Documentación de la Práctica 2 – BD NoSQL

Lander Gallastegi Muñoa

BD NoSQL utilizada

A continuación se describen varias herramientas de bases de datos NoSQL y se justifica la elección de MongoDB para esta práctica.

Cassandra

Apache Cassandra es una base de datos NoSQL orientada a columnas, diseñada para manejar grandes volúmenes de datos distribuidos en múltiples centros de datos. Su arquitectura está optimizada para ofrecer alta disponibilidad, ya que no tiene un solo punto de falla y permite la replicación automática de datos. Cassandra es especialmente eficiente en escrituras, siendo capaz de soportar altas tasas de inserción y actualización de datos. Aunque proporciona escalabilidad horizontal, su esquema es más rígido que el de otras bases de datos NoSQL, ya que requiere definir de antemano las columnas de las tablas. Esto limita la flexibilidad en comparación con MongoDB. Además, las capacidades de consulta de Cassandra son más limitadas, ya que no permite realizar búsquedas tan complejas como las que ofrece MongoDB o Neo4j, lo que puede ser una desventaja en ciertos casos de uso.

Neo4j

Neo4j es una base de datos NoSQL especializada en la gestión de grafos. Está diseñada para modelar y consultar relaciones complejas entre entidades, lo que la hace ideal para casos de uso como redes sociales, análisis de rutas y sistemas de recomendación. Utiliza un modelo de grafos que permite representar los datos como nodos, relaciones y propiedades. Su lenguaje de consulta, Cypher, facilita la ejecución de búsquedas rápidas y expresivas, optimizando el rendimiento en consultas de relaciones complejas. Aunque ofrece una solución potente para problemas específicos, Neo4j tiene una escalabilidad más limitada en comparación con MongoDB y Cassandra. Además, la modelación de datos en forma de grafos puede ser más compleja para usuarios acostumbrados a bases de datos tradicionales, lo que podría incrementar la curva de aprendizaje.

MongoDB

MongoDB es una base de datos NoSQL orientada a documentos que almacena los datos en formato BSON, una representación binaria de JSON. Se destaca por su flexibilidad en la definición de esquemas, ya que permite que los documentos dentro de una misma colección tengan estructuras diferentes. Esta capacidad facilita la incorporación de nuevos campos o la modificación

de los existentes sin necesidad de redefinir el esquema completo. MongoDB soporta consultas complejas, agregaciones avanzadas y búsqueda de texto, lo que lo convierte en una opción robusta para manejar datos semiestructurados. Además, ofrece una escalabilidad horizontal eficiente mediante fragmentación, lo que permite distribuir los datos en múltiples servidores. Su arquitectura permite manejar grandes volúmenes de datos y proporciona replicación automática para garantizar la alta disponibilidad. Sin embargo, MongoDB consume más memoria que otras bases de datos debido al formato BSON, y aunque soporta transacciones ACID, estas no son tan avanzadas como en sistemas relacionales.

Para esta práctica se ha utilizado MongoDB porque es la tecnología que más se adecúa a los requisitos. Guardar los datos en un formato JSON de forma desestructurada podría permitir agregar datos adicionales o hacer cambios en el futuro.

Además, al utilizar JSON es posible agrupar datos en sub-objetos haciendo más facil la lectura de los mismos.

Datos extraidos del dataset

Se han elegido las siguientes categorías de información: información personal, participación en partidos, rendimiento, pases y contribuciones defensivas. Las categorías elegidas contienen la información suficiente para presentar un perfil lo suficientemente detallado de una jugadora.

A continuación se listan los datos extraídos del dataset separados por categoría.

Información Personal

- Nombre de la jugadora (`Player`): Identificador esencial para distinguir una jugadora de otra.
- Nacionalidad (`Nation`): Representa el país de origen de la jugadora, relevante para hacer consultas por país.
- Posición (`Pos`): Categoriza a las jugadoras por su rol (e.g., delanteras, defensoras).
- Equipo (`Squad`): Indica el equipo al que pertenece, utilizado una consulta después.
- Edad y año de nacimiento (`Age` y `Born`): Proporciona información sobre la edad y el potencial de la jugadora.
- Año de inicio (`Starting_Year`): El año en que la jugadora inició su carrera en el fútbol.

Estadísticas de partidos

- Partidos jugados y minutos jugados (`Matches_Played` y `Minutes_Played`): Reflejan la participación total y resistencia.
- Partidos como titular (`Matches_Started`): Indica si la jugadora es titular habitual.
- Goles y asistencias (`Goals` y `Assists`): Evalúa el nivel de aportación en un partido.

Métricas de Rendimiento

- Goles esperados y asistencias esperadas (`Expected_Goals` y `Expected_Assists`): Miden la calidad de las oportunidades creadas o aprovechadas.
- Tiros y tiros a puerta (`Shots` y `Shots_on_Target`): Representan el ratio de tiros a frente a tiros a puerta.
- Regates y regates intentados (`Dribbles_Successful` y `Dribbles_Attempted`): Representan el ratio de regates exitosos y regates intentados.

