Comandos pymongo vs mongodb

Juan Carlos Pérez González

Conexión

```
mongodb://user123:pass123@ds123456.mlab.com:12345/bbdd
```

```python
import csv
from pymongo import MongoClient

# Conexión a MongoDB
client = MongoClient('mongodb://172.21.0.3:27017/')
db = client['compras']    # creamos la base de datos
collection = db['registros']  # la colección

# Nombre del archivo CSV
filename = 'RegistroComprasOnline.csv'

# Insertar registros en la colección
with open(filename, 'r') as file:
    reader = csv.DictReader(file)
    for row in reader:
        # Insertar cada fila como un documento en la colección
        collection.insert_one(row)

print("Registros insertados exitosamente en MongoDB.")
```

```
Registros insertados exitosamente en MongoDB.
```

Conexión

```
]: from pymongo import MongoClient
import pandas as pd

# Conexión a MongoDB buscar ip con #docker inspect NUMEROCONTENEDOR O NOMBRE CONTENEDOR
client = MongoClient('mongodb://172.21.0.3:27017/')

# Imprime el nombre de todas las bases de datos disponibles
print(client.list_database_names())

db = client['compras']

# IMPRIME LAS COLECCIONES DE ESA BASE DEDATOS
print(db.list_collection_names())

# muestras los documentos que hay en la colección registros
registros = db['registros']

df = pd.DataFrame(list(registros.find()))

display(df)
```

```
['admin', 'bbdd', 'compras', 'config', 'local']
['registros']
```

	_id	IDCliente	Importecompra	Fechacompra	Numerodeclicks	Fechadenacimiento	Codigopostalenvio	Tiemporealizacioncompra	Clienteexistente	Producto
0	65db73305553f4da98adf244	5747	707.833723	09/03/22	5	22/09/85	2246	25	SI	MONITOR
1	65db73305553f4da98adf245	8383	196.881215	02/05/22	1	28/06/84	15568	5	SI	CPU
2	65db73305553f4da98adf247	3922	220.281941	24/06/22	9	18/03/81	7711	45	NO	RATONES Y TECLADOS
3	65db73305553f4da98adf248	3462	771.172393	27/01/21	9	17/04/72	15684	45	NO	ALMACENAMIENTO
4	65db73305553f4da98adf249	6671	44.812645	20/07/19	7	07/11/83	29738	35	NO	ALMACENAMIENTO
...	...	...	...	...	...	...	...	...	...	...

INSERCIONES

Para trabajar con MongoDB desde Python, podemos utilizar el paquete **pymongo**. Veremos una comparativa de los comandos

Método mongoDB	Método pymongo	Parámetros	Qué devuelve
.insertOne()	.insert_one()	Documento	Objeto de tipo InsertOneResult
.insertMany()	.insert_many()	Lista de documentos	Objeto de tipo InsertManyResult

El documento a insertar es un diccionario con un conjunto de pares clave-valor.

```
documento = {
    'ID Cliente': 12345,
    'Estado': "Pagado",
    'Importe': 56.45,
    'Dirección': {
        'Calle': "Avenida de Galicia",
        'Número': 103,
        'Piso': "2B",
        'Código Postal': 36217,
        'Localidad': "Vigo",
        'Provincia': "Pontevedra"
    },
    'Fecha de compra': datetime.datetime(2011, 9, 23),
    'Items': ['Leche', 'Pan']
}
resultadoInsert = registro.insert_one(documento)
```

CONSULTAS - I

Método mongoDB	Método pymongo	Qué devuelve
.findOne()	.find_one()	Diccionario con los datos del documento
.find()	.find()	Cursor a los resultados. Debemos recorrerlo.

```
registro.find_one(filter={'ID Cliente': 12345},projection={'_id': False, 'Importe': True})
```

```
registro.find_one({'ID Cliente': 12345},{'_id': False, 'Importe': True})
```

