;"
        resultados = connectToMySQL('tienda').query_db(query)
        usuarios = []
        for usuario in resultados:
            usuarios.append(cls(usuario))
        return usuarios

    @classmethod
    def get_one(cls, usuario_id):
        query = "SELECT * FROM usuarios WHERE id = %(id)s;"
        datos = {'id': usuario_id}
        resultado = connectToMySQL('tienda').query_db(query, datos)
        if resultado:
            return cls(resultado[0])
        return None

    @classmethod
    def save(cls, datos):
        query = """
            INSERT INTO usuarios (nombre, apellido, email, edad, created_at, updated_at)
            VALUES (%(nombre)s, %(apellido)s, %(email)s, %(edad)s, NOW(), NOW());
        """
        return connectToMySQL('tienda').query_db(query, datos)

    @classmethod
    def update(cls, datos):
        query = """
            UPDATE usuarios
            SET nombre = %(nombre)s, apellido = %(apellido)s,
                email = %(email)s, edad = %(edad)s, updated_at = NOW()
            WHERE id = %(id)s;
        """
        return connectToMySQL('tienda').query_db(query, datos)

    @classmethod
    def delete(cls, usuario_id):
        query = "DELETE FROM usuarios WHERE id = %(id)s;"
        datos = {'id': usuario_id}
        return connectToMySQL('tienda').query_db(query, datos)
```

💪 Ejercicio: CRUD Usuarios Completo

¡Hora de practicar!

Crea una aplicación CRUD completa para gestionar usuarios:

Requisitos:

1. **Modelo Usuario** con métodos:
   - `get_all()` - Obtener todos
   - `get_one(id)` - Obtener uno por ID
   - `save(datos)` - Crear nuevo
   - `update(datos)` - Actualizar existente
   - `delete(id)` - Eliminar

2. **Rutas GET:**
    - `/` - Mostrar todos los usuarios en tabla
    - `/usuarios/nuevo` - Formulario para crear
    - `/usuarios/<id>` - Ver detalles de un usuario
    - `/usuarios/editar/<id>` - Formulario de edición
3. **Rutas POST:**
    - `/usuarios/crear` - Procesar creación
    - `/usuarios/actualizar` - Procesar actualización
    - `/usuarios/eliminar/<id>` - Procesar eliminación

4. **Tabla con acciones:**
    - Enlaces para Ver, Editar
    - Botón Eliminar con confirmación

Pistas:

- Usa sentencias preparadas en todas las consultas
- Verifica que los usuarios existen antes de editar/eliminar
- Redirige después de cada operación POST
- Pre-llena el formulario de edición con datos actuales

⏱️ **Tiempo:** 45 minutos

Resumen de Conceptos Clave

1. **get_one()**: Método que obtiene un registro por ID, retorna objeto o None
2. **update()**: Método que actualiza registros usando UPDATE con WHERE
3. **delete()**: Método que elimina registros usando DELETE con WHERE
4. **Formularios pre-llenados**: Usan `value="{{ objeto.campo }}"` para mostrar datos actuales
5. **Campo oculto**: `<input type="hidden">` para enviar el ID en formularios de edición
6. **Confirmación JavaScript**: `confirm()` previene eliminaciones accidentales
7. **POST para DELETE**: Usa POST en lugar de GET para operaciones destructivas
8. **CRUD completo**: Create, Read, Update, Delete - las 4 operaciones básicas

Lo Más Importante

✅ **SIEMPRE usa WHERE en UPDATE y DELETE** - Sin WHERE puedes modificar/eliminar todos los registros
✅ **Verifica que el registro existe** antes de editar o eliminar
✅ **Usa campos ocultos** para enviar el ID en formularios de edición
✅ **Pre-llena formularios** con `value="{{ objeto.campo }}"`
✅ **Confirma antes de eliminar** usando JavaScript `confirm()`
✅ **Usa POST para DELETE** - Es más seguro que GET
✅ **Redirige después de POST** - Sigue el patrón POST-Redirect-GET
✅ **Sentencias preparadas siempre** - Para seguridad y prevenir inyección SQL

Recursos para Seguir Aprendiendo

- **Documentación Flask**: https://flask.palletsprojects.com/
- **PyMySQL Docs**: https://pymysql.readthedocs.io/
- **SQL UPDATE**: https://dev.mysql.com/doc/refman/8.0/en/update.html
- **SQL DELETE**: https://dev.mysql.com/doc/refman/8.0/en/delete.html
- **Flask URL Building**: https://flask.palletsprojects.com/en/2.3.x/quickstart/#url-building

Próximos Pasos

En la siguiente clase aprenderemos:

- 🔗 **JOINs** para obtener datos relacionados entre tablas
- 🔐 **Validaciones** de datos antes de guardar
- 📊 **Relaciones** entre tablas (uno a muchos, muchos a muchos)
- 🎨 **Mejoras de UI** con CSS y JavaScript
- 🔒 **Autenticación** y sesiones de usuario

Práctica para Casa

Proyecto sugerido:

Crea una aplicación de "Gestión de Libros":

- Ver todos los libros
- Crear nuevos libros (título, autor, año, género)
- Editar libros existentes
- Eliminar libros
- Ver detalles de un libro

Desafío extra:

- Agrega búsqueda por título o autor
- Agrega filtros por género
- Agrega validaciones (año entre 1000-2024, título no vacío)

Consejos Finales

💡 **Depuración**: Usa `print()` para ver qué contiene `request.form` y los datos antes de actualizar/eliminar

💡 **Errores SQL**: Copia el query de la terminal y pruébalo en MySQL Workbench

💡 **Prueba paso a paso**: Implementa una operación a la vez (primero UPDATE, luego DELETE)

💡 **Backup mental**: Antes de eliminar, verifica que tienes el registro correcto

💡 **Nombres consistentes**: Usa los mismos nombres en formularios, diccionarios y queries

🎉 ¡Felicidades!

Ya sabes completar operaciones CRUD en Flask

Has aprendido todas las operaciones: Create, Read, Update y Delete

? Preguntas

¿Alguna duda sobre UPDATE, DELETE o CRUD completo?

🙌 ¡Excelente Trabajo!