🚀 Flask CRUD: GET y POST

Mostrar y Crear Datos desde la Base de Datos

Python Full Stack - Clase 25

¿Qué aprenderemos hoy?

- 🔍 Usar sentencias preparadas para consultas con variables
- 🛡️ Entender qué es la inyección SQL y cómo prevenirla
- 📝 Crear formularios HTML que envíen datos a Flask
- 💾 Guardar datos de formularios en la base de datos con INSERT
- 🔄 Redirigir después de crear registros
- 🎯 Implementar operaciones CRUD básicas (Create y Read)

🔍 Consultas con Datos Variables

El Problema: Consultas Estáticas

Hasta ahora hemos usado consultas **estáticas** (fijas):

```
query = "SELECT * FROM productos;"
resultados = connectToMySQL('tienda').query_db(query)
```

¿Qué pasa si queremos buscar un producto específico? 🤔

Necesitamos Consultas Dinámicas

Muchas veces necesitamos consultas que **cambian** según lo que el usuario pide:

- `SELECT * FROM productos WHERE id = 1;` (el ID cambia)
- `SELECT * FROM productos WHERE nombre = "Laptop HP";` (el nombre cambia)
- `UPDATE productos SET precio = 799.99 WHERE id = 3;` (ambos valores cambian)

Necesitamos usar variables en nuestras consultas SQL.

Sentencias Preparadas: La Solución

Las **sentencias preparadas** nos permiten usar variables de forma segura:

```
# 1. Definimos el query con placeholders
query = "SELECT * FROM productos WHERE id = %(id_producto)s;"

# 2. Creamos un diccionario con los valores
datos = {
    'id_producto': 1
}

# 3. Ejecutamos pasando ambos
resultado = connectToMySQL('tienda').query_db(query, datos)
```

¿Por qué `%(id_producto)s`? Veremos la razón en un momento.

Desglosando las Sentencias Preparadas

Componentes importantes:

1. `query` : Cadena de texto con la consulta SQL
    - Usa `%(clave)s` como placeholder (con la `s` al final)
    - La `clave` debe coincidir con la clave del diccionario
2. `datos` : Diccionario con los valores
    - Las claves deben coincidir con los placeholders del query
    - Los valores pueden ser strings, números, fechas, etc.
3. `query_db(query, datos)` : Ejecuta la consulta de forma segura

Ejemplo Completo: Buscar por ID

```python
from flask_app.config.mysqlconnection import connectToMySQL

class Producto:
    @classmethod
    def get_one(cls, id_producto):
        # Query con placeholder
        query = "SELECT * FROM productos WHERE id = %(id_producto)s;"

        # Diccionario con el valor
        datos = {
            'id_producto': id_producto
        }

        # Ejecutamos la consulta
        resultado = connectToMySQL('tienda').query_db(query, datos)

        # Retornamos el primer resultado (si existe)
        if resultado:
            return cls(resultado[0])
        return None
```

Ejemplo: Buscar por Nombre

```python
@classmethod
def get_by_nombre(cls, nombre_buscado):
    query = "SELECT * FROM productos WHERE nombre = %(nombre)s;"

    datos = {
        'nombre': nombre_buscado
    }

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

    productos = []
    for producto in resultados:
        productos.append(cls(producto))

    return productos
```

Nota: Este método puede retornar múltiples resultados si hay varios productos con el mismo nombre.

Múltiples Condiciones

También podemos usar múltiples variables en una consulta:

```
@classmethod
def get_by_categoria_y_precio(cls, categoria_id, precio_maximo):
    query = """
        SELECT * FROM productos
        WHERE categoria_id = %(categoria_id)s
        AND precio <= %(precio_maximo)s;
    """

    datos = {
        'categoria_id': categoria_id,
        'precio_maximo': precio_maximo
    }

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

    productos = []
    for producto in resultados:
        productos.append(cls(producto))

    return productos
```

🤏 Ejercicio: Consultas con variables

¡Hora de practicar!

Crea un archivo `ejercicios/01-consultas-variables.py` con:

1. Crea un método `get_by_id()` que busque un producto por su ID
2. Crea un método `get_by_categoria()` que busque todos los productos de una categoría específica
3. Crea un método `get_by_precio_maximo()` que busque productos con precio menor o igual a un valor

Pistas:

- Usa sentencias preparadas con `%(clave)s`
- Recuerda que `get_by_id()` debe retornar un solo objeto o None
- Los métodos que pueden retornar múltiples resultados deben retornar una lista
- La base de datos se llama `tienda`

Seguridad: Inyección SQL

¿Por Qué No Concatenar Strings?

