

M3.1 BD Nosq



Qué es una base de datos NoSql ?



Una base de datos NoSQL (Not Only SQL) es un tipo de base de datos que no sigue el modelo relacional tradicional basado en tablas y SQL (Structured Query Language). En lugar de eso, las bases de datos NoSQL están diseñadas para manejar grandes volúmenes de datos no estructurados o semiestructurados, ofreciendo flexibilidad y escalabilidad.

Características principales de las bases de datos NoSQL:

1. **Modelos de datos flexibles:** No requieren un esquema predefinido, lo que permite almacenar datos en formatos variados, como documentos, gráficos, columnas, o pares clave-valor.
2. **Alta escalabilidad:** Están diseñadas para escalar horizontalmente, lo que significa que se pueden agregar más servidores para manejar el incremento de datos y la carga de trabajo.
3. **Desempeño rápido:** Optimizadas para manejar grandes volúmenes de datos y accesos rápidos, las bases de datos NoSQL pueden ser más eficientes en ciertas aplicaciones que requieren consultas rápidas y de baja latencia.
4. **Distribución geográfica:** Pueden estar distribuidas a través de múltiples ubicaciones geográficas, mejorando la disponibilidad y la resistencia a fallos.

Tipos de bases de datos NoSQL:

- **Documentales:** Almacenan datos en documentos similares a JSON. Ejemplo: MongoDB.
- **Claves-valor:** Guardan datos como pares de clave y valor. Ejemplo: Redis.
- **Columnas:** Almacenan datos en tablas, pero optimizadas para trabajar con columnas en lugar de filas. Ejemplo: Cassandra.
- **Grafos:** Están diseñadas para almacenar y consultar relaciones complejas entre datos. Ejemplo: Neo4j.

Estas bases de datos son especialmente útiles en aplicaciones modernas como redes sociales, análisis en tiempo real, y aplicaciones que manejan grandes volúmenes de datos no estructurados.



Cuando se habla de API, ¿a qué se refiere un endpoint?



Un **endpoint** en el contexto de una API (Interfaz de Programación de Aplicaciones) es una URL específica dentro del servidor que representa una función, recurso o servicio particular que la API ofrece. Es el punto de acceso donde los clientes (como aplicaciones web, móviles u otros servicios) pueden interactuar con la API para realizar operaciones como obtener datos, enviar datos, actualizar o eliminar información.

Ejemplo sencillo de cómo funcionan los endpoints en una API:

- **API URL base:** ``https://api.ejemplo.com``
- **Endpoint para obtener información de usuario:** ``https://api.ejemplo.com/usuarios/{id}``

Aquí, el **endpoint** ``https://api.ejemplo.com/usuarios/{id}`` podría ser utilizado para obtener los detalles de un usuario específico reemplazando ``{id}`` con un número de identificación particular, como ``https://api.ejemplo.com/usuarios/123``.

Tipos de operaciones que suelen manejar los endpoints:

- **GET:** Recuperar datos de un recurso (ej., obtener información de un usuario).
- **POST:** Crear un nuevo recurso (ej., agregar un nuevo usuario).
- **PUT:** Actualizar un recurso existente (ej., modificar los datos de un usuario).
- **DELETE:** Eliminar un recurso (ej., borrar un usuario).

Cada endpoint está asociado a una acción específica en la API y suele requerir parámetros, como identificadores de recursos o datos en el cuerpo de la solicitud, para realizar la operación deseada.



Qué es un resource, dentro del contexto de las bases de datos NoSql ?



En el contexto de las bases de datos NoSQL, un **resource** generalmente se refiere a un **recurso** que representa una entidad o un objeto en el sistema que puede ser accedido, manipulado o consultado a través de la base de datos. Dependiendo del tipo de base de datos NoSQL, un resource puede tomar diferentes formas:

1. Documentos (en bases de datos documentales):

- En bases de datos como MongoDB, un resource suele ser un **documento** que almacena datos en formato JSON o BSON (Binary JSON). Un documento puede representar cualquier cosa, como un usuario, un producto, o una transacción, con sus correspondientes atributos.
- Ejemplo: Un documento que representa un usuario podría tener atributos como nombre, correo electrónico, y dirección.

2. Pares clave-valor (en bases de datos de clave-valor):

- Aquí, un resource es un **par clave-valor**, donde la clave es un identificador único y el valor es el contenido asociado. Cada par representa un recurso que puede ser accedido usando

la clave.

- Ejemplo: En Redis, una clave puede representar un ID de sesión de usuario, y el valor asociado puede ser la información de la sesión.

3. Columnas o familias de columnas (en bases de datos orientadas a columnas):

- En bases de datos como Cassandra, un resource puede ser un **conjunto de columnas** que representan una entidad dentro de una tabla. Las columnas pueden ser agrupadas en familias de columnas, que son similares a las filas en las bases de datos relacionales pero optimizadas para grandes volúmenes de datos.
- Ejemplo: Una familia de columnas podría almacenar los atributos de un producto, como nombre, precio, y descripción.

