🔗 Relaciones entre Modelos

Conectando Tablas con JOINs y Llaves Foráneas

Python Full Stack - Clase 28

¿Qué aprenderemos hoy?

- 🔑 Implementar relaciones uno a muchos en modelos Flask
- 📊 Crear métodos con LEFT JOIN para obtener datos relacionados
- 📝 Parsear datos de JOINs en objetos Python
- 🎯 Trabajar con listas de objetos relacionados
- ⚠️ Manejar nombres de columnas ambiguos en JOINs
- 🎮 Integrar relaciones en controladores y vistas

🔗 Implementando Relaciones en Flask

Recordatorio Rápido

Relación Uno a Muchos:

- Un restaurante tiene muchos tacos
- Un taco pertenece a un restaurante
- **Llave foránea:** `restaurante_id` en la tabla `tacos`

Crear el Modelo con Relación

Modelo Restaurante:

```python
# flask_app/models/restaurante.py
from flask_app.config.mysqlconnection import connectToMySQL

class Restaurante:
    def __init__(self, data):
        self.id = data['id']
        self.nombre = data['nombre']
        self.created_at = data['created_at']
        self.updated_at = data['updated_at']
        # Lista vacía para almacenar los tacos relacionados
        self.tacos = []
```

¿Por qué `self.tacos = []`?

- Prepararemos esta lista para guardar objetos `Taco` relacionados
- Más adelante la llenaremos con datos de la base de datos

Métodos Básicos del Modelo Restaurante

```python
@classmethod
def save(cls, datos):
    """Guarda un nuevo restaurante"""
    query = "INSERT INTO restaurantes (nombre) VALUES (%(nombre)s)"
    return connectToMySQL('esquema_tacos').query_db(query, datos)

@classmethod
def get_all(cls):
    """Obtiene todos los restaurantes"""
    query = "SELECT * FROM restaurantes;"
    resultados = connectToMySQL('esquema_tacos').query_db(query)
    restaurantes = []
    for restaurante in resultados:
        restaurantes.append(cls(restaurante))
    return restaurantes
```

Estos métodos son iguales a los que ya conocemos.

Actualizar el Modelo Taco

Agregar `restaurante_id` al guardar:

```
# flask_app/models/taco.py
@classmethod
def save(cls, datos):
    query = """
        INSERT INTO tacos
        (tortilla, guiso, salsa, restaurante_id)
        VALUES(%(tortilla)s, %(guiso)s, %(salsa)s, %(restaurante_id)s);
    """
    return connectToMySQL('esquema_tacos').query_db(query, datos)
```

Cambio importante:

- Ahora incluimos `restaurante_id` en el INSERT
- Este valor viene del formulario

Actualizar el Formulario

Agregar un `<select>` para elegir restaurante:

```
<!-- flask_app/templates/index.html -->
<form action="/crear" method="post">
    <!-- ... otros campos del formulario ... -->

    <div class="form-group">
        <label for="restaurante_id">Restaurante:</label>
        <select name="restaurante_id" id="restaurante_id">
            {% for restaurante in todos_restaurantes %}
            <option value="{{restaurante.id}}">
                {{restaurante.nombre}}
            </option>
            {% endfor %}
        </select>
    </div>
    <button type="submit">Crear Taco</button>
</form>
```

Actualizar el Controlador

Enviar la lista de restaurantes al formulario:

```
# flask_app/controllers/tacos.py
from flask_app.models.restaurante import Restaurante
from flask_app.models.taco import Taco

@app.route('/')
def index():
    # Obtener todos los restaurantes para el select
    todos_restaurantes = Restaurante.get_all()
    return render_template("index.html", todos_restaurantes=todos_restaurantes)

@app.route('/crear', methods=['POST'])
def crear():
    datos = {
        "tortilla": request.form['tortilla'],
        "guiso": request.form['guiso'],
        "salsa": request.form['salsa'],
        "restaurante_id": request.form['restaurante_id']  # Nuevo campo
    }
    Taco.save(datos)
    return redirect('/tacos')
```

🔍 Obtener Datos Relacionados con JOINs

El Problema: Obtener Datos Relacionados

Necesitamos: Ver un restaurante con todos sus tacos.

