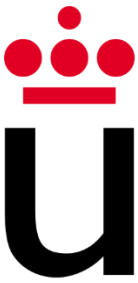


Universidad
Rey Juan Carlos

Práctica 3

Diseño e implementación de una Base de Datos en MongoDB

Gestión de Datos en Medios Digitales

Curso 2024/25

Grupo E:

Alberto Alegre Burcio | Rodrigo Dueñas Herrero | Manuel Gutiérrez Castro |

Lucas Rodríguez Bravo | Antón Rodríguez Seselle | Bernat Roselló Muñoz

Índice

Índice de ilustraciones

1. Introducción

En este documento se detalla y justifica el proceso de rediseño y adaptación a MongoDB de la base de datos creada en las anteriores prácticas, sobre el videojuego *Dark Souls*, así como las decisiones tomadas según las consultas.

Ilustración 1. Dark Souls carátula

Dark Souls se trata de una saga de videojuegos de rol y acción, desarrollados por *FromSoftware*. Estos juegos se caracterizan por su profundo *Worldbuilding*, en el que el jugador asume el papel de un no-muerto que viaja por un mundo oscuro, luchando contra una diversidad de monstruos implacables.

El jugador deberá explorar y descubrir los secretos ocultos a lo largo del juego, mientras consigue equipamiento (como armas, armaduras, hechizos y otros objetos) que lo potencia y que permita maximizar las posibilidades de sobrevivir y matar a los jefes más fuertes. Al acabar con estos conseguirá grandes cantidades de almas, que le permitirá mejorarse sus estadísticas y mejorar su equipamiento.

Además, en estos juegos se permite la conexión con otros jugadores a través de invasiones, que pueden ser de distintos tipos: invasión como aliado, o invasión como enemigo.

1.1. Glosario de términos

- **Elemento imbuido**. En Dark Souls, la mayoría de las armas pueden aumentar su nivel utilizando unos minerales, mejorando sus estadísticas de daño. Además, mediante determinados unas gemas, es posible añadirles efectos concretos (p.ej: imbuir un arma con una gema de cristal, le añadirá daño de magia. Una gema de fuego, en cambio, le añadirá daño de fuego).
- **Efecto (en los hechizos)**. Se trata de una descripción de lo que realiza este hechizo en concreto.
- **Pieza de armadura**. Una pieza de armadura es una parte de un conjunto completo de armadura. Además, el slot de armadura se refiere a qué parte del cuerpo pertenece esa pieza de armadura (p.ej: yelmo de hierro es una pieza del conjunto de hierro, que pertenece al slot de armadura de la cabeza).
- **Ítem**. Hemos denominado ítem a cualquier objeto útil de Dark Souls, que no es ni arma, armadura ni hechizo (p.ej: frasco de Estus. Un objeto que permite que el jugador se cure).
- **Almas**. Se trata de la moneda universal del juego. Sirven para subir de nivel tu personaje, comprar objetos, mejorar armas y armaduras, entre otros. Además, al morir desaparecen estas, y para recuperarlas, es necesario volver al último lugar donde el jugador perdió. En caso de no lograrlo, perderá las almas obtenidas para siempre (pues se reemplazarán por las almas que tenía en el último momento al morir de nuevo).
- **Pacto**. Un Pacto es una facción a la que el jugador puede unirse. Por una parte, formar parte de uno otorga varios beneficios y recompensas, pero, por otro lado, impone ciertas reglas al jugador. Romperlas puede generar efectos negativos. Ciertos pactos tienen efecto en las interacciones en línea del jugador.
- **Partida**. Se refiere a una partida concreta de un jugador concreto. Esta tendría todas las estadísticas del mundo y del jugador que pertenece a esa partida.
- **Invasión**. Una de las interacciones en línea que permite el juego es mediante invasiones. Existen dos tipos de invasiones en el juego: invasión como enemigo, en el que un jugador invade la partida de otro jugador para enfrentarse a él; e invasión como aliado, en el que se invade la partida para ayudarle, tanto para acabar con jefes del juego en sí, como para defenderte de invasiones enemigas. Las invasiones pueden ocurrir en contra de la voluntad del jugador (siempre que se tenga el juego configurado para ello), permitiendo ser atacado por sorpresa por otro jugador.
- **Zona**. El juego cuenta con un extenso mundo dividido en distintas zonas, en las que se encuentran distintos enemigos, jefes y objetos del juego.
- **Mancha de sangre**. En el juego, las muertes dentro del mundo se representan mediante las “Manchas de sangre”. Al interactuar con estas, un jugador puede ver los instantes previos a la muerte de otros jugadores. En nuestra base de datos, se ha decidido implementar esto como cierta información que indica el momento de la muerte en el tiempo que lleva de partida, la zona de muerte, la causa y lo que lleva equipado.

