

M3.3 API BD Sql



Ayúdame a crear una API en Python para extraer registros de la tabla "Curso" para una base de datos MySQL



Crear una API en Python para extraer registros de una tabla MySQL implica varios pasos, desde la configuración de la conexión con la base de datos hasta la creación de endpoints para interactuar con los datos. Vamos a usar `Flask` como framework para la API y `PyMySQL` para conectarnos a MySQL. A continuación te muestro un ejemplo básico de cómo puedes hacerlo.

1. Instalación de Dependencias

Primero, necesitas instalar las librerías necesarias:

```
bash
```

```
pip install flask pymysql
```

2. Conectar con la Base de Datos MySQL

Crea un archivo llamado `app.py` y configura la conexión con la base de datos:

```
python
```

```
from flask import Flask, jsonify
import pymysql

app = Flask(__name__)

# Configuración de la conexión a MySQL
def get_db_connection():
    return pymysql.connect(
        host='localhost',      # Cambia esto según sea necesario
        user='tu_usuario',
        password='tu_contraseña',
        db='nombre_de_tu_base_de_datos',
        cursorclass=pymysql.cursors.DictCursor
    )

# Ruta para obtener todos los registros de la tabla "Curso"
@app.route('/cursos', methods=['GET'])
def get_cursos():
    connection = get_db_connection()
    try:
```

```

        with connection.cursor() as cursor:
            sql = "SELECT * FROM Curso"
            cursor.execute(sql)
            result = cursor.fetchall()
            return jsonify(result)
    finally:
        connection.close()

# Ruta para obtener un curso específico por ID
@app.route('/curso/<int:curso_id>', methods=['GET'])
def get_curso(curso_id):
    connection = get_db_connection()
    try:
        with connection.cursor() as cursor:
            sql = "SELECT * FROM Curso WHERE id = %s"
            cursor.execute(sql, (curso_id,))
            result = cursor.fetchone()
            if result:
                return jsonify(result)
            else:
                return jsonify({"error": "Curso no encontrado"}), 404
    finally:
        connection.close()

if __name__ == '__main__':
    app.run(debug=True)

```

3. Explicación del Código

- **Conexión a la Base de Datos:** La función `get_db_connection()` establece una conexión a la base de datos MySQL utilizando la librería `pymysql`.
- **Ruta `/cursos`:** Este endpoint maneja las solicitudes GET y devuelve todos los registros de la tabla "Curso".
- **Ruta `/curso/<int:curso_id>`:** Este endpoint maneja solicitudes GET para obtener un curso específico basado en el `curso_id` proporcionado en la URL.

4. Ejecutar la API

Guarda el archivo `app.py` y ejecuta el script:

```
bash
```

```
python app.py
```

La API estará disponible en `http://localhost:5000`. Puedes hacer solicitudes GET a los endpoints `/cursos` y `/curso/<int:curso_id>` para probar la API.

5. Mejoras Sugeridas

- **Control de Errores:** Implementar manejo de errores más robusto.
- **Autenticación:** Agregar autenticación para proteger la API.
- **Paginación:** Implementar paginación para manejar grandes volúmenes de datos.

Este es un ejemplo básico para que puedas empezar. Si necesitas alguna funcionalidad adicional o ajustes específicos, no dudes en pedírmelo.



Ahora genera una API en python para que por medio del método GET actualice un registro específico en la tabla "Curso" para una base de datos MySQL



Para actualizar un registro en la tabla "Curso" utilizando un método GET no es la práctica más común ni recomendada, ya que generalmente los métodos GET se utilizan para obtener información, mientras que los métodos como POST, PUT o PATCH son más adecuados para realizar actualizaciones. Sin embargo, te mostraré cómo hacerlo bajo la premisa solicitada. Es importante que solo uses este enfoque en situaciones controladas y que consideres cambiarlo a POST, PUT o PATCH para un entorno de producción.