Si realizamos un ***.find()*** tenemos un cursor que debemos recorrer para acceder a cada documento, accediendo a cada campo por su clave. La ventaja de pymongo es que si no existe el valor da error, mientras Mongo no.

```
for documento in registro.find({'Importe': {'$gt': 10}}):
    print("ID del cliente:", documento['ID Cliente'], "Importe:", documento['Importe'])
```

```
ID del cliente: 12345 Importe: 56.45
ID del cliente: 384 Importe: 23.45
ID del cliente: 9853 Importe: 101.23
```

CONSULTAS - II

Puedes acceder a campos que están en documentos dentro de otros documentos mediante el operador ".", tal y como hacías en la consola de Mongo

```
for documento in registro.find({'Dirección.Calle': "Avenida de Galicia"}):
    print(documento)
```

```
{'_id': ObjectId('630cdfa311a593f149911042'), 'ID Cliente': 12345, 'Estado': 'Pagado', 'Importe': 56.45, 'D
irección': {'Calle': 'Avenida de Galicia', 'Número': 103, 'Piso': '2B', 'Código Postal': 36217, 'Localidad
': 'Vigo', 'Provincia': 'Pontevedra'}, 'Fecha de compra': datetime.datetime(2011, 9, 23, 0, 0), 'Items': ['
Leche', 'Pan']}
```

Método **sort()**

```
for documento in registro.find({'Numerodeclicks': {'$gt': 15}}) \
                         .sort('Numerodeclicks', -1).limit(1):
    print(documento)
```

```
{'_id': ObjectId('63091374f44e52566ed976cf'), 'IDCliente': 5092, 'Importecompra': '708,612348634111', 'Fech
acompra': '15/03/15', 'Categoria': 'ALMACENAMIENTO', 'Numerodeclicks': 20, 'Fechadenacimiento': '26/01/95',
'Codigopostalenvio': 49543, 'Tiemporealizacioncompra': 100, 'Clienteexistente': 'NO'}
```

CONSULTAS - III

Puedes acceder a campos que están en documentos dentro de otros documentos mediante el operador “.”, tal y como hacías en la consola de Mongo

```
for documento in registro.find({'Dirección.Calle': "Avenida de Galicia"}):
    print(documento)
```

```
{'_id': ObjectId('630cdfa311a593f149911042'), 'ID Cliente': 12345, 'Estado': 'Pagado', 'Importe': 56.45, 'D
irección': {'Calle': 'Avenida de Galicia', 'Número': 103, 'Piso': '2B', 'Código Postal': 36217, 'Localidad
': 'Vigo', 'Provincia': 'Pontevedra'}, 'Fecha de compra': datetime.datetime(2011, 9, 23, 0, 0), 'Items': ['
Leche', 'Pan']}
```

Método **sort().** Limit indica cuantos documentos quieres que muestre, en este caso 1, si fuese 5, pues los 5 primeros

```
for documento in registro.find({'Numerodeclicks': {'$gt': 15}}) \
                         .sort('Numerodeclicks', -1).limit(1):
    print(documento)
```

```
{'_id': ObjectId('63091374f44e52566ed976cf'), 'IDCliente': 5092, 'Importecompra': '708,612348634111', 'Fech
acompra': '15/03/15', 'Categoria': 'ALMACENAMIENTO', 'Numerodeclicks': 20, 'Fechadenacimiento': '26/01/95',
'Codigopostalenvio': 49543, 'Tiemporealizacioncompra': 100, 'Clienteexistente': 'NO'}
```

CONSULTAS - IV

Método **count().** Dos formas

```
len(list(registro.find()))
```

```
registro.count_documents({})
```

```
len(list(registro.find()))

2279
```

```
registro.count_documents({})

2279
```

Método **distinct().** Valores distintos de un campo

```
registro.find({'Importe': {'$gt': 5}}).distinct('ID Cliente')
```

```
registro.find({'Importe':

[7864, 9853, 12345]
```

```
registro.distinct('ID Cliente', {'Importe': {'$gt': 5}})
```

```
registro.distinct('ID Cli

[7864, 9853, 12345]
```

CONSULTAS - V

Expresiones regulares. No lo recomiendo por complejidad, aunque son muy potentes. Se usa el operador ***$regex*** y ***re*** en python.

