



Deusto

Facultad de Ingeniería
Ingeniaritza Fakultatea

ARQUITECTURA TECNOLÓGICA PARA BIG DATA

PRÁCTICA 2 – BD NoSQL

Autor

Asier Bujedo Álvarez

Docente

Ana Isabel Torre Bastida

Fecha

2 de diciembre de 2024

Índice de contenido

1.	Base de datos seleccionada y justificación	1
2.	Definición del esquema y las sentencias de creación	3
1.1	Sentencias de creación	4
1.2	Explicación y ejecución de scripts	4
1.2.1	Ejecución.....	5
3.	Inserción de 100 registros	6
4.	Modificación de dos registros.....	7
5.	Consultas específicas	8
5.1	Consulta por una jugadora específica	8
5.1.1	Filtrando por el año de comienzo en el fútbol mayor de 2020	8
5.1.2	Filtrando por un equipo que empiece por “Manchester...”	9
5.2	Consulta por un país concreto donde juega una jugadora.....	10
5.3	Tabla de tiempos de ejecución	11
6.	Código.....	12
6.1	Main.py.....	12
6.2	modifyRegistries.py	14
6.3	queries.py.....	15
7.	Bibliografía	16

Índice de ilustraciones

Ilustración 1 - Registro de muestra.....	6
Ilustración 2 - Registros modificados	7
Ilustración 3 - Resultado de la ejecución 5.1.1	8
Ilustración 4 - Resultado de la ejecución 5.1.2.....	9
Ilustración 5 - Resultado de la ejecución 5.2	10
Ilustración 6 - Tabla de tiempos de ejecución	11

1. BASE DE DATOS SELECCIONADA Y JUSTIFICACIÓN

Se ha decidido escoger MongoDB porque los datos del dataset seleccionado encajan perfectamente con su modelo documental. MongoDB almacena la información como documentos en formato JSON, ofreciendo un esquema que no requiere de estructuras predefinidas (como en las bases de datos relacionales). Esto permite manejar datos de distinto tipo y adaptarse a cambios en su estructura sin necesidad de realizar modificaciones en la base de datos. En este caso, los datos pueden agruparse de manera lógica en documentos JSON. El dataset incluiría datos como las estadísticas de las jugadoras, equipos o características individuales de las mismas.

Cassandra, por su parte, está diseñada para casos que requieren de una alta escalabilidad y disponibilidad en la gestión de datos distribuidos. Su arquitectura descentralizada y su enfoque en particionado y replicación la hacen mejor base de datos para operaciones masivas de escritura o consultas para claves específicas. Sin embargo, este dataset no exige ese tipo de capacidad, ya que su principal necesidad es manejar datos que describen entidades con atributos detallados sobre las jugadoras, no grandes volúmenes de operaciones concurrentes. Cassandra es menos eficiente para representar relaciones jerárquicas o agrupar información relacionada dentro de una misma unidad de datos, algo que MongoDB resuelve de manera natural.

En comparación con Neo4j, que se especializa en modelar relaciones mediante grafos, MongoDB es más adecuada para este caso porque el dataset no está orientado al análisis de conexiones entre entidades. Aunque Neo4j sería una elección excelente para explorar relaciones como transferencias de jugadoras entre equipos o redes sociales (como se habló en clase), este dataset está creado para la consulta de datos descriptivos de las jugadoras, como, entre otras cosas, la nacionalidad, las estadísticas o los equipos a los que pertenecen, por lo que MongoDB, se convierte en la base de datos idónea.

Finalmente, InfluxDB, como base de datos para series temporales, está diseñada para gestionar datos dependientes del tiempo. Aunque podría ser útil en escenarios donde se analice el rendimiento de las jugadoras a lo largo del tiempo, no es el enfoque principal de este dataset. MongoDB, permite manejar este tipo de información en caso de ser necesario, pero además proporciona una solución más amplia para los datos documentales de este caso de uso.