4. Nodos y relaciones (en bases de datos de grafos):

- En bases de datos como Neo4j, los resources son **nodos y relaciones**. Los nodos representan entidades, y las relaciones representan las conexiones entre esas entidades.
- Ejemplo: Un nodo puede representar un usuario, y la relación puede representar una amistad entre dos usuarios.

En resumen, un **resource** en una base de datos NoSQL es un componente fundamental que representa datos de una entidad específica, adaptado al modelo de datos de la base de datos (documento, par clave-valor, columna, nodo, etc.).



Ayúdame a crear un programa en Python para consultar una tabla de mockapi.io en la URL <https://66b4e5529f9169621ea4c37d.mockapi.io/api/v1/contactos>, que me muestre todos los registros en JSON formateado, en formato Dataframe y los exporte a un csv



Aquí te dejo un programa en Python que consulta la API de MockAPI, formatea la respuesta en JSON, la convierte a un DataFrame y la exporta a un archivo CSV:

python

```
import requests
import pandas as pd
import json

# URL de la API
url = "https://66b4e5529f9169621ea4c37d.mockapi.io/api/v1/contactos"

# Realiza la solicitud GET a la API
response = requests.get(url)

# Verifica si la solicitud fue exitosa
if response.status_code == 200:
    # Obtiene los datos en formato JSON
    data = response.json()

    # Formatea y muestra los datos en JSON
```

```
print(json.dumps(data, indent=4))

# Convierte los datos a un DataFrame
df = pd.DataFrame(data)

# Muestra el DataFrame
print("\nDataFrame:")
print(df)

# Exporta el DataFrame a un archivo CSV
csv_filename = "contactos.csv"
df.to_csv(csv_filename, index=False)
print(f"\nDatos exportados a {csv_filename} exitosamente.")

else:
    print(f"Error al consultar la API. Código de estado: {response.status_code}")
```

Explicación del código:

1. Importación de librerías:

- ``requests``: Para realizar solicitudes HTTP.
- ``pandas``: Para manejar datos tabulares.
- ``json``: Para manejar y formatear datos en JSON.

2. Realización de la solicitud GET:

- Se hace una solicitud GET a la URL proporcionada.
- Se verifica si la solicitud fue exitosa comprobando que el código de estado sea ``200``.

3. Manejo de datos:

- Si la solicitud es exitosa, los datos se obtienen en formato JSON usando ``response.json()``.
- Los datos JSON se formatean y se imprimen con ``json.dumps(data, indent=4)`` para que sean más legibles.
- Se convierte el JSON a un DataFrame de pandas para su manipulación tabular.

4. Exportación a CSV:

- Se exporta el DataFrame a un archivo CSV llamado ``contactos.csv`` sin incluir índices.

5. Error handling:

- Si la solicitud falla, se muestra un mensaje con el código de estado.

Ejecución:

1. Guarda el código en un archivo Python (por ejemplo, ``consulta_contactos.py``).
2. Asegúrate de tener instaladas las librerías necesarias. Puedes instalarlas con:

```
bash
```

```
pip install requests pandas
```

3. Ejecuta el script:

```
bash
```

```
python consulta_contactos.py
```

Esto te mostrará los datos obtenidos de la API en formato JSON y DataFrame, y luego los exportará a un archivo `contactos.csv` en el mismo directorio donde se ejecute el script.



Ahora, ayúdame con el programa para mostrar solamente un determinado registro de la misma URL de MockAPI, muestra todos los campos en formato plano y ya no lo muestres en DataFrame ni lo exportes a CSV



Aquí tienes un programa en Python que permite consultar un registro específico de la API de MockAPI, mostrando todos los campos en formato plano:

```
python
```

```
import requests
import json

# URL base de la API
url_base = "https://66b4e5529f9169621ea4c37d.mockapi.io/api/v1/contactos"

# ID del registro que deseas consultar
registro_id = "1" # Cambia este valor al ID del registro que quieras obtener

# URL completa con el ID del registro
url = f"{url_base}/{registro_id}"

# Realiza la solicitud GET a la API
response = requests.get(url)

# Verifica si la solicitud fue exitosa
if response.status_code == 200:
    # Obtiene los datos en formato JSON
    data = response.json()

    # Muestra todos los campos del registro en formato plano
    print("Datos del registro:")
    for key, value in data.items():
        print(f"{key}: {value}")
else:
    print(f"Error al consultar la API. Código de estado: {response.status_code}")
```

Explicación del código:

1. Definición de la URL base:

- ``url_base`` es la URL base de la API sin incluir un ID específico.