2. Identificación de las consultas

DarkSouls es un juego en el que la dificultad y la muerte de los jugadores es clave para su popularidad. Además, la muerte de los jugadores se comparte en línea entre las distintas partidas en tiempo real, mediante las "manchas de sangre" (consultar Glosario de términos). Debido a esto, es imprescindible tener mucha información de todas las muertes de los jugadores para poder reproducirlas dentro del juego.

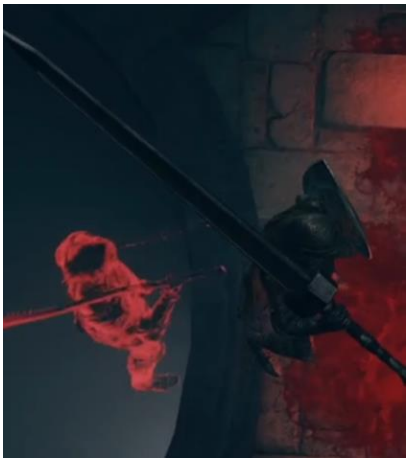


Ilustración 2. Ejemplo mancha de sangre

Es por ello por lo que, para la Base de Datos de MongoDB, se ha querido añadir más información a los jugadores, mediante las manchas de sangre.

2.1. Consultas más frecuentes

Antes de realizar los cambios necesarios en la BBDD, se ha determinado que el tipo de consultas más frecuentes para nuestro videojuego serían las que tengan que ver con los jugadores (para poder ver la información concreta de cada uno en el momento en el que se encuentran), y la información de sus muertes (que reflejan muy bien el estado de las partidas desde el principio), para poder balancear los enemigos, la cantidad de enemigos que aparecen en cada zona, las estadísticas de las armas, etc.

Consultas frecuentes con el fin de equilibrar el juego:

- Causa de muerte que se repite con mayor frecuencia, momento y zona en el que ocurre en la partida.
- Cantidad de muertes de los jugadores por cada zona.
- Mayor cantidad de almas perdidas por los jugadores.
- Jugadores con menor número de muertes y jugadores con mayor número de muertes.
- Búsqueda de las mayores y menores diferencias de tiempos entre muertes de cada jugador.
- Objetos equipables (armaduras, armas, hechizos, ítems) más utilizados (tanto por los jugadores como en las muertes de estos).
- Enemigos que salen en más zonas.
- Jugadores pertenecientes a un pacto que más invade.

2.2. Cambios en la Base de Datos

Debido a estas consultas y al añadido de las manchas de sangre, se han hecho varios cambios fundamentales en el diseño de la BBDD.

Antes de añadir la nueva entidad, se ha cambiado la manera en la que funcionan las partidas y los jugadores. En las prácticas previas, se contemplaban jugadores y partidas como dos entidades distintas una de la otra, algo que dentro del contexto del videojuego tendría sentido. Sin embargo, a efectos prácticos, no había realmente diferencia entre los jugadores y las partidas, y, debido al diseño de las consultas más frecuentes, se ha tomado la decisión de unificar las tres entidades principales ("jugadores", "partidas" e "invasiones") en una sola: "jugadores". De esta manera, la BBDD en MongoDB no tiene que acceder a distintos documentos para poder acceder a las zonas, las invasiones, ni tampoco las manchas de sangre, ya que al principio se contempló la idea de que "Mancha de sangre formase parte de Partidas".

Aparte de esta modificación, se añadieron las Manchas de Sangre ya mencionadas previamente. La entidad contiene la siguiente información:

- 'tiempo_muerte': instante en el que murió el jugador dentro de su partida.
- 'causa_muerte': pequeña descripción con la causa de muerte del jugador.
- 'zona_muerte': identificador de la zona en la que murió el jugador.
- 'almas_al_morir': número de almas que tenía el jugador al morir.
- 'piezas_armadura_al_morir': contiene los identificadores del casco, torso, pernera y guanteletes en el instante de la muerte.
- 'arma_al_morir': identificador del arma que llevaba el jugador en ese momento.