2. DEFINICIÓN DEL ESQUEMA Y LAS SENTENCIAS DE CREACIÓN

Dado que MongoDB es una base de datos NoSQL orientada a documentos, el esquema será representado como una estructura lógica en formato JSON. Se ha optado por agrupar varias estadísticas básicas en “Stats” y otras más variadas en “OtherStats”. De la misma forma, Se crea otro sub-objeto dentro de “Stats” llamado “AerialDuels”, que muestra cuándo esa jugadora ha ganado o perdido un enfrentamiento en el aire para hacerse con el balón. Por último, “Position” se añade en formato de lista ya que una jugadora podría jugar en posiciones distintas.

```
{
  "Player": "string",
  "Nation": "string",
  "Position": ["string"],
  "Squad": "string",
  "Age": "integer",
  "Born": "integer",
  "EstimatedStartYear": "integer",
  "Stats": {
    "MatchesPlayed": "integer",
    "Starts": "integer",
    "MinutesPlayed": "integer",
    "Goals": "integer",
    "Assists": "integer",
    "TacklesWon": "integer",
    "AerialDuels": {
      "Won": "integer",
      "Lost": "integer"
    }
  },
  "OtherStats": {
    "Crosses": "integer",
    "Recoveries": "integer",
    "OwnGoals": "integer"
  }
}
```

Además, debe destacarse que “EstimatedStartYear” es un atributo calculado. Debido a la alta dificultad de encontrar un dataset que cumpla todos los requisitos del enunciado, se ha optado por realizar un pequeño cálculo aproximatorio con aleatorización que asigna un valor a dicho campo. De esta forma, se pretende cumplir con los requisitos y realizar los posteriores ejercicios de esta práctica. El script de asignación aproximatorio con aleatorización de adjunta en los anexos de este documento.

1.1 SENTENCIAS DE CREACIÓN

MongoDB no impone un esquema rígido, pero el uso de índices es una buena práctica para mejorar el rendimiento de las consultas, por lo que se ha optado en utilizarlos. Los índices permiten a MongoDB buscar documentos dentro de una colección de forma más eficiente, especialmente en campos que se consultan frecuentemente.

En este caso, se han añadido índices en los campos Player, Nation, y Squad para facilitar consultas rápidas en esos atributos.

```
from pymongo import MongoClient

client =
MongoClient('mongodb://<USERNAME>:<PASSWORD>@<SERVER>:<PORT>/')
db = client['football']
collection = db['players']

collection.create_index("Player")    # Búsqueda rápida por jugador
collection.create_index("Nation")    # Búsqueda rápida por nacionalidad
collection.create_index("Squad")     # Búsqueda rápida por equipo
```

1.2 EXPLICACIÓN Y EJECUCIÓN DE SCRIPTS

El script “main.py” adapta datos desde un archivo CSV para insertarlos en una base de datos MongoDB, estructurándolos en formato documental mediante Python. Utiliza pandas para transformar las filas del dataset en documentos JSON, organizando atributos en subdocumentos para optimizar consultas y agrupaciones. Además, conecta con MongoDB utilizando credenciales seguras desde un archivo .env (que deberá modificarse para cada caso), elimina datos redundantes en la colección existente, y crea índices en campos clave como Player, Squad y EstimatedStartYear para mejorar el rendimiento.

Los datos transformados se insertan en la base de datos, con un caso especial donde se filtra por jugadores cuyo equipo contiene "Manchester", agregándolos como registros adicionales.

Por otro lado, “transform.py” prepara el dataset original para su uso en MongoDB. Calcula un campo adicional llamado “Estimated_Start_Year”, que estima el año de inicio de la carrera de cada jugadora basándose en su edad y otros atributos. El cálculo utiliza funciones personalizadas que consideran el número de partidos como titular y el año de nacimiento de la jugadora. Se ha utilizado este método debido a la alta dificultad de encontrar un dataset que cumpla todos los requisitos exigidos por el enunciado. Por último, “modifyRegistries.py” y “queries.py” contienen las consultas y el cálculo de los tiempos requeridos por el enunciado.

1.2.1 Ejecución

1. Creación de un entorno virtual en Python e instalar las dependencias.
 - a. `$ python -m venv env`
 - b. (Windows) `$ env\Scripts\activate`
 - c. (Linux) `$ source env/bin/activate`
 - d. `$ pip install -r requirements.txt`
2. Modificar el fichero .env para configurar las 4 variables de acceso a mongo.
3. Ejecutar los ficheros en este orden:
 - a. `$ python main.py`
 - b. `$ python modifyRegisters.py`
 - c. `$ python queries.py`

3. INSERCIÓN DE 100 REGISTROS


MongoDB dispone de una función para insertar varios registros al mismo tiempo, este es el método que se utilizará para completar este apartado de la práctica.