1. Modificación del Código

Primero, asegúrate de que el archivo `app.py` está configurado con las rutas anteriores, luego agrega una nueva ruta para actualizar un curso utilizando el método GET.

python

```
from flask import Flask, jsonify, request
import pymysql

app = Flask(__name__)

# Configuración de la conexión a MySQL
def get_db_connection():
    return pymysql.connect(
        host='localhost',
        user='tu_usuario',
        password='tu_contraseña',
        db='nombre_de_tu_base_de_datos',
        cursorclass=pymysql.cursors.DictCursor
    )

# Ruta para obtener todos los registros de la tabla "Curso"
@app.route('/cursos', methods=['GET'])
def get_cursos():
    connection = get_db_connection()
    try:
        with connection.cursor() as cursor:
            sql = "SELECT * FROM Curso"
            cursor.execute(sql)
            result = cursor.fetchall()
            return jsonify(result)
    finally:
        connection.close()

# Ruta para obtener un curso específico por ID
@app.route('/curso/<int:curso_id>', methods=['GET'])
```

```
def get_curso(curso_id):
    connection = get_db_connection()
    try:
        with connection.cursor() as cursor:
            sql = "SELECT * FROM Curso WHERE id = %s"
            cursor.execute(sql, (curso_id,))
            result = cursor.fetchone()
            if result:
                return jsonify(result)
            else:
                return jsonify({"error": "Curso no encontrado"}), 404
    finally:
        connection.close()

# Ruta para actualizar un curso por ID (usando método GET)
@app.route('/actualizar_curso/<int:curso_id>', methods=['GET'])
def actualizar_curso(curso_id):
    nuevo_nombre = request.args.get('nombre')
    nuevo_descripcion = request.args.get('descripcion')
    nuevo_profesor = request.args.get('profesor')

    if not (nuevo_nombre or nuevo_descripcion or nuevo_profesor):
        return jsonify({"error": "Faltan parámetros de actualización"}), 400

    connection = get_db_connection()
    try:
        with connection.cursor() as cursor:
            sql = "UPDATE Curso SET"
            updates = []
            params = []

            if nuevo_nombre:
                updates.append(" nombre = %s")
                params.append(nuevo_nombre)
            if nuevo_descripcion:
                updates.append(" descripcion = %s")
                params.append(nuevo_descripcion)
            if nuevo_profesor:
                updates.append(" profesor = %s")
                params.append(nuevo_profesor)

            sql += ", ".join(updates)
            sql += " WHERE id = %s"
            params.append(curso_id)

            cursor.execute(sql, tuple(params))
            connection.commit()

            if cursor.rowcount > 0:
                return jsonify({"message": "Curso actualizado correctamente"})
            else:
                return jsonify({"error": "Curso no encontrado o sin cambios"}), 404
    finally:
        connection.close()

if __name__ == '__main__':
    app.run(debug=True)
```

2. Explicación del Código

- **Ruta** ``/actualizar_curso/<int:curso_id>``: Este endpoint maneja solicitudes GET para actualizar un curso. Los parámetros que deseas actualizar se pasan como argumentos de la URL, por ejemplo:

```
bash
```

```
http://localhost:5000/actualizar_curso/1?
nombre=NuevoNombre&descripcion=NuevaDescripcion&profesor=NuevoProfesor
```

- ``nombre``: Nuevo nombre para el curso.
- ``descripcion``: Nueva descripción del curso.
- ``profesor``: Nuevo nombre del profesor.
- **Validación de Parámetros:** El código verifica si al menos uno de los parámetros (``nombre``, ``descripcion``, ``profesor``) se proporciona; si no, devuelve un error 400.
- **Actualización Condicional:** Solo se actualizan los campos que se proporcionan en la solicitud.

3. Ejecución

Guarda el archivo ``app.py`` y ejecútalo de nuevo:

```
bash
```

```
python app.py
```

Ahora puedes hacer una solicitud GET para actualizar un curso. Por ejemplo:

```
bash
```

```
http://localhost:5000/actualizar_curso/1?nombre=NuevoNombre
```

Esto actualizará el nombre del curso con ``id = 1`` a "NuevoNombre".

4. Advertencia

Recuerda que, en una aplicación real, deberías utilizar métodos más apropiados como POST o PUT para actualizar datos. Utilizar GET para realizar cambios puede llevar a problemas de seguridad y violar las convenciones de REST.



Ahora genera una API en python para que por medio del método POST actualice un registro específico en la tabla "Curso" para una base de datos MySQL



Actualizar un registro en una base de datos utilizando el método POST es una práctica común y recomendada. A continuación, te mostraré cómo crear una API en Python utilizando Flask para actualizar un registro en la tabla "Curso" de una base de datos MySQL.