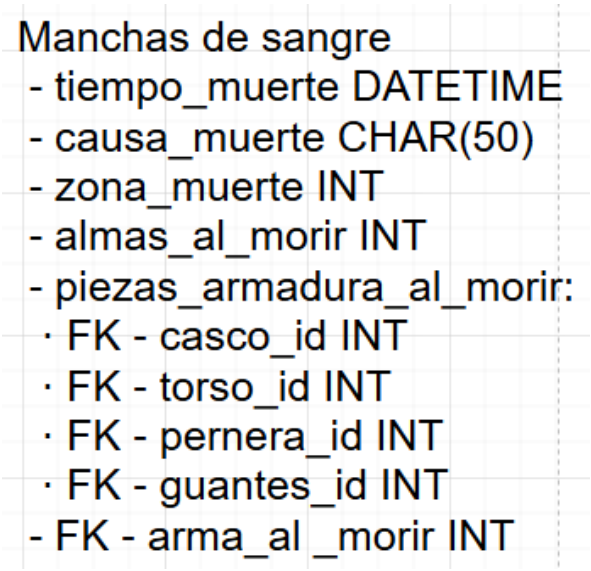


Ilustración 3. Entidad Mancha de Sangre

Por tanto, el esquema definitivo de la BBDD completa es el siguiente:

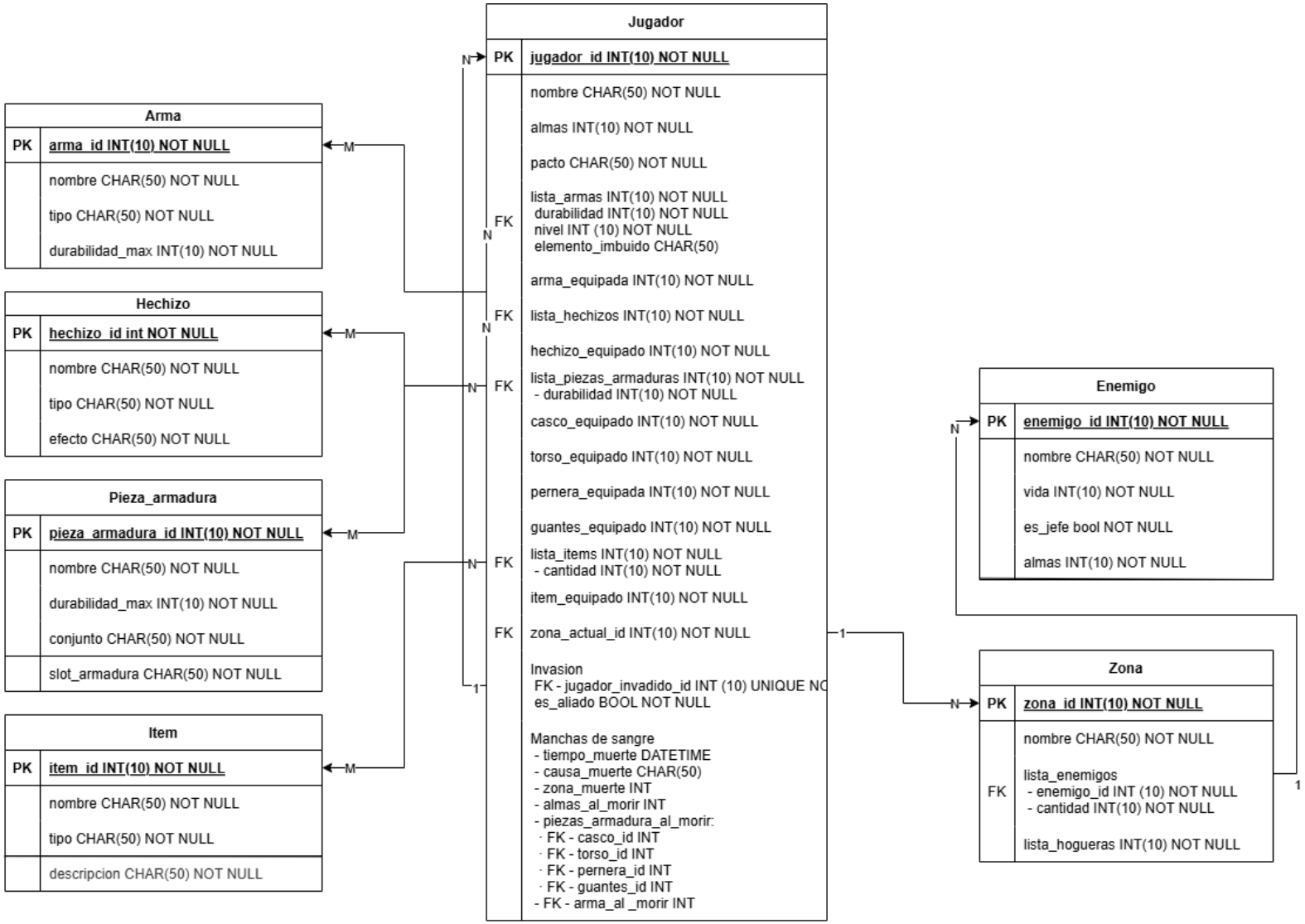


Ilustración 4. Diseño completo de la BBDD modificada

3. Creación de Documentos

3.1. Modificación de “jugador-partida.xml” a “jugadores.xml”

Para poder crear los documentos de la BBDD, primero ha sido necesario adaptar el fichero “jugador-partida.xml” al nuevo diseño. Para ello, se han modificado los scripts “generate_jugador_partida.py” y “jugador_partida_to_xml.py”, que ahora han pasado a llamarse “generate_jugador.py” y “jugador_to_xml.py”, respectivamente. Se ha seguido haciendo uso de la librería ElementTree para la creación, estructuración y acceso a los ficheros ‘.xml’, y, además se ha incorporado la librería ‘numpy’, para poder obtener elementos aleatorios aplicando distintos pesos.