Haciendo uso de pandas y de pymongo, se procederá a la lectura del csv y a crear un dataframe para posteriormente, enviarlo a MongoDB.

```
csv_file = './data/all_players.csv'  
data = pd.read_csv(csv_file)  
data_subset = data.head(100)  
documents = [transform_row(row) for _, row in data_subset.iterrows()]
```

El método head() se utiliza directamente para la obtención de X registros del dataframe “data”, de esta forma se evita, por ejemplo, la utilización de bucles y/o listas y gastar recursos computacionales de forma innecesaria. Por último, insert_many() envía de golpe los 100 registros escogidos mediante head() a MongoDB.

```
result = collection.insert_many(documents)
```



```
{  
  "_id": ObjectId('67443233cd6ad6beaf2b50fb'),  
  "Player": "Ivana Andrés",  
  "Nation": "es ESP",  
  "Position": Array (1)  
    0: "DF",  
  "Squad": "Real Madrid",  
  "Age": 28,  
  "Born": 1994,  
  "EstimatedStartYear": 2018,  
  "Stats": Object  
    MatchesPlayed: 7  
    Starts: 7  
    MinutesPlayed: 630  
    Goals: 0  
    Assists: 1  
    TacklesWon: 4  
    AerialDuels: Object  
  "OtherStats": Object  
    Crosses: 0  
    Recoveries: 50  
    OwnGoals: 0  
}
```

Ilustración 1 - Registro de muestra

4. MODIFICACIÓN DE DOS REGISTROS

Los dos registros se han modificado directamente en la colección players de MongoDB utilizando el script modifyRegistries.py. Este script busca cada registro por el nombre de la jugadora (Teresa Abilleira y Jessica Aby) usando un filtro exacto en el campo Player. Luego, actualiza el valor del nombre a mayúsculas empleando la operación \$set. Cada modificación se ejecuta por separado y se mide el tiempo de ejecución para evaluar el rendimiento de las actualizaciones.

```
name2 = "Jessica Aby"
start_time = time.time()
collection.update_one(
    {"Player": name2},
    {"$set": {"Player": name2.upper()}}
)
end_time = time.time()
execution_time2 = end_time - start_time
```

```
_id: ObjectId('6744385b2df3d50464ba9827')
Player: "TERESA ABILLEIRA"
Nation: "es ESP"
▶ Position: Array (1)
Squad: "Real Madrid"
Age: 22
Born: 2000
EstimatedStartYear: 2020
▶ Stats: Object
▶ OtherStats: Object
```

```
_id: ObjectId('6744385b2df3d50464ba9828')
Player: "JESSICA ABY"
Nation: "ci CIV"
▶ Position: Array (2)
Squad: "Alavés"
Age: 24
Born: 1998
EstimatedStartYear: 2021
▶ Stats: Object
▶ OtherStats: Object
```

Ilustración 2 - Registros modificados

5. CONSULTAS ESPECÍFICAS

Todas las consultas tienen un método en común mediante el cual se mide el tiempo en la que tarda cada una en ejecutarse. Es un método que recibe y ejecuta la query para posteriormente, devolver el resultado en formato JSON imprimiéndolo por pantalla.

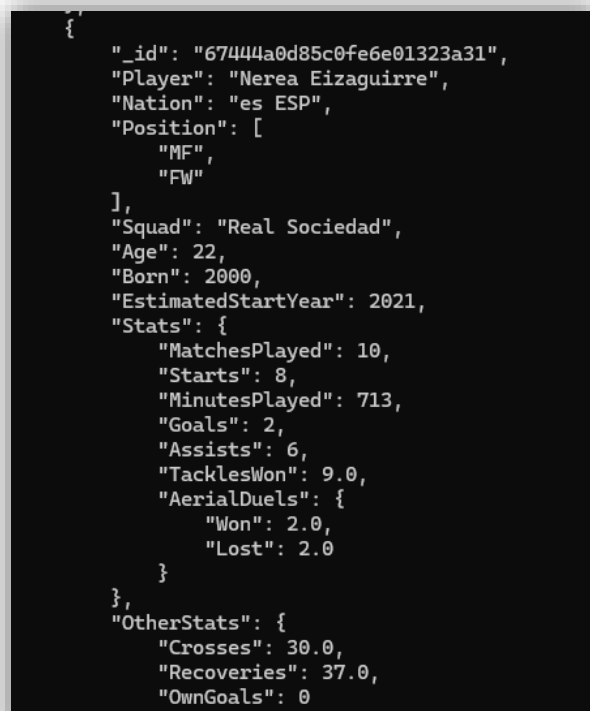
```
def measure_query_time(query):  
    start_time = time.time()  
    result = list(collection.find(query))  
    end_time = time.time()  
    execution_time = end_time - start_time  
    pretty_result = json.dumps(result, indent=4, default=str)  
    return execution_time, pretty_result
```