2. Especificación del registro a consultar:

- ``registro_id`` es el ID del registro que deseas obtener. Puedes cambiar el valor ``"1"`` a cualquier otro ID de un registro existente.

3. Construcción de la URL completa:

- Se construye la URL completa combinando la URL base y el ID del registro.

4. Realización de la solicitud GET:

- Se hace una solicitud GET a la URL específica del registro.
- Se verifica si la solicitud fue exitosa comprobando que el código de estado sea ``200``.

5. Muestra de los datos en formato plano:

- Si la solicitud es exitosa, los datos se muestran en un formato simple, iterando sobre cada clave y valor del diccionario JSON.

6. Manejo de errores:

- Si la solicitud falla, se muestra un mensaje con el código de estado.

Ejecución:

1. Guarda el código en un archivo Python (por ejemplo, ``consulta_registro.py``).
2. Asegúrate de tener instalada la librería ``requests``. Si no la tienes, instálala con:

```
bash
```

```
pip install requests
```

3. Ejecuta el script:

```
bash
```

```
python consulta_registro.py
```

Resultado:

El programa mostrará los datos del registro especificado en un formato sencillo, por ejemplo:

```
yaml
```

```
Datos del registro:
id: 1
nombre: Juan Pérez
telefono: 555-1234
email: juan.perez@example.com
```

Puedes cambiar el valor de ``registro_id`` en el script para consultar diferentes registros según su ID.



Ahora dame el programa para agregar un registro en la misma URL de MockAPI



Aquí tienes un programa en Python que te permite agregar un nuevo registro a la API de MockAPI utilizando una solicitud POST:

python

```
import requests
import json

# URL de la API
url = "https://66b4e5529f9169621ea4c37d.mockapi.io/api/v1/contactos"

# Datos del nuevo registro que deseas agregar
nuevo_registro = {
    "nombre": "Ana López",
    "telefono": "555-6789",
    "email": "ana.lopez@example.com"
}

# Realiza la solicitud POST a la API para agregar el nuevo registro
response = requests.post(url, json=nuevo_registro)

# Verifica si la solicitud fue exitosa
if response.status_code == 201:
    # Muestra la respuesta de la API con los detalles del nuevo registro agregado
    data = response.json()
    print("Registro agregado exitosamente:")
    for key, value in data.items():
        print(f"{key}: {value}")
else:
    print(f"Error al agregar el registro. Código de estado: {response.status_code}")
```

Explicación del código:

1. URL de la API:

- `url` es la URL de la API donde se enviará la solicitud para agregar un nuevo registro.

2. Datos del nuevo registro:

- `nuevo_registro` es un diccionario que contiene los datos que deseas agregar a la API. Puedes modificar los valores de `nombre`, `telefono`, y `email` según el registro que quieras añadir.

3. Solicitud POST:

- Se realiza una solicitud POST a la API con los datos del nuevo registro. La función `requests.post()` se utiliza para enviar los datos en formato JSON a la URL especificada.

4. Manejo de la respuesta:

- Si la solicitud es exitosa (código de estado 201, que significa que el recurso fue creado), se muestra la respuesta de la API con los detalles del registro agregado.
- Si la solicitud falla, se muestra un mensaje con el código de estado correspondiente.

Ejecución:

1. Guarda el código en un archivo Python (por ejemplo, `agregar_registro.py`).
2. Asegúrate de tener instalada la librería `requests`. Si no la tienes, instálala con:

```
bash
```

```
pip install requests
```

3. Ejecuta el script:

```
bash
```

```
python agregar_registro.py
```

Resultado:

Si la solicitud es exitosa, verás un mensaje similar al siguiente, mostrando los detalles del registro que se agregó:

```
yaml
```

```
Registro agregado exitosamente:  
id: 6  
nombre: Ana López  
telefono: 555-6789  
email: ana.lopez@example.com
```

Este mensaje confirma que el registro fue añadido correctamente a la base de datos de MockAPI. Puedes cambiar los datos del registro para agregar diferentes entradas según lo necesites.



Ahora dame el programa para modificar un registro en la misma URL de MockAPI



Aquí tienes un programa en Python que te permite modificar un registro existente en la API de MockAPI utilizando una solicitud PUT:

```
python
```

```
import requests  
import json
```



```
# URL base de la API
url_base = "https://66b4e5529f9169621ea4c37d.mockapi.io/api/v1/contactos"

# ID del registro que deseas modificar
registro_id = "1" # Cambia este valor al ID del registro que quieras modificar

# Datos actualizados para el registro
datos_actualizados = {
    "nombre": "Carlos García",
    "telefono": "555-9876",
    "email": "carlos.garcia@example.com"
}

# URL completa con el ID del registro
url = f"{url_base}/{registro_id}"

# Realiza la solicitud PUT a la API para modificar el registro
response = requests.put(url, json=datos_actualizados)

# Verifica si la solicitud fue exitosa
if response.status_code == 200:
    # Muestra la respuesta de la API con los detalles del registro modificado
    data = response.json()
    print("Registro modificado exitosamente:")
    for key, value in data.items():
        print(f"{key}: {value}")
else:
    print(f"Error al modificar el registro. Código de estado: {response.status_code}")
```

Explicación del código:

1. URL base de la API:

- `url_base` es la URL base de la API sin incluir un ID específico.