Las modificaciones principales en el script de generación de datos son:

- Unificación de los elementos de partida e invasión dentro de jugador: el elemento zona_actual_id de partida ha pasado a estar en jugador, e invasión ha pasado a ser un elemento interno también de jugador. Las invasiones pueden o ser null, o contener el id del jugador invadido, y si la invasión se realiza como aliado o no.
- Para la generación de invasión, no se puede crear directamente junto al jugador, sino que ha sido necesario crear una función ‘choose_player’ que se llama posteriormente a la creación de todos los jugadores.
- Se ha creado una función ‘generate_bloodstain’ que es la encargada de generar la información de una mancha de sangre. La función ‘generate_player’ hará varias llamadas a esta función para obtener las manchas de sangre y guardarlas en un array, que será añadido a la información de la partida.
- En la función ‘generate_bloodstain’ se crea información de manera aleatoria, de modo que se obtienen los ids del equipamiento del jugador, el número de almas, la zona, el tiempo de muerte y la causa de muerte.
- Para el tiempo de muerte y la causa de muerte ha sido necesario crear algunas funciones extra:
    - ‘generate_death_reason’ elige unos tipos de muertes más generales aleatoriamente y con distintos pesos (muerte por un jefe: 60%, muerte por un enemigo: 30%, muerte por otra causa: 10%), y según el tipo de muerte, se elige aleatoriamente un jefe, un enemigo o una causa de muerte distinta dentro de las establecidas.
    - En el caso de los tiempos de muerte se han creado las funciones ‘generate_deathtime’, para generar un tiempo aleatorio (entre 0h, 0min, 0 sec y 0 milisec, y 0 h, 30 m, 59, sec y 999 milisec. El máximo establecido se ha puesto de manera que el tiempo de la última muerte generada no sea exageradamente alto y así tenga algo más de coherencia con una partida real), y ‘add_time’, que suma dos tiempos y devuelve el resultado.

```python
pIds_chosen = []
def choose_player(player, players):
    player_id = random.randint(0, len(players)-1)
    #players_ids = {int(p["jugador_id"]) for p in players}
    while player_id in pIds_chosen or player_id == int(player["jugador_id"]):
        player_id = random.randint(0, len(players)-1)
        continue
    player["invasion"] = {
        "jugador_invadido_id": str(player_id),
        "es_aliado": random.choice(["True", "False"])
    }
    pIds_chosen.append(player_id)
    return player
```

Ilustración 5. Función 'choose_player'

```python
manchas_sangre = []

    last_time = [0,0,0,0]
    for i in range(random.randint(50,300)):
        generated_bloodstain, last_time = generate_bloodstain(i, last_time)
        manchas_sangre.append(generated_bloodstain)

    player = {
        "jugador_id": str(player_id),
        "nombre": f"{prefix}{fake.first_name()}{suffix}",
        "almas": str(random.randint(0, 999999999)),
        "pacto": random.choice(["Way of White", "Princess's Guard", "Blade of the Darkmoon",
                                "Warrior of Sunlight", "Forest Hunter", "Chaos Servant",
                                "Gravelord Servant", "Path of the Dragon", "Darkwraith"]),
        "lista_armas": list_weapons,
        "arma_equipada": f"{equipped_weapon["arma_id"]}",
        "lista_hechizos": list_magic_elems,
        "hechizo_equipado": f"{equipped_magic["hechizo_id"]}",
        "lista_piezas_armaduras": list_armor_elems,
        "casco_equipado": equipped_head,
        "torso_equipado": equipped_torso,
        "pernera_equipada": equipped_legs,
        "guantes_equipado": equipped_hands,
        "lista_items": list_consumables,
        "item_equipado": f"{equipped_consumable["item_id"]}",
        "zona_actual_id": str(random.randint(0, 30)),
        "invasion": None,
        "manchas_sangre" : manchas_sangre
    }

    return player
```