5.1 CONSULTA POR UNA JUGADORA ESPECÍFICA

5.1.1 Filtrando por el año de comienzo en el fútbol mayor de 2020

Haciendo uso del campo “EstimatedStartYear” y la función \$gt (greater than) de mongo, se obtendrán todos los registros cuyo campo “EstimatedStartYear” tenga un valor superior al especificado a la función \$gt.

```
query_ai = {"EstimatedStartYear": {"$gt": 2020}}  
time_ai, result_ai = measure_query_time(query_ai)
```



```
{  
  "_id": "67444a0d85c0fe6e01323a31",  
  "Player": "Nerea Eizaguirre",  
  "Nation": "es ESP",  
  "Position": [  
    "MF",  
    "FW"  
  ],  
  "Squad": "Real Sociedad",  
  "Age": 22,  
  "Born": 2000,  
  "EstimatedStartYear": 2021,  
  "Stats": {  
    "MatchesPlayed": 10,  
    "Starts": 8,  
    "MinutesPlayed": 713,  
    "Goals": 2,  
    "Assists": 6,  
    "TacklesWon": 9.0,  
    "AerialDuels": {  
      "Won": 2.0,  
      "Lost": 2.0  
    }  
  },  
  "OtherStats": {  
    "Crosses": 30.0,  
    "Recoveries": 37.0,  
    "OwnGoals": 0  
  }  
}
```

Ilustración 3 - Resultado de la ejecución 5.1.1

5.1.2 Filtrando por un equipo que empiece por “Manchester...”

Haciendo uso del campo “Squad”, expresiones regulares y la función \$options de mongo, se obtendrán todos los registros cuyo campo “Squad” empiece (\$options: ‘i’) por “Manchester” (\$regex: “^Manchester”)

```
query_a11 = {"Squad": {"$regex": "^Manchester", "$options": "i"}}
time_a11, result_a11 = measure_query_time(query_a11)
```

```
{
  "_id": "67444a0d85c0fe6e01323a37",
  "Player": "Laia Aleixandri",
  "Nation": "es ESP",
  "Position": [
    "DF",
    "MF"
  ],
  "Squad": "Manchester City",
  "Age": 22,
  "Born": 2000,
  "EstimatedStartYear": 2019,
  "Stats": {
    "MatchesPlayed": 8,
    "Starts": 8,
    "MinutesPlayed": 679,
    "Goals": 0,
    "Assists": 0,
    "TacklesWon": 0.0,
    "AerialDuels": {
      "Won": 6.0,
      "Lost": 5.0
    }
  },
  "OtherStats": {
    "Crosses": 1.0,
    "Recoveries": 65.0,
    "OwnGoals": 0
  }
}
```

Ilustración 4 - Resultado de la ejecución 5.1.2

5.2 CONSULTA POR UN PAÍS CONCRETO DONDE JUEGA UNA JUGADORA

Por último, para filtrar por país, en este caso, España (“es ESP”), solamente habrá que obtener los registros cuyo campo “Nation” sea “es ESP”.

```
query_b = {"Nation": "es ESP"}  
time_b, result_b = measure_query_time(query_b)
```

```
{  
  "_id": "67444a0d85c0fe6e01323a31",  
  "Player": "Nerea Eizaguirre",  
  "Nation": "es ESP",  
  "Position": [  
    "MF",  
    "FW"  
  ],  
  "Squad": "Real Sociedad",  
  "Age": 22,  
  "Born": 2000,  
  "EstimatedStartYear": 2021,  
  "Stats": {  
    "MatchesPlayed": 10,  
    "Starts": 8,  
    "MinutesPlayed": 713,  
    "Goals": 2,  
    "Assists": 6,  
    "TacklesWon": 9.0,  
    "AerialDuels": {  
      "Won": 2.0,  
      "Lost": 2.0  
    }  
  },  
  "OtherStats": {  
    "Crosses": 30.0,  
    "Recoveries": 37.0,  
    "OwnGoals": 0  
  }  
},
```

Ilustración 5 - Resultado de la ejecución 5.2

5.3 TABLA DE TIEMPOS DE EJECUCIÓN

Ejecución	Tiempo
Año de comienzo mayor que 2020	0.005805 (s)
El equipo comienza por Manchester	0.000690 (s)
Por un país concreto	0.000701 (s)

Ilustración 6 - Tabla de tiempos de ejecución

Para las consultas realizadas sobre mongo, se ha medido el tiempo de ejecución de cada una para evaluar el rendimiento y extraer conclusiones sobre posibles optimizaciones en el esquema o índices existentes. La consulta que filtra jugadoras por el año de inicio de su carrera muestra un tiempo más alto debido al gran volumen de resultados devueltos, lo que resalta la necesidad de mantener el índice en “EstimatedStartYear”. Las consultas relacionadas con la búsqueda por equipo y país presentan tiempos mucho más bajos, indicando que los índices existentes en Squad y Nation son bastante efectivos para cada uno de estos casos.

6. CÓDIGO

6.1 MAIN.PY

```
import os
import pandas as pd
import logging as logger
from pymongo import MongoClient
from urllib.parse import quote_plus
from dotenv import load_dotenv

# Initialize logger
log = logger.getLogger("pr02")
logger.basicConfig(filename='pr02.log', level=logger.INFO)

# Load environment variables
load_dotenv()
SERVER = os.getenv("SERVER")
PORT = os.getenv("PORT")
UNAME = quote_plus(os.getenv("USER"))
UPASS = quote_plus(os.getenv("PASSWORD"))

# Transform data grouping by stats and other stats
def transform_row(row):
    return {
        "Player": row["Player"],
        "Nation": row["Nation"],
        "Position": row["Pos"].split(",") if pd.notnull(row["Pos"])
    }
else [],
        "Squad": row["Squad"],
        "Age": int(row["Age"].split("-")[0]) if pd.notnull(row["Age"])
    }
else None,
        "Born": int(row["Born"]) if pd.notnull(row["Born"]) else None,
        "EstimatedStartYear": int(row["Estimated_Start_Year"]) if
pd.notnull(row["Estimated_Start_Year"]) else None,
        "Stats": {
            "MatchesPlayed": row["MP"] if pd.notnull(row["MP"]) else
0,
            "Starts": row["Starts"] if pd.notnull(row["Starts"]) else
0,
            "MinutesPlayed": row["Min"] if pd.notnull(row["Min"]) else
0,
            "Goals": row["Gls"] if pd.notnull(row["Gls"]) else 0,
            "Assists": row["Ast"] if pd.notnull(row["Ast"]) else 0,
            "TacklesWon": row["TklW"] if pd.notnull(row["TklW"]) else
0,
            "AerialDuels": {
                "Won": row["AerialDuels_Won"] if
pd.notnull(row["AerialDuels_Won"]) else 0,
                "Lost": row["AerialDuels_Lost"] if
pd.notnull(row["AerialDuels_Lost"]) else 0
            }
        },
        "OtherStats": {
            "Crosses": row["Crs"] if pd.notnull(row["Crs"]) else 0,
```



```

    "Recoveries": row["Recov"] if
pd.notnull(row["Recov"]) else 0,
    "OwnGoals": row["OG"] if pd.notnull(row["OG"]) else 0
}

}

# Import and load data into pandas
csv_file = './data/dataset.csv'
data = pd.read_csv(csv_file)
data_subset = data.head(100)
documents = [transform_row(row) for _, row in data_subset.iterrows()]

# Inserts the data into Mongo
try:
    client =
MongoClient(f'mongodb://{UNAME}:{UPASS}@{SERVER}:{PORT}/') # Connects
to mongo
    print(f'mongodb://{UNAME}:{UPASS}@{SERVER}:{PORT}/')
    db = client['football']
    collection = db['players']

    collection.create_index("EstimatedStartYear")
    collection.create_index("Player")
    collection.create_index("Squad")
    collection.create_index("Nation")

    collection.drop() # Drops the collection to avoid redundance
    log.info(f"Collection {collection.name} dropped")
    result = collection.insert_many(documents) # Inserts 100 registers
    log.info(f"Inserted {len(result.inserted_ids)} register(s) to
mongo")

    # Inserts a player from Manchester City
    manchester_players = data[data['Squad'].str.contains('Manchester',
na=False, case=False)]
    if not manchester_players.empty:
        first_manchester_player_row = manchester_players.iloc[0]
        first_manchester_player =
transform_row(first_manchester_player_row.to_dict())
        collection.insert_one(first_manchester_player)
        log.info(f"Inserted player {first_manchester_player['Player']}
from Manchester.")
    else:
        log.info("No players found with Squad containing
'Manchester'.")

except Exception as e:
    log.error("Could not connect to mongo: " + str(e))

```

6.2 MODIFYREGISTRIES.PY

```
import os
import time
from pymongo import MongoClient
from urllib.parse import quote_plus
from dotenv import load_dotenv
import logging as log

load_dotenv()
SERVER = os.getenv("SERVER")
PORT = os.getenv("PORT")
UNAME = quote_plus(os.getenv("USER"))
UPASS = quote_plus(os.getenv("PASSWORD"))

try:
    client =
MongoClient(f'mongodb://{UNAME}:{UPASS}@{SERVER}:{PORT}/')
    db = client['football']
    collection = db['players']

    name1 = "Teresa Abilleira"
    start_time = time.time()
    collection.update_one(
        {"Player": name1},
        {"$set": {"Player": name1.upper()}}
    )
    end_time = time.time()
    execution_time1 = end_time - start_time
    print(f"Actualización de {name1} ejecutada en
{execution_time1:.6f} segundos.")
    input("Presiona enter para ejecutar la segunda modificación")

    name2 = "Jessica Aby"
    start_time = time.time()
    collection.update_one(
        {"Player": name2},
        {"$set": {"Player": name2.upper()}}
    )
    end_time = time.time()
    execution_time2 = end_time - start_time
    print(f"Actualización de {name2} ejecutada en
{execution_time2:.6f} segundos.")

except Exception as e:
    log.error("Could not modify registries: " + str(e))
    print("Error durante las actualizaciones:", str(e))
```

6.3 QUERIES.PY

```
import time, os, json
from pymongo import MongoClient
from urllib.parse import quote_plus
from dotenv import load_dotenv

load_dotenv()
SERVER = os.getenv("SERVER")
PORT = os.getenv("PORT")
UNAME = quote_plus(os.getenv("USER"))
UPASS = quote_plus(os.getenv("PASSWORD"))

client = MongoClient(f'mongodb://{UNAME}:{UPASS}@{SERVER}:{PORT}/')
db = client['football']
collection = db['players']

def measure_query_time(query):
    start_time = time.time()
    result = list(collection.find(query))
    end_time = time.time()
    execution_time = end_time - start_time
    pretty_result = json.dumps(result, indent=4, default=str)
    return execution_time, pretty_result

# a.i Filtrar por el año de comienzo mayor a 2020
query_ai = {"EstimatedStartYear": {"$gt": 2020}}
time_ai, result_ai = measure_query_time(query_ai)
print("Resultado: \n" + str(result_ai))
print(f"Consulta a.i ejecutada en {time_ai:.6f} segundos")
input("Presiona enter para ejecutar la consulta a.ii")

# a.ii Filtrar por un equipo cuyo nombre empieza con "Manchester"
query_aii = {"Squad": {"$regex": "^Manchester", "$options": "i"}}
time_aii, result_aii = measure_query_time(query_aii)
print("Resultado: \n" + str(result_aii))
print(f"Consulta a.ii ejecutada en {time_aii:.6f} segundos")
input("Presiona enter para ejecutar la consulta a.ii")

# b Consulta por un país específico (por ejemplo, España)
query_b = {"Nation": "es ESP"}
time_b, result_b = measure_query_time(query_b)
print("Resultado: \n" + str(result_b))
print(f"Consulta b ejecutada en {time_b:.6f} segundos")
```

7. BIBLIOGRAFÍA