Solución: Usar **LEFT JOIN** en el modelo.

Recordatorio: LEFT JOIN garantiza obtener el restaurante aunque no tenga tacos.

Paso 1: Importar la Clase Relacionada

```
# flask_app/models/restaurante.py
from flask_app.config.mysqlconnection import connectToMySQL
from flask_app.models import taco  # ⚠️ Importar la clase Taco
```

Importante: Necesitamos importar `taco` para crear objetos Taco.

Paso 2: Crear el Método con LEFT JOIN

```python
@classmethod
def get_restaurante_y_tacos(cls, datos):
    # 1. Query con LEFT JOIN
    query = """
        SELECT * FROM restaurantes
        LEFT JOIN tacos ON tacos.restaurante_id = restaurantes.id
        WHERE restaurantes.id = %(id)s;
    """
    resultados = connectToMySQL('esquema_tacos').query_db(query, datos)

    # 2. Crear objeto Restaurante con el primer registro
    restaurante = cls(resultados[0])

    # 3. Parsear cada taco y agregarlo a la lista
    for fila_en_db in resultados:
        if fila_en_db['tacos.id']:  # Verificar que existe
            datos_taco = {
                "id": fila_en_db['tacos.id'],           # ⚠️ tabla.columna
                "tortilla": fila_en_db['tortilla'],
                "guiso": fila_en_db['guiso'],
                "salsa": fila_en_db['salsa'],
                "restaurante_id": fila_en_db['restaurante_id'],
                "created_at": fila_en_db['tacos.created_at'],  # ⚠️ tabla.columna
                "updated_at": fila_en_db['tacos.updated_at'],  # ⚠️ tabla.columna
            }
            restaurante.tacos.append(taco.Taco(datos_taco))

    return restaurante
```

Pasos clave:

1. LEFT JOIN une las tablas
2. Primer registro → crear Restaurante
3. Iterar resultados → crear Tacos y agregarlos a la lista

⚠️ Regla Importante: tabla.columna

Cuando columnas se repiten (`id` , `created_at` , `updated_at`):

- `fila_en_db['id']` → id del restaurante
- `fila_en_db['tacos.id']` → id del taco
- `fila_en_db['tacos.created_at']` → created_at del taco

Cuando NO se repiten (tortilla, guiso, salsa):

- `fila_en_db['tortilla']` → funciona directamente

Regla: Si se repite → usa `tabla.columna` . Si no se repite → usa solo el nombre.

Paso 3: Usar en el Controlador

```
# flask_app/controllers/tacos.py
from flask_app.models.restaurante import Restaurante

@app.route('/restaurante/<int:restaurante_id>')
def mostrar_restaurante(restaurante_id):
    datos = {'id': restaurante_id}
    restaurante = Restaurante.get_restaurante_y_tacos(datos)
    return render_template("restaurante.html", restaurante=restaurante)
```

Paso 4: Mostrar en la Vista

```
<!-- templates/restaurante.html -->
<h1>{{ restaurante.nombre }}</h1>

<h2>Tacos de este restaurante:</h2>
{% for taco in restaurante.tacos %}
    <p>{{ taco.tortilla }} - {{ taco.guiso }} - {{ taco.salsa }}</p>
{% endfor %}
```

¡Listo! Ya puedes mostrar un restaurante con todos sus tacos.