Ilustración 6. Cambios dentro de la función 'generate_player'

```python
tree = ET.parse("zonas.xml")
root = tree.getroot()
list_of_zones = root.findall(".//zona")
tree = ET.parse("objetos.xml")
root = tree.getroot()
list_of_weapons = root.findall(".//arma")
list_of_helmet = root.findall(".//pieza_armadura[slot_armadura='Cabeza']")
list_of_torso = root.findall(".//pieza_armadura[slot_armadura='Torso']")
list_of_legs = root.findall(".//pieza_armadura[slot_armadura='Piernas']")
list_of_hands = root.findall(".//pieza_armadura[slot_armadura='Manos']")

def generate_bloodstain(idx, last_time):
    random_weapon = random.choice(list_of_weapons)
    random_helmet = random.choice(list_of_helmet)
    random_torso = random.choice(list_of_torso)
    random_legs = random.choice(list_of_legs)
    random_hands = random.choice(list_of_hands)
    random_zone = random.choice(list_of_zones)

    arma_id = random_weapon.get("arma_id")
    casco_id = random_helmet.get("pieza_armadura_id")
    torso_id = random_torso.get("pieza_armadura_id")
    piernas_id = random_legs.get("pieza_armadura_id")
    guantes_id = random_hands.get("pieza_armadura_id")
    zona_id = random_zone.get("zona_id")

    death_reason = generate_death_reason()

    new_time = add_time(last_time, generate_deathtime())

    bloodstain = {
        "tiempo_muerte": format_time(new_time),
        "causa_muerte": death_reason,
        "zona_muerte": zona_id,
        "almas_al_morir": str(random.randint(0, 999999)),
        "piezas_armaduras_al_morir": {
            "casco_id": casco_id,
            "torso_id": torso_id,
            "piernas_id": piernas_id,
            "guantes_id": guantes_id
        },
        "arma_al_morir": arma_id
    }
    return bloodstain, new_time
```

Ilustración 7. 'generate_bloodstain'

```python
tree = ET.parse("enemigos.xml")
root = tree.getroot()
list_of_enemies = root.findall(".//enemigo[es_jefe='0']")
list_of_bosses = root.findall(".//enemigo[es_jefe='1']")

def generate_death_reason():
    other_death_reasons = ["caída", "caída de altura", "trampa de flechas", "lanzas desde el suelo",
                           "lanzas desde el techo", "glamur", "trampilla o suelo falso", "NPC hostil",
                           "invasión", "fuego", "veneno", "sangrado", "magia", "maldición", "ahogamiento"]

    list_of_choices = ["boss", "enemy", "other"]
    choice = numpy.random.choice(list_of_choices, p=[0.6, 0.3, 0.1], size=1)[0]

    death_reason = "Muerte por "

    if choice == "boss":
        random_boss = random.choice(list_of_bosses)
        boss_name = random_boss.find("nombre").text
        death_reason += f"jefe: {boss_name}"
    elif choice == "enemy":
        random_enemy = random.choice(list_of_enemies)
        enemy_name = random_enemy.find("nombre").text
        death_reason += f"enemigo: {enemy_name}"
    else:
        death_reason += random.choice(other_death_reasons)

    return death_reason
```

Ilustración 8.'generate_death_reason'

```python
def generate_deathtime():
    # horas , minutos, segundos, milisegundos
    return [0, random.randint(0,30), random.randint(0,59), random.randint(0,999)]

def add_time(last_time, add_time):
    # Sumar el tiempo de la muerte a la hora actual
    new_time = [0, 0, 0, 0]
    new_time[3] = last_time[3] + add_time[3]
    if new_time[3] >= 1000:
        new_time[2] += new_time[3] // 1000
        new_time[3] %= 1000
    new_time[2] += last_time[2] + add_time[2]
    if new_time[2] >= 60:
        new_time[1] += new_time[2] // 60
        new_time[2] %= 60
    new_time[1] += last_time[1] + add_time[1]
    if new_time[1] >= 60:
        new_time[0] += new_time[1] // 60
        new_time[1] %= 60
    new_time[0] += last_time[0] + add_time[0]
    return new_time
```

Ilustración 9. 'generate_deathtime' y 'add_time'