Formato

```
{ <campo>: { '$regex': '<patrón>', '$options': '<opciones>'} }
```

```
import re

resultados = registro.find({'Categoria': re.compile('^D')})
for documento in resultados:
    print(documento)
```

```
{'_id': ObjectId('63091374f44e52566ed976cc'), 'IDCliente': 9087, 'Importecompra': '223,739120385146', 'Fech
acompra': '14/03/15', 'Categoria': 'DESCONOCIDO', 'Numerodeclicks': 11, 'Fechadenacimiento': '11/05/01', 'C
odigopostalenvio': 33936, 'Tiemporealizacioncompra': 55, 'Clienteexistente': 'NO', 'Numero': [illegible], 'OtroNumero
': 0}
{'_id': ObjectId('63091374f44e52566ed976cd'), 'Categoria': 'DESCONOCIDO'}
```

UPDATES - I

Si lo que queremos es realizar update sobre **campos de los documentos encontrados:**

- *.update_one()*
- *.update_many()*

Si lo que queremos es hacer una **actualización completa indicando el documento exacto** a almacenar

- *.replace_one()*

Método mongoDB	Método pymongo	Qué devuelve
.updateOne()	.update_one()	Instancia de UpdateResult
.updateMany()	.update_many()	Instancia de UpdateResult
.replaceOne()	.replace_one()	Instancia de UpdateResult
.findOneAndUpdate()	.find_one_and_update()	Documento (el original o el actualizado)
.findOneAndReplace()	.find_one_and_replace()	Documento (el original o el reemplazado)

UPDATES - II

Salvo los dos últimos métodos que veremos ahora, devuelven una instancia de que incluye entre sus propiedades:

- ***matched_count***: número de registros que cumplían el filtro
- ***modified_count***: número de documentos modificados
- ***upserted_id***: en caso de haber hecho un upsert, el id del documento.
- ***raw_result***: diccionario con el resultado del update.

```
resultado = registro.update_one({'IDCliente': 9087},
        {'$set': {'Importecompra': 125.24}})
```

Actualizamos el valor del campo

UPDATES - III

Otro métodos

- ***.find_one_and_update():*** funciona igual que el .update_one(), pero devuelve el documento afectado.

Por defecto devuelve el documento ANTES de la actualización. Ese comportamiento se puede modificar pasando el parámetro *return_document=ReturnDocument.AFTER*

- ***.find_one_and_replace()***: equivalente a .replace_one(), pero devuelve el documento.

Ejemplo de reemplazo mostrando el **documento nuevo:**

```
documento_nuevo = registro.find_one_and_replace(
    {'IDCliente': 9087}, {'IDCliente': 9087, 'Importecompra': 555},
    return_document=ReturnDocument.AFTER
)
```

Ejemplo de reemplazo mostrando el **documento antiguo**

```
documento_original = registro.find_one_and_update(
    {'IDCliente': 11},
    {'$inc':{'Numerodeclicks': 1}}
)
```

UPSERTS

- Los métodos de update vistos anteriormente permiten realizar un **upsert**. En caso de que ningún documento cumpla el filtro indicado, **se creará un nuevo documento** con la información proporcionada. Para que se ejecute un upsert debes incluir un parámetro adicional: ***upsert=True***

```
documento_original = registro.find_one_and_replace(
    {'IDCliente': 11},
    {'IDCliente': 11, 'Importe': 3.25},
    upsert:True
)

print("DOCUMENTO ANTIGUO:", documento_original)
documento_nuevo = registro.find_one({'IDCliente': 11})
print("DOCUMENTO NUEVO:", documento_nuevo)
```

```
DOCUMENTO ANTIGUO: None
DOCUMENTO NUEVO: {'_id': ObjectId('630e25ddd8a441914d6493bd'), 'IDCliente': 11, 'Importe': 3.25}
```