Podrías pensar: "¿Por qué no simplemente concatenar el valor?"

```
# ❌ NUNCA HAGAS ESTO
nombre = request.form['nombre']
query = f"SELECT * FROM productos WHERE nombre = '{nombre}';"
resultado = mysql.query_db(query)
```

Esto parece más simple, pero es MUY PELIGROSO. ⚠️

El Problema: Inyección SQL

Un usuario malintencionado podría ingresar en el formulario:

```
Laptop' OR '1'='1
```

Con concatenación, la consulta se convertiría en:

```
SELECT * FROM productos WHERE nombre = 'Laptop' OR '1'='1';
```

Como `'1'='1'` siempre es verdadero, esto devolvería TODOS los productos. 😱

Ejemplo Más Peligroso

Un atacante podría ingresar:

```
Laptop'; DROP TABLE productos; --
```

La consulta se convertiría en:

```
SELECT * FROM productos WHERE nombre = 'Laptop'; DROP TABLE productos; --';
```

¡Esto eliminaría toda la tabla de productos! 💥

Cómo Funcionan las Sentencias Preparadas

Con sentencias preparadas, el mismo ataque se vuelve inofensivo:

```
query = "SELECT * FROM productos WHERE nombre = %(nombre)s;"
datos = {'nombre': "Laptop' OR '1'='1"}
resultado = mysql.query_db(query, datos)
```

MySQL interpreta esto como:

```
SELECT * FROM productos WHERE nombre = 'Laptop'' OR ''1''=''1';
```

Ahora está buscando un producto con ese nombre exacto (que no existe), no ejecutando código SQL malicioso. ✅

¿Por Qué Son Seguras?

Las sentencias preparadas son seguras porque:

1. **Separación de código y datos**: El SQL y los datos se envían por separado
2. **Escape automático**: MySQL escapa automáticamente caracteres especiales
3. **Una sola consulta**: PyMySQL solo ejecuta una consulta a la vez
4. **Validación**: Los datos se tratan como valores, no como código

Siempre usa sentencias preparadas cuando tengas variables en tus consultas. ✅

Regla de Oro

```
🔒 SIEMPRE usa sentencias preparadas cuando:
   - Tengas variables en tu consulta SQL
   - Los datos vengan de un formulario
   - Los datos vengan de la URL
   - Los datos vengan de cualquier fuente externa
```

Nunca concatenes strings para crear consultas SQL. ❌

📝 Del Formulario a la Base de Datos

El Flujo Completo: Crear un Registro

```
1. Usuario llena formulario HTML
        ↓
2. Formulario se envía (POST) a Flask
        ↓
3. Flask recibe datos en request.form
        ↓
4. Creamos diccionario con los datos
        ↓
5. Llamamos método save() del modelo
        ↓
6. Método ejecuta INSERT en la BD
        ↓
7. Redirigimos a página de visualización
```

Paso 1: Crear el Método save() en el Modelo

Primero, agreguemos un método para guardar nuevos registros:

```
class Producto:
    # ... métodos anteriores (get_all, get_one, etc.) ...

    @classmethod
    def save(cls, datos):
        query = """
            INSERT INTO productos
            (nombre, precio, descripcion, categoria_id, created_at, updated_at)
            VALUES (%(nombre)s, %(precio)s, %(descripcion)s, %(categoria_id)s, NOW(), NOW());
        """

        # INSERT retorna el ID del nuevo registro
        nuevo_id = connectToMySQL('tienda').query_db(query, datos)
        return nuevo_id
```

Nota: `INSERT` retorna el ID del nuevo registro creado.

En nuestra plantilla, agregamos un formulario:

```
<h2>Agregar un producto</h2>

<form action="/productos/crear" method="POST">
    <label>Nombre:</label>
    <input type="text" name="nombre"><br>

    <label>Precio:</label>
    <input type="number" step="0.01" name="precio"><br>

    <label>Descripción:</label>
    <textarea name="descripcion"></textarea><br>

    <label>Categoría ID:</label>
    <input type="number" name="categoria_id"><br>

    <input type="submit" value="Agregar Producto">
</form>
```

Importante:

- `action="/productos/crear"` → Ruta donde se enviará el formulario

En `server.py`, creamos la ruta que recibe el formulario:

```
from flask import Flask, render_template, request, redirect

@app.route("/productos/crear", methods=['POST'])
def crear_producto():
    # Creamos diccionario desde request.form
    datos = {
        "nombre": request.form['nombre'],
        "precio": float(request.form['precio']),
        "descripcion": request.form['descripcion'],
        "categoria_id": int(request.form['categoria_id'])
    }

    # Guardamos en la base de datos
    Producto.save(datos)

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

Importante:

- `methods=['POST']` → Solo acepta peticiones POST

Desglosando request.form

Cuando el usuario envía el formulario:

```
<input type="text" name="nombre" value="Laptop HP">
<input type="number" name="precio" value="899.99">
```

Flask recibe esto como un diccionario:

```
request.form = {
    'nombre': 'Laptop HP',
    'precio': '899.99',
    'descripcion': 'Laptop HP 15 pulgadas',
    'categoria_id': '1'
}
```

Las claves del diccionario coinciden con el atributo `name` de los inputs.

Modelo (producto.py):

```
@classmethod
def save(cls, datos):
    query = "INSERT INTO productos (nombre, precio, descripcion, categoria_id, created_at, updated_at) VALUES (%(nombre)s, %(precio)s, %(descripcion)s, %(categoria_id)s, NOW(), NOW());"
    return connectToMySQL('tienda').query_db(query, datos)
```

Controlador (server.py):

```
@app.route("/productos/crear", methods=['POST'])
def crear_producto():
    datos = {
        "nombre": request.form['nombre'],
        "precio": float(request.form['precio']),
        "descripcion": request.form['descripcion'],
        "categoria_id": int(request.form['categoria_id'])
    }
    Producto.save(datos)
    return redirect('/')
```

Vista (index.html):

```
<form action="/productos/crear" method="POST">
    <input type="text" name="nombre">
```

¿Por Qué Redirigir Después de POST?

Patrón POST-Redirect-GET:

1. Usuario envía formulario (POST)
2. Guardamos datos en la BD
3. Redirigimos a una página GET (visualización)

Ventajas:

- ✅ Evita reenvío accidental del formulario
- ✅ Muestra el nuevo registro inmediatamente
- ✅ Mejor experiencia de usuario
- ✅ Buena práctica de desarrollo web

Siempre redirige después de un POST. ✅

💪 Ejercicio: Formulario de Creación

¡Hora de practicar!

Crea una aplicación Flask completa:

1. Crea un modelo `Producto` con método `save()`
2. Crea una ruta GET `/` que muestre todos los productos
3. Crea un formulario HTML para agregar productos
4. Crea una ruta POST `/productos/crear` que guarde y redirija

Pistas:

- Usa sentencias preparadas en el método save()
- Recuerda importar `redirect` de Flask
- El formulario debe tener `method="POST"`
- La base de datos se llama `tienda`
- Los campos del producto son: nombre, precio, descripcion, categoria_id

⏱️ **Tiempo:** 20 minutos

Manejo de Errores: request.form

¿Qué pasa si falta un campo?

Si intentas acceder a un campo que no existe:

```
nombre = request.form['nombre']  # ❌ Error si 'nombre' no existe
```

Solución segura:

```
nombre = request.form.get('nombre', '')  # Retorna '' si no existe
```

O valida antes de usar:

```
if 'nombre' in request.form:
    nombre = request.form['nombre']
else:
    # Manejar el error
    return redirect('/')
```

Verificando que Funciona

Pasos para probar:

1. **Ejecuta el servidor**: `python server.py`
2. **Visita** `http://localhost:5000/`
3. **Llena el formulario** con datos de prueba
4. **Envía el formulario**
5. **Verifica** que aparezca en la lista

Si hay errores:

- Revisa la terminal para ver el query SQL generado
- Copia el query y pruébalo en MySQL Workbench
- Verifica que los nombres de las columnas coincidan

🎯 Actividad Práctica: CRUD Usuarios

Actividad: Sistema de Usuarios CR

Vamos a crear una aplicación completa para gestionar usuarios:

Funcionalidades:

- ✅ Ver todos los usuarios (Read)
- ✅ Crear nuevos usuarios (Create)

Base de datos:

- Base de datos `tienda`
- Tabla `usuarios` con: `id`, `nombre`, `apellido`, `email`, `edad`, `created_at`, `updated_at`

Estructura del Proyecto

```
usuarios_app/
   ├── server.py
   ├── config/
   │    └── mysqlconnection.py
   ├── models/
   │    └── usuario.py
   └── templates/
        ├── index.html (mostrar usuarios)
        └── nuevo_usuario.html (formulario)
```

Crea esta estructura antes de empezar.

Paso 1: Crear el Modelo Usuario

```python
from flask_app.config.mysqlconnection import connectToMySQL

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

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

    @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)
```

Paso 2: Ruta para Mostrar Usuarios

```
from flask import Flask, render_template, request, redirect
from flask_app.models.usuario import Usuario

app = Flask(__name__)

@app.route('/')
def index():
    usuarios = Usuario.get_all()
    return render_template('index.html', todos_usuarios=usuarios)
```

Paso 3: Plantilla para Mostrar Usuarios

```
<!DOCTYPE html>
<html>
<head>
    <title>Usuarios</title>
</head>
<body>
    <h1>Lista de Usuarios</h1>