Estadísticas de Pases

- Pases completados y pases intentados (`Total_Passes_Completed` y `Total_Passes_Attempted`): Ratio de pases intentados frente a pases completados
- Pases clave y pases al área de penalti (`Key_Passes` y `Passes_into_Penalty_Area`): Destacan la creatividad en la generación de juego.

Estadísticas Defensivas

- Entradas, intercepciones y bloqueos (`Tackles`, `Interceptions` y `Blocks`): Miden contribuciones defensivas.
- Recuperación de balón (`Ball_Recovery`): Refleja la capacidad para recuperar la posesión.
- Duelos aéreos ganados y perdidos (`Aerial_Duels_Won` y `Aerial_Duels_Lost`): Evalúan la eficacia en los desafíos aéreos.

Carga de datos

En este apartado se describe el proceso de carga de la carga del dataset a MongoDB utilizando Python.

Creación de colecciones

El primer paso es conectarse a la base de datos y crear la colección. Además. Elminimaremos cualquier dato anterior en la colección.

```
# Connect to MongoDB
client = pymongo.MongoClient('mongodb://root:root@localhost:27017/')
db = client['football-female']
collection = db['players']

# Drop collection
collection.drop()
```

En el código, primero nos conectamos a la base de datos con el usuario y contrasña `root`. Después, accedemos a la base de datos `football-female` y dentro a la colección `players`. Finalmente, descartamos cualquier dato que había anteriormente en la colección.

Inserción de datos

Para insertar los datos del dataset en la base de datos, es necesario cargar los datos desde el archivo CSV. Despues, se filtran los datos interesantes y se crean documentos de MongoDB.

```
def create_document(record):
    return {
        "Player": record["Player"],
        "Nation": record["Nation"],
        "Position": record["Pos"],
        "Squad": record["Squad"],
        "Age": record["Age"],
        "Born": record["Born"],
        "Starting_Year": record["Start_Year"],

        "Match_Stats": {
            "Matches_Played": record["MP"],
            "Matches_Started": record["Starts"],
            "Minutes_Played": record["Min"],
            "Goals": record["Gls"],
            "Assists": record["Ast"]
        },

        "Performance_Stats": {
            "Expected_Goals": record["xG"],
            "Expected_Assists": record["xAG"],
            "Shots": record["Sh"],
            "Shots_on_Target": record["SoT"],
            "Dribbles_Successful": record["Dribbles_Succ"],
            "Dribbles_Attempted": record["Dribbles_Att"]
        },

        "Passing_Stats": {
            "Total_Passes_Completed": record["Total_Cmp"],
            "Total_Passes_Attempted": record["Total_Att"],
            "Key_Passes": record["KP"],
            "Passes_into_Penalty_Area": record["PPA"]
        },

        "Defensive_Stats": {
            "Tackles": record["Tackles_Tkl"],
            "Interceptions": record["Int"],
            "Blocks": record["Blocks_Blocks"],
            "Ball_Recovery": record["Recov"],
            "Aerial_Duels_Won": record["AerialDuels_Won"],
            "Aerial_Duels_Lost": record["AerialDuels_Lost"]
        }
    }

# Load data
data = pd.read_csv('all_players.csv')

# Insert data
documents = data.head(100).apply(create_document, axis=1).to_list()
collection.insert_many(documents)
```

Se utiliza Pandas para cargar los datos del archivo CSV y luego se aplica la función de crear documento sobre las 100 primeras entradas. Finalmente, los documentos son insertados en la colección de MongoDB.

Modificación de datos

Uno de los requisitos de la práctica es cambiar a mayúsculas el nombre de las jugadoras. Para hacerlo, recorremos todas las jugadoras de la colección y actualizamos su nombre.

```
# Convert all player names to uppercase
@time_execution('uppercase')
def uppercase_names():
    for player in collection.find():
        collection.update_one(
            {'_id': player['_id']},
            {'$set': {'Player': player['Player'].upper()}}
        )
uppercase_names()
```

Consulta de datos

Las consultas siguientes obtienen la primera jugadora que cumpla con el requisito. Si se ha encontrado un resultado, se imprime por pantalla.

Año de comienzo mayor a 2020

```
# Get the first player with start year greater than 2020 and print to console
@time_execution('start_year')
def get_players_start_year():
    player = collection.find_one({'Starting_Year': {'$gt': 2020}})
    if player:
        print(player)
get_players_start_year()
```

Pertenece a un equipo que empieza por “Manchester”

```
# Get the first player whose team name starts with 'Manchester' and print to console
@time_execution('team_name')
def get_players_team_name():
    player = collection.find_one({'Squad': {'$regex': '^Manchester', '$options': 'i'}})
    if player:
        print(player)
get_players_team_name()
```

País de una jugadora específica

En esta última consulta, se imprime el país de la primera jugadora de la colección.

```
# Get the country of the first player
@time_execution('first_player_country')
def get_country():
    player = collection.find_one()
    if player:
        print(player['Nation'])
get_country()
```