- Si existiese el documento y ejecutamos con ***upsert=True*** ***.....*** y si no existe:

```
resultado = registro.update_one(
    {'IDCliente': 11},
    {'$set': {'Importe': 22.22}},
    upsert=True)

print("Documentos que coinciden con el filtro:",resultado.matched_count)
print("Documentos modificados:",resultado.modified_count)
print("ID de documento upsertado:", resultado.upserted_id)
```

```
Documentos que coinciden con el filtro: 1
Documentos modificados: 1
ID de documento upsertado: None
```

```
resultado = registro.update_one(
    {'IDCliente': 15},
    {'$set': {'Importe': 1.15}},
    upsert=True)

print("Documentos que coinciden con el filtro:",resultado.matched_count)
print("Documentos modificados:",resultado.modified_count)
print("ID de documento upsertado:", resultado.upserted_id)
```

```
Documentos que coinciden con el filtro: 0
Documentos modificados: 0
ID de documento upsertado: 630e274bd8a441914d649539
```

DELETE

- Los métodos ***Delete***

Método mongoDB	Método pymongo	Qué devuelve
.deleteOne()	.delete_one()	Instancia de DeleteResult
.deleteMany()	.delete_many()	Instancia de DeleteResult
.findOneAndDelete()	.find_one_and_delete()	Documento borrado (o None si no existe)

```
resultado = registro.delete_one({'IDCliente': 15})
```

```
print("Documentos borrados:",resultado.deleted_count)
print("Documento con resultado:", resultado.raw_result)
```

```
Documentos borrados: 1
Documento con resultado: {'n': 1, 'ok': 1.0}
```

```
resultado = registro.delete_many(
    {'ID Cliente': {'$exists': True}}
)
```

```
Documentos borrados: 3
Documento con resultado: {'n': 3, 'ok': 1.0}
```

Agregaciones

- MongoDB dispone de los **pipelines de agregación** para realizar operaciones complejas.
- El **pipeline es una secuencia de 1 a n etapas**, cada una de las cuales hace alguna operación de búsqueda, agrupación o transformación.
- El método a ejecutar sobre la instancia de la colección es ***.aggregate().*** Le pasaremos una lista con las etapas a ejecutar.
  <colección>.aggregate([{<etapa1>}, …, {<etapaN>}])
- Devuelve un cursor con el que recorrer los resultados

```
#Obtenemos los 2 clientes con más pedidos
resultado = registro.aggregate([
    { '$group': { '_id': '$IDCliente', 'Total': { '$sum': 1 } } },
    { '$sort': { 'Total': -1 } },
    { '$limit': 2 }
])

for documento in resultado:
    print(documento)
```

```
{'_id': 9369, 'Total': 4}
{'_id': 3462, 'Total': 4}
```

```
#Operación de agregación
pipeline = [
   {"$group": {"_id": "$pais", "total_ventas": {"$sum": "$cantidad"}}}
]
```

Importación de datos desde MongoDB a un Dataframe - I

- Mongo permite guardar documentos con diferentes esquemas en una misma colección.
- Para conseguirlo, usaremos la función ***list()*** de Python, a la que le pasamos el cursor de la búsqueda

```
import pymongo
import pandas as pd

# Conexión a la base de datos MongoDB
client = pymongo.MongoClient('mongodb://localhost:27017/')
db = client['mi_base_de_datos']
ventas_collection = db['ventas']

# Obtener datos de MongoDB
datos_mongo = list(ventas_collection.find())

# Convertir datos a DataFrame de pandas
df_ventas = pd.DataFrame(datos_mongo)

# Mostrar el DataFrame
print(df_ventas)
```

Importación de datos de MongoDB a un Dataframe

- Para crear el dataframe indicando que IDCliente es el índice ejecutaremos:

```
df = pd.DataFrame.from_records(list(documentos), index='IDCliente')
```