    <table>
        <thead>
            <tr>
                <th>ID</th>
                <th>Nombre</th>
                <th>Apellido</th>
                <th>Email</th>
            </tr>
        </thead>
        <tbody>
            {% for usuario in todos_usuarios %}
            <tr>
                <td>{{ usuario.id }}</td>
                <td>{{ usuario.nombre }}</td>
                <td>{{ usuario.apellido }}</td>
                <td>{{ usuario.email }}</td>
            </tr>
            {% endfor %}
        </tbody>
    </table>

    <a href="/usuarios/nuevo">Agregar Usuario</a>
</body>
</html>
```

Paso 4: Ruta para Mostrar Formulario

```
@app.route('/usuarios/nuevo')
def nuevo_usuario():
    return render_template('nuevo_usuario.html')
```

Paso 5: Plantilla del Formulario

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

    <form action="/usuarios/crear" method="POST">
        <label>Nombre:</label>
        <input type="text" name="nombre"><br>

        <label>Apellido:</label>
        <input type="text" name="apellido"><br>

        <label>Email:</label>
        <input type="email" name="email"><br>

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

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

Paso 6: Ruta POST para Crear Usuario

```
@app.route('/usuarios/crear', methods=['POST'])
def crear_usuario():
    datos = {
        "nombre": request.form['nombre'],
        "apellido": request.form['apellido'],
        "email": request.form['email']
    }
    Usuario.save(datos)
    return redirect('/')
```

💪 Ejercicio: Sistema CRUD Usuarios Completo

¡Hora de practicar!

Crea una aplicación completa de gestión de usuarios:

1. Usa la base de datos `tienda` (ya debe estar creada con la tabla `usuarios`)
2. Crea el modelo `Usuario` con métodos `get_all()` y `save()`
3. Crea ruta GET `/` que muestre todos los usuarios en una tabla
4. Crea ruta GET `/usuarios/nuevo` que muestre el formulario
5. Crea ruta POST `/usuarios/crear` que guarde y redirija
6. Agrega enlaces de navegación entre páginas

Requisitos:

- Usa sentencias preparadas en todas las consultas
- Redirige después de crear un usuario
- Muestra mensaje si no hay usuarios
- El formulario debe incluir: nombre, apellido, email, edad

⏱️ **Tiempo:** 30 minutos

Resumen de Conceptos Clave

1. **Sentencias preparadas**: Usa `%(clave)s` para variables en queries
2. **Inyección SQL**: Siempre usa sentencias preparadas para seguridad
3. **request.form**: Diccionario con datos del formulario
4. **POST-Redirect-GET**: Patrón para manejar formularios
5. **INSERT retorna ID**: El método save() retorna el ID del nuevo registro
6. **redirect()**: Redirige a otra ruta después de procesar POST

Lo Más Importante

✅ **Siempre usa sentencias preparadas** cuando tengas variables en SQL
✅ **Nunca concatenes strings** para crear consultas SQL
✅ **Redirige después de POST** para evitar reenvíos accidentales
✅ **Las claves del diccionario** deben coincidir con los placeholders
✅ **request.form** contiene los datos del formulario enviado

Recursos para Seguir Aprendiendo

- **Documentación Flask**: https://flask.palletsprojects.com/
- **PyMySQL Docs**: https://pymysql.readthedocs.io/
- **OWASP SQL Injection**: https://owasp.org/www-community/attacks/SQL_Injection
- **Flask Request Object**: https://flask.palletsprojects.com/en/2.3.x/api/#flask.Request

Próximos Pasos

En la siguiente clase aprenderemos:

- ✏️ **Actualizar datos** (UPDATE) con formularios de edición
- 🗑️ **Eliminar datos** (DELETE) con confirmación
- 🔗 **JOINs** para obtener datos relacionados
- 🔐 **Validaciones** de datos antes de guardar

Práctica para Casa

Proyecto sugerido:

Crea una aplicación de "Lista de Tareas" (To-Do List):

- Ver todas las tareas
- Crear nuevas tareas
- Cada tarea tiene: título, descripción, fecha de creación

Desafío extra:

- Agrega un campo "completada" (booleano)
- Filtra las tareas completadas vs pendientes

Consejos Finales

💡 **Depuración**: Usa `print()` para ver qué contiene `request.form`
💡 **Errores SQL**: Copia el query de la terminal y pruébalo en MySQL Workbench
💡 **Nombres consistentes**: Usa los mismos nombres en formularios, diccionarios y queries
💡 **Prueba paso a paso**: No intentes hacer todo de una vez, prueba cada parte

🎉 ¡Felicidades!

Ya sabes crear y mostrar datos desde Flask

Has aprendido las operaciones C (Create) y R (Read) del CRUD

? Preguntas

¿Alguna duda sobre GET, POST y sentencias preparadas?

🙌 ¡Excelente Trabajo!

¡Nos vemos en la siguiente clase con UPDATE y DELETE!

No olvides completar todos los ejercicios 📝