Las modificaciones en el script de conversión de datos a xml han sido las pertinentes para la creación de las manchas de sangre, y para unificar los elementos de partidas e invasiones en jugadores:

```python
# Zona actual
        ET.SubElement(jugador_elem,"zona_actual_id").text = jugador["zona_actual_id"]

        #Invasion
        invasion_elem = ET.SubElement(jugador_elem, "invasion")
        if jugador["invasion"] != None:
            ET.SubElement(invasion_elem, "jugador_invadido_id").text = jugador["invasion"]["jugador_invadido_id"]
            ET.SubElement(invasion_elem, "es_aliado").text = jugador["invasion"]["es_aliado"]

        # Manchas de sangre
        manchas_sangre_elem = ET.SubElement(jugador_elem,"manchas_sangre")
        for mancha in jugador["manchas_sangre"]:
            mancha_elem = ET.SubElement(manchas_sangre_elem, "mancha_sangre")
            ET.SubElement(mancha_elem,"tiempo_muerte").text = mancha["tiempo_muerte"]
            ET.SubElement(mancha_elem,"causa_muerte").text = mancha["causa_muerte"]
            ET.SubElement(mancha_elem,"zona_muerte").text = mancha["zona_muerte"]
            ET.SubElement(mancha_elem,"almas_al_morir").text = mancha["almas_al_morir"]
            piezas_armaduras_al_morir_elem = ET.SubElement(mancha_elem,"piezas_armaduras_al_morir")
            ET.SubElement(piezas_armaduras_al_morir_elem,"casco_id").text = mancha["piezas_armaduras_al_morir"]["casco_id"]
            ET.SubElement(piezas_armaduras_al_morir_elem,"torso_id").text = mancha["piezas_armaduras_al_morir"]["torso_id"]
            ET.SubElement(piezas_armaduras_al_morir_elem,"piernas_id").text = mancha["piezas_armaduras_al_morir"]["piernas_id"]
            ET.SubElement(piezas_armaduras_al_morir_elem,"guantes_id").text = mancha["piezas_armaduras_al_morir"]["guantes_id"]
            ET.SubElement(mancha_elem,"arma_al_morir").text = mancha["arma_al_morir"]
```

Ilustración 10. Modificación de 'jugador _to_xml' para crear las manchas de sangre

Además, se hicieron las modificaciones necesarias en el esquema, que pasó a llamarse ‘jugadores.xsd’, para que el xml fuese validado correctamente.

```xml
<!-- ZONA ACTUAL -->
              <xs:element type="xs:string" name="zona_actual_id"/>
              <!-- INVASION -->
              <xs:element name="invasion" minOccurs="0">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element type="xs:string" name="jugador_invadido_id" minOccurs="0"/>
                    <xs:element type="xs:string" name="es_aliado" minOccurs="0"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <!-- MANCHAS DE SANGRE -->
              <xs:element name="manchas_sangre">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="mancha_sangre" maxOccurs="unbounded" minOccurs="0">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element type="xs:string" name="tiempo_muerte"/>
                          <xs:element type="xs:string" name="causa_muerte"/>
                          <xs:element type="xs:int" name="zona_muerte"/>
                          <xs:element type="xs:int" name="almas_al_morir"/>
                          <xs:element name="piezas_armaduras_al_morir">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:element type="xs:string" name="casco_id"/>
                                <xs:element type="xs:string" name="torso_id"/>
                                <xs:element type="xs:string" name="piernas_id"/>
                                <xs:element type="xs:string" name="guantes_id"/>
                              </xs:sequence>
                            </xs:complexType>
                          </xs:element>
                          <xs:element type="xs:string" name="arma_al_morir"/>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute type="xs:int" name="jugador_id"/>
```

Ilustración 11. Manchas de sangre añadidas al esquema de 'jugadores'

3.2. Conversión de XML a JSON

Para convertir los archivos XML a documentos JSON, se ha hecho de dos maneras. Para 'enemigos' y 'objetos' se ha utilizado la herramienta online JSON formatter: https://jsonformatter.org/xml-to-json. En el caso de 'jugadores', los cambios hicieron que aumentara de manera exponencial su tamaño, pasando de 1.31 MB a 54.7 MB, haciendo imposible el procesamiento del archivo por parte de la página. Es por ello por lo que se creó un nuevo script 'xml_to_json.py' que convierte un archivo XML a formato JSON, utilizando la librería xmltodict. Además, al crearse 'zonas', el formato que se creaba para los arrays no era el adecuado, así que también se optó por realizar el script para que fuese compatible con este fichero.

```python
def convert_xml_to_json(xml_file_path, json_file_path):
    with open(xml_file_path, 'r', encoding='utf-8') as xml_file:
        xml_data = xml_file.read()
        data_dict = xmltodict.parse(xml_data)
        cleaned_data = reorder_attributes(data_dict)

    # 🛠️ Quitar la raíz si es {"jugadores": {"jugador": [...]}}
    if "jugadores" in cleaned_data and "jugador" in cleaned_data["jugadores"]:
        cleaned_data = cleaned_data["jugadores"]["jugador"]
    elif "zonas" in cleaned_data and "zona" in cleaned_data["zonas"]:
        cleaned_data = cleaned_data["zonas"]["zona"]

    rename_enemy_text_fields(cleaned_data)

    with open(json_file_path, 'w', encoding='utf-8') as json_file:
        json.dump(cleaned_data, json_file, indent=4, ensure_ascii=False)

    print(f"✅ Archivo JSON guardado en: {json_file_path}")


# Uso: python xml_to_json.py entrada.xml salida.json
if __name__ == "__main__":
    file_name = input()
    convert_xml_to_json(f"{file_name}.xml", f"{file_name}.json")
```

Ilustración 12. Parte del código de 'xml_to_json'

Finalmente, dentro de MongoDB se crearon 4 colecciones: enemigos, jugadores, objetos y zonas, en las que se importaron los archivos JSON creados para crear los distintos documentos.