- MongoDB permite tener **documentos con distinto esquema** en una colección, pero un dataframe tiene estructura con lo que en aquellos documentos de una misma colección con **esquemas que tengan distintos campos** Pandas **mostrará NaN**

```
df[df['IDCliente']==1015].T
```

	144	205	2272
IDCliente	1015	1015	1015
Fechacompra	02/01/15	29/07/21	NaN
Categoria	CPU	RATONES Y TECLADOS	NaN
Numerodeclicks	21.0	14.0	NaN
Fechadenacimiento	08/09/70	08/09/70	NaN
Codigopostalenvio	24218.0	40291.0	NaN
Tiemporealizacioncompra	95.0	60.0	NaN
Clienteexistente	NO	NO	NO
Importe de compra	829.971462	545.803349	NaN
DatosCliente	NaN	NaN	NaN
Fechanacimiento	NaN	NaN	NaN
Importecompra	NaN	NaN	102.89
elementos	NaN	NaN	[{'nombre': 'leche', 'cantidad': 5}, {'nombre'...
Cantidad	NaN	NaN	NaN
Importe	NaN	NaN	NaN

Importación de datos de un Dataframe a MongoDB - I

- Pandas dispone un método que transforma un dataframe a una lista de diccionarios.
- Facilita entonces a MongoDB recibir y almacenar datos, ya que sus documentos son pares clave-valor
- El método ***pd.to_dict()*** admite como parámetro el formato de diccionario que queremos generar.
  {columna -> valor}, … , {columna -> valor}
- Que es justo el formato que necesitamos para pasárselo al método ***.insert_many()*** de Pymongo.
- Debemos hacer es revisar si el índice del dataframe debe ser almacenado en MongoDB. Si el índice fuese IDCliente. En ese caso debes pasar el índice a una columna:
  df.reset_index(inplace = True)
- A continuación pasamos los registros a documentos
  documentos = df.to_dict('records')
- Creamos una instancia e insertamos (col_panda es la colección)

  coleccionDestino = bbdd.col_pandas

  resultado = coleccionDestino.insert_many(documentos)
- Los campos que no tengan valor en MongoDB también serán ***NaN***,

```
#Verificamos el número de documentos insertados
print("Se han insertado",len(resultado.inserted_ids),"documentos")
```

```
Se han insertado 2277 documentos
```

```
  elementos: NaN,
  Cantidad: NaN,
  Importe: NaN
}
```

Importación de datos de un Dataframe a MongoDB - II

- Los campos que no tengan valor en MongoDB también serán ***NaN***,

```
  elementos: NaN,
  Cantidad: NaN,
  Importe: NaN
}
```

- Si los queremos eliminar podemos: con *$unset: {campo:1}*

```
#Primera opción, eliminamos de los documentos aquellos campos que tengan valor NaN
campos=['IDCliente','Fechacompra','Categoria','Numerodeclicks','Fechadenacimiento',
'Codigopostalenvio','Tiemporealizacioncompra','Clienteexistente',
'Importe de compra','DatosCliente','Fechanacimiento','Importecompra','elementos',
'Cantidad','Importe']

#Para cada campo lo borramos de los documentos que lo tengan a NaN
for campo in campos:
    resultado = coleccionDestino.update_many({campo: np.nan},{'$unset': {campo: 1}})
    print("Para el campo",campo,"se han actualizado",resultado.modified_count,"documentos")
```

```
Para el campo IDCliente se han actualizado 1 documentos
Para el campo Fechacompra se han actualizado 7 documentos
Para el campo Categoria se han actualizado 6 documentos
Para el campo Numerodeclicks se han actualizado 14 documentos
Para el campo Fechadenacimiento se han actualizado 9 documentos
```

- Otra forma, más sencilla y hasta lógica, antes de insertar, generas un dataframe “limpio” y luego insertas

```
documentos = [dict((clave, valor)
              for clave, valor in zip(df.columns, row)
              if valor != None and valor == valor)
          for row in df.values]
```

VUESTRO TURNO