2. Especificación del registro a modificar:

- `registro_id` es el ID del registro que deseas modificar. Cambia el valor `"1"` al ID correspondiente del registro que quieres actualizar.

3. Datos actualizados:

- `datos_actualizados` es un diccionario que contiene los nuevos datos que deseas asignar al registro existente. Puedes modificar los valores de `nombre`, `telefono`, y `email` según lo que desees cambiar.

4. Construcción de la URL completa:

- Se construye la URL completa combinando la URL base y el ID del registro para que la solicitud PUT apunte directamente al registro que deseas modificar.

5. Solicitud PUT:

- Se realiza una solicitud PUT a la API con los datos actualizados. La función `requests.put()` envía los datos en formato JSON a la URL especificada.

6. Manejo de la respuesta:

- Si la solicitud es exitosa (código de estado 200), se muestra la respuesta de la API con los detalles del registro modificado.
- Si la solicitud falla, se muestra un mensaje con el código de estado correspondiente.

Ejecución:

1. Guarda el código en un archivo Python (por ejemplo, `modificar_registro.py`).
2. Asegúrate de tener instalada la librería `requests`. Si no la tienes, instálala con:

```
bash
```

```
pip install requests
```

3. Ejecuta el script:

```
bash
```

```
python modificar_registro.py
```

Resultado:

Si la solicitud es exitosa, verás un mensaje similar al siguiente, mostrando los detalles del registro que se modificó:

```
yaml
```

```
Registro modificado exitosamente:  
id: 1  
nombre: Carlos García  
telefono: 555-9876  
email: carlos.garcia@example.com
```

Este mensaje confirma que el registro fue actualizado correctamente en la base de datos de MockAPI. Puedes cambiar el valor de `registro_id` y `datos_actualizados` para modificar diferentes registros según lo necesites.



Ahora dame el programa para eliminar un registro en la misma URL de MockAPI



Aquí tienes un programa en Python que te permite eliminar un registro existente en la API de MockAPI utilizando una solicitud DELETE:

```
python
```

```
import requests  
  
# URL base de la API
```

```
url_base = "https://66b4e5529f9169621ea4c37d.mockapi.io/api/v1/contactos"

# ID del registro que deseas eliminar
registro_id = "1" # Cambia este valor al ID del registro que quieras eliminar

# URL completa con el ID del registro
url = f"{url_base}/{registro_id}"

# Realiza la solicitud DELETE a la API para eliminar el registro
response = requests.delete(url)

# Verifica si la solicitud fue exitosa
if response.status_code == 200:
    print(f"Registro con ID {registro_id} eliminado exitosamente.")
else:
    print(f"Error al eliminar el registro. Código de estado: {response.status_code}")
```

Explicación del código:

1. URL base de la API:

- `url_base` es la URL base de la API sin incluir un ID específico.

2. Especificación del registro a eliminar:

- `registro_id` es el ID del registro que deseas eliminar. Cambia el valor `"1"` al ID correspondiente del registro que quieras borrar.

3. Construcción de la URL completa:

- Se construye la URL completa combinando la URL base y el ID del registro para que la solicitud DELETE apunte directamente al registro que deseas eliminar.

4. Solicitud DELETE:

- Se realiza una solicitud DELETE a la API utilizando `requests.delete(url)`, lo que indica que el registro especificado debe ser eliminado.

5. Manejo de la respuesta:

- Si la solicitud es exitosa (código de estado 200), se muestra un mensaje confirmando que el registro ha sido eliminado.
- Si la solicitud falla, se muestra un mensaje con el código de estado correspondiente.

Ejecución:

1. Guarda el código en un archivo Python (por ejemplo, `eliminar_registro.py`).
2. Asegúrate de tener instalada la librería `requests`. Si no la tienes, instálala con:

```
bash
```

```
pip install requests
```

3. Ejecuta el script:

```
bash
```

```
python eliminar_registro.py
```

Resultado:

Si la solicitud es exitosa, verás un mensaje similar al siguiente:

```
Registro con ID 1 eliminado exitosamente.
```

Este mensaje confirma que el registro fue eliminado correctamente de la base de datos de MockAPI. Puedes cambiar el valor de `registro_id` para eliminar diferentes registros según sea necesario.

ChatGPT can make mistakes. Check important info.