Errores Comunes

Error 1: No usar tabla.columna

```
# ❌ INCORRECTO - Ambiguo
datos_taco = {
    "id": fila_en_db['id'],  # ¿Cuál id? ¿Restaurante o taco?
}

# ✅ CORRECTO - Específico
datos_taco = {
    "id": fila_en_db['tacos.id'],  # Claramente el id del taco
}
```

Errores Comunes (continuación)

Error 2: No verificar si existe el taco

```
# ❌ INCORRECTO - Puede crear objetos vacíos
for fila_en_db in resultados:
    datos_taco = {...}
    restaurante.tacos.append(taco.Taco(datos_taco))

# ✅ CORRECTO - Verifica primero
for fila_en_db in resultados:
    if fila_en_db['tacos.id']:  # Verificar que existe
        datos_taco = {...}
        restaurante.tacos.append(taco.Taco(datos_taco))
```

Errores Comunes (continuación)

Error 3: Usar INNER JOIN en lugar de LEFT JOIN

```
# ❌ INCORRECTO - No muestra restaurantes sin tacos
query = """
    SELECT * FROM restaurantes
    INNER JOIN tacos ON ...
"""

# ✅ CORRECTO - Muestra todos los restaurantes
query = """
    SELECT * FROM restaurantes
    LEFT JOIN tacos ON ...
"""
```

Flujo Completo de una Relación

```
1. Usuario visita /restaurante/1
   ↓
2. Controlador llama Restaurante.get_restaurante_y_tacos({'id': 1})
   ↓
3. Modelo ejecuta LEFT JOIN
   ↓
4. Base de datos devuelve lista de diccionarios
   ↓
5. Modelo crea objeto Restaurante
   ↓
6. Modelo parsea cada taco y lo agrega a restaurante.tacos
   ↓
7. Controlador recibe objeto Restaurante con lista de Tacos
   ↓
8. Controlador renderiza plantilla con restaurante
   ↓
9. Plantilla itera sobre restaurante.tacos
```

📋 Checklist de Implementación

En el Modelo:

- [ ] `self.tacos = []` en `__init__`
- [ ] Importar clase relacionada: `from flask_app.models import taco`
- [ ] Método con LEFT JOIN
- [ ] Usar `tabla.columna` para columnas repetidas
- [ ] Verificar `if fila_en_db['tacos.id']:` antes de crear objetos

En el Controlador:

- [ ] Importar modelo: `from flask_app.models.restaurante import Restaurante`
- [ ] Llamar método: `Restaurante.get_restaurante_y_tacos(datos)`
- [ ] Pasar a plantilla: `render_template("restaurante.html", restaurante=restaurante)`

En la Vista:

Recursos para Seguir Aprendiendo

📚 **Documentación oficial:**

- MySQL JOINs: https://dev.mysql.com/doc/refman/8.0/en/join.html
- PyMySQL: https://pymysql.readthedocs.io/

💡 **Próximos pasos:**

- Relaciones muchos a muchos
- Múltiples JOINs en una consulta
- Optimización de consultas

Próximos Pasos

En la siguiente clase veremos:

- 🔗 **Relaciones muchos a muchos** (tablas intermedias)
- 📊 **Múltiples JOINs** en una sola consulta
- 🎯 **Consultas más complejas** con múltiples relaciones

Práctica para Casa

🏠 **Proyecto sugerido:**

Completa el proyecto de tacos:

- Crea varios restaurantes
- Crea tacos asociados a diferentes restaurantes
- Crea una página que muestre todos los restaurantes
- Cada restaurante debe mostrar cuántos tacos tiene
- Crea una página de detalle de restaurante con todos sus tacos

Objetivo: Practicar relaciones uno a muchos y JOINs.

Consejos Finales

✅ **Siempre usa LEFT JOIN** para relaciones uno a muchos
✅ **Verifica que existe** antes de crear objetos relacionados
✅ **Usa tabla.columna** para evitar ambigüedad
✅ **Prueba casos extremos**: restaurante sin tacos, restaurante con muchos tacos
✅ **Usa print()** para depurar y ver qué devuelve el JOIN

¡La práctica hace al maestro! 💪

🎉 ¡Felicidades!

Ya sabes trabajar con relaciones entre modelos

Has aprendido a conectar tablas con JOINs y llaves foráneas

? Preguntas

¿Alguna duda sobre relaciones, JOINs o parsear datos?

🙌 ¡Excelente Trabajo!

¡Nos vemos en la siguiente clase con relaciones muchos a muchos!

No olvides completar todos los ejercicios 📝