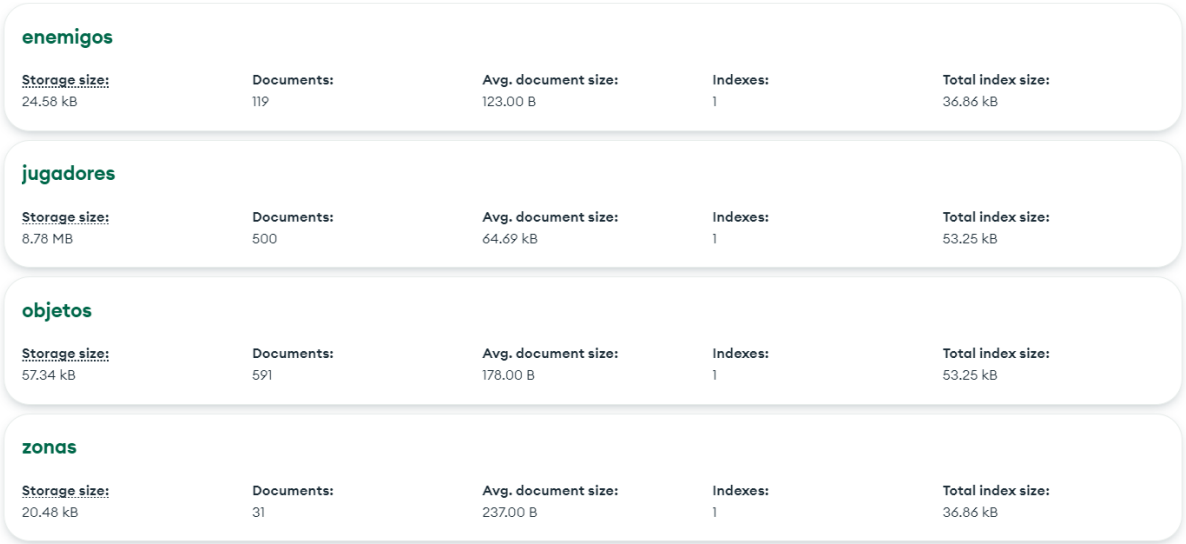


Ilustración 13. Base de Datos en MongoDB

4. Diseño de operaciones CRUD

Para realizar las consultas, se han tenido que adaptar algunas consultas de la base de datos con respecto a la primera práctica, debido a que no ha sido posible replicar algunas de estas consultas.

Además, para la medición de tiempos de las consultas que utilizan Aggregation Framework, ha sido necesario realizarlo utilizando los siguientes comandos en la Shell:

```
1  const start = new Date();
2  const result = db.jugadores.aggregate([...]).toArray();
3  const end = new Date();
4  print(`Execution time: ${end - start} ms`);
5  printjson(result);
```

Ilustración 14. Medición de tiempos utilizando el Aggregation Framework

A pesar de ser esta última una opción no determinista, nos permite comparar en términos generales la eficiencia de una misma consulta, pero aplicando la creación de índices.

Además, es importante mencionar que algunas de las consultas que se realizaron en la primera práctica, no contenían varias condiciones, es por ello por lo que se ha hecho una selección de algunas de las consultas originales, pero añadiendo alguna condición extra.

4.1. Consultas de filtrado

4.1.1. Por una sola condición

4.1.1.1. *Consulta 1*

Enemigos que son jefes.

```
22  //- Enemigos que son jefes - Bernat
23  db.enemigos.find({ "es_jefe": "1" })
24  // indices añadidos para acelerar el query:
25  db.enemigos.createIndex({ "es_jefe": -1 })
```

'es_jefe' es un campo de los enemigos que codifica si es jefe (1) o si no lo es (0), aunque en un futuro podría expandirse para codificar más categorías de enemigos de forma retro compatible. Se ha probado a añadir un índice para mejorar la búsqueda, ya que, a pesar de tratarse de una búsqueda rápida y sencilla, nos evita tocar todos los documentos de enemigos innecesariamente lo cual mejora la escalabilidad y rendimiento de la BBDD.

RESULTADO	SIN ÍNDICES	CON ÍNDICE

```
db.enemigos.find({ "es_jefe": "1" })
< {
  _id: ObjectId('681111df77b13d1af1b79408'),
  nombre: 'Demonio del asilo',
  vida: '825',
  es_jefe: '1',
  almas: '2000',
  _enemigo_id: 'enemigo93'
}
{
  _id: ObjectId('681111df77b13d1af1b79409'),
  nombre: 'Gárgola campana',
  vida: '1000',
  es_jefe: '1',
  almas: '10000',
  _enemigo_id: 'enemigo94'
}
{
  _id: ObjectId('681111df77b13d1af1b7940a'),
  nombre: 'Demonio de Aries Menor',
  vida: '1176',
  es_jefe: '1',
  almas: '6000',
  _enemigo_id: 'enemigo95'
}
```

Ilustración 15. Resultado de la búsqueda de enemigos que son jefes.

```
executionStats: {
  executionSuccess: true,
  nReturned: 26,
  executionTimeMillis: 0,
  totalKeysExamined: 0,
  totalDocsExamined: 119,
  executionStages: {
    isCached: false,
    stage: 'COLLSCAN',
    filter: {
      es_jefe: {
        '$eq': '1'
      }
    },
```

Ilustración 16. Rendimiento de la consulta de enemigos que son jefes sin índices.

```
executionStats: {
  executionSuccess: true,
  nReturned: 26,
  executionTimeMillis: 0,
  totalKeysExamined: 26,
  totalDocsExamined: 26,
```

Ilustración 17. Rendimiento de la consulta de enemigos que son jefes con índice: es_jefe_-1 .

* Observe la estadística de ejecución '*totalDocsExamined*'

4.1.1.2. *Consulta 2*

Jugadores que tengan equipada una magia de tipo milagro.

```
db.jugadores.aggregate([
    {
        $lookup: {
            from: "objetos",
            localField: "hechizo_equipado",
            foreignField: "_hechizo_id",
            as: "hechizo_info"
        }
    },
    { $unwind: "$hechizo_info" },
    {
        $match: {
            "hechizo_info.tipo": { $regex: /miracle/i }
        }
    },
    {
        $project: {
            _id: 0,
            nombre: 1,
            hechizo_equipado: 1,
            tipo_hechizo: "$hechizo_info.tipo",
            nombre_hechizo: "$hechizo_info.nombre"
        }
    }
])
```

Resultado:

SIN ÍNDICES

```
Execution time (VERSION 1): 314 ms
[
  {
    nombre: 'Night_Robert_Lord',
    hechizo_equipado: 'hechizo63',
    tipo_hechizo: 'Miracle - Support',
    nombre_hechizo: 'Tranquil Walk of Peace'
  }
```

Ilustración 18. Consulta 1 sin índices

CON ÍNDICES

Índice: tipo

```
Execution time (VERSION 1): 36 ms
```

Ilustración 19. Consulta 1 con índice tipo

4.1.2. Por varias condiciones

4.1.2.1. *Consulta 1*

Mostrar los jefes cuyas almas que dan son mayores a 50000.

```
1  db.enemigos.find(
2      { $expr: {$and: [ {es_jefe:"1"},
3      {$gt: [ {$toInt: "$almas"}, 50000]}]}},
4      { _id: 0, nombre: 1, almas: 1 })
```

Ilustración 20. Mostrar los jefes cuyas almas que dan son mayores a 50000.

Resultado de la consulta:

Ilustración 21. Resultado de fsulta 1

SIN ÍNDICES

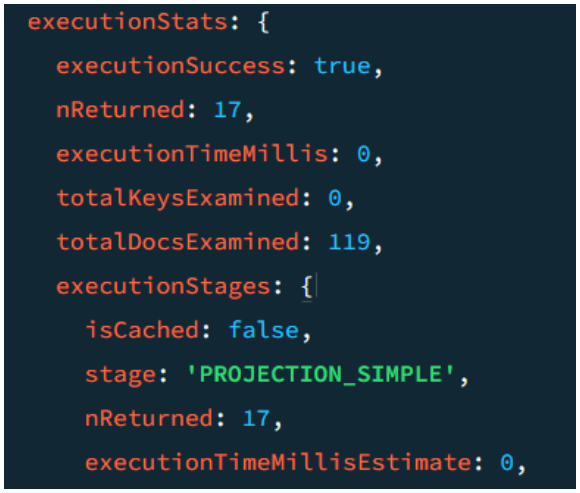


Ilustración 22. Consulta 1 sin índices

CON ÍNDICES

Índice: es_jefe

Ilustración 23. Consulta 1 con índice es_jefe

Índice: es_jefe y almas