Medición de rendimiento y tiempos de consulta

Todas las consultas realizadas en esta practica han llevado el decorador de `@@time_execution`. Al ejecutar las funciones que llevan ese decorador, se imprime en consola el tiempo de ejecución.

```
#Time execution decorator
def time_execution(name):
    def wrapper(func):
        def wrapped_func(*args, **kwargs):
            start = time.time()
            result = func(*args, **kwargs)
            end = time.time()
            print(f'Execution time ({name}): {end - start} seconds')
            return result
        return wrapped_func
    return wrapper
```

Los tiempos de ejecución de las consultas son los siguientes:

N.º	Consulta	Tiempo
1	Modificar nombres de jugadoras a mayúsculas	0,09946 s
2	Año de comienzo mayor a 2020	0,00108 s
3	Pertenece a un equipo que empieza por “Manchester”	0,00169 s
4	País de una jugadora específica	0,00087 s

Tabla 1: Tiempos de consultas

Se puede observar cómo la primera consulta toma mucho más tiempo que las otras. Esto puede deberse a varios factores. En primer lugar, en eta primera consulta recogemos todos los resultados que satisfacen la consulta, en vez de solamente el primero. Además, modificamos uno de los datos en todos los documentos que se han devuelto en la consulta.

La razón de que la tercera consulta haya tardado más que la segunda podría ser la complejidad de la misma. Es más sencillo obtener todos los valores mayores a un valor que evaluar una expresión regular.

La complejidad de la consulta también es la razón de que la última consulta haya tomado tán poco tiempo, ya que la complejidad de la misma es nula.

Código

El código se puede encontrar en https://github.com/LNDF/BDNOSQL_ATBD. Además, continuación se proporciona el código utilizado tanto para la carga de datos como para las consultas.

Carga de datos

```
import pandas as pd
import pymongo

def create_document(record):
    return {
        "Player": record["Player"],
        "Nation": record["Nation"],
        "Position": record["Pos"],
        "Squad": record["Squad"],
        "Age": record["Age"],
        "Born": record["Born"],
        "Starting_Year": record["Start_Year"],

        "Match_Stats": {
            "Matches_Played": record["MP"],
            "Matches_Started": record["Starts"],
            "Minutes_Played": record["Min"],
            "Goals": record["Gls"],
            "Assists": record["Ast"]
        },

        "Performance_Stats": {
            "Expected_Goals": record["xG"],
            "Expected_Assists": record["xAG"],
            "Shots": record["Sh"],
            "Shots_on_Target": record["SoT"],
            "Dribbles_Successful": record["Dribbles_Succ"],
            "Dribbles_Attempted": record["Dribbles_Att"]
        },

        "Passing_Stats": {
            "Total_Passes_Completed": record["Total_Cmp"],
            "Total_Passes_Attempted": record["Total_Att"],
            "Key_Passes": record["KP"],
            "Passes_into_Penalty_Area": record["PPA"]
        },

        "Defensive_Stats": {
            "Tackles": record["Tackles_Tkl"],
            "Interceptions": record["Int"],
            "Blocks": record["Blocks_Blocks"],
            "Ball_Recovery": record["Recov"],
            "Aerial_Duels_Won": record["AerialDuels_Won"],
            "Aerial_Duels_Lost": record["AerialDuels_Lost"]
        }
    }

# Load data
data = pd.read_csv('all_players.csv')

# Connect to MongoDB
client = pymongo.MongoClient('mongodb://root:root@localhost:27017/')
db = client['football-female']
collection = db['players']

# Drop collection
collection.drop()

# Insert data
documents = data.head(100).apply(create_document, axis=1).to_list()
collection.insert_many(documents)
```

Consultas

```python
import pymongo
import time

def time_execution(name):
    def wrapper(func):
        def wrapped_func(*args, **kwargs):
            start = time.time()
            result = func(*args, **kwargs)
            end = time.time()
            print(f'Execution time ({name}): {end - start} seconds')
            return result
        return wrapped_func
    return wrapper

# Connect to MongoDB
client = pymongo.MongoClient('mongodb://root:root@localhost:27017/')
db = client['football-female']
collection = db['players']

# Convert all player names to uppercase
@time_execution('uppercase')
def uppercase_names():
    for player in collection.find():
        collection.update_one(
            {'_id': player['_id']},
            {'$set': {'Player': player['Player'].upper()}}
        )
uppercase_names()

# Get the first player with start year greater than 2020 and print to console
@time_execution('start_year')
def get_players_start_year():
    player = collection.find_one({'Starting_Year': {'$gt': 2020}})
    if player:
        print(player)
get_players_start_year()

# Get the first player whose team name starts with 'Manchester' and print to console
@time_execution('team_name')
def get_players_team_name():
    player = collection.find_one({'Squad': {'$regex': '^Manchester', '$options': 'i'}})
    if player:
        print(player)
get_players_team_name()


# Get the country of the first player
@time_execution('first_player_country')
def get_country():
    player = collection.find_one()
    if player:
        print(player['Nation'])
get_country()
```