1. Modificación del Código

Primero, asegúrate de que el archivo `app.py` tiene las rutas anteriores, y luego agrega una nueva ruta para actualizar un curso utilizando el método POST.

python

```
from flask import Flask, jsonify, request
import pymysql

app = Flask(__name__)

# Configuración de la conexión a MySQL
def get_db_connection():
    return pymysql.connect(
        host='localhost',
        user='tu_usuario',
        password='tu_contraseña',
        db='nombre_de_tu_base_de_datos',
        cursorclass=pymysql.cursors.DictCursor
    )

# Ruta para obtener todos los registros de la tabla "Curso"
@app.route('/cursos', methods=['GET'])
def get_cursos():
    connection = get_db_connection()
    try:
        with connection.cursor() as cursor:
            sql = "SELECT * FROM Curso"
            cursor.execute(sql)
            result = cursor.fetchall()
            return jsonify(result)
    finally:
        connection.close()

# Ruta para obtener un curso específico por ID
@app.route('/curso/<int:curso_id>', methods=['GET'])
def get_curso(curso_id):
    connection = get_db_connection()
    try:
        with connection.cursor() as cursor:
            sql = "SELECT * FROM Curso WHERE id = %s"
            cursor.execute(sql, (curso_id,))
            result = cursor.fetchone()
            if result:
                return jsonify(result)
            else:
                return jsonify({"error": "Curso no encontrado"}), 404
    finally:
        connection.close()

# Ruta para actualizar un curso por ID (usando método POST)
@app.route('/actualizar_curso/<int:curso_id>', methods=['POST'])
def actualizar_curso(curso_id):
    datos = request.json
    nuevo_nombre = datos.get('nombre')
    nuevo_descripcion = datos.get('descripcion')
    nuevo_profesor = datos.get('profesor')
```

```

if not (nuevo_nombre or nuevo_descripcion or nuevo_profesor):
    return jsonify({"error": "Faltan parámetros de actualización"}), 400

connection = get_db_connection()
try:
    with connection.cursor() as cursor:
        sql = "UPDATE Curso SET"
        updates = []
        params = []

        if nuevo_nombre:
            updates.append(" nombre = %s")
            params.append(nuevo_nombre)
        if nuevo_descripcion:
            updates.append(" descripcion = %s")
            params.append(nuevo_descripcion)
        if nuevo_profesor:
            updates.append(" profesor = %s")
            params.append(nuevo_profesor)

        sql += ", ".join(updates)
        sql += " WHERE id = %s"
        params.append(curso_id)

        cursor.execute(sql, tuple(params))
        connection.commit()

        if cursor.rowcount > 0:
            return jsonify({"message": "Curso actualizado correctamente"})
        else:
            return jsonify({"error": "Curso no encontrado o sin cambios"}), 404
finally:
    connection.close()

if __name__ == '__main__':
    app.run(debug=True)

```

2. Explicación del Código

- **Ruta** `/actualizar_curso/<int:curso_id>`: Este endpoint maneja solicitudes POST para actualizar un curso. Los datos a actualizar se envían en el cuerpo de la solicitud en formato JSON.
- **Datos Esperados**: El JSON enviado en el cuerpo de la solicitud puede contener los siguientes campos:
 - `nombre`: Nuevo nombre para el curso.
 - `descripcion`: Nueva descripción del curso.
 - `profesor`: Nuevo nombre del profesor.

Por ejemplo:

json

```

{
  "nombre": "NuevoNombre",
  "descripcion": "NuevaDescripcion",
  "profesor": "NuevoProfesor"
}

```

- **Validación de Parámetros:** El código verifica si al menos uno de los parámetros (``nombre``, ``descripcion``, ``profesor``) se proporciona en el cuerpo de la solicitud; si no, devuelve un error 400.
- **Actualización Condicional:** Solo se actualizan los campos que se proporcionan en la solicitud.

3. Ejecución

Guarda el archivo ``app.py`` y ejecútalo:

```
bash
```

```
python app.py
```

Ahora puedes hacer una solicitud POST para actualizar un curso. Por ejemplo, utilizando ``curl`` o una herramienta como Postman:

```
bash
```

```
curl -X POST http://localhost:5000/actualizar_curso/1 -H "Content-Type: application/json" -d '{"nombre": "NuevoNombre"}
```

Esto actualizará el nombre del curso con ``id = 1`` a "NuevoNombre".

4. Ventajas de Usar POST

- **Seguridad:** Los datos se envían en el cuerpo de la solicitud, lo que permite enviar más información de manera segura.
- **Conformidad con REST:** Es más adecuado para actualizaciones parciales o totales en recursos existentes.

Este enfoque es más seguro y sigue las convenciones de REST, lo que lo hace más apropiado para una aplicación en producción.

ChatGPT can make mistakes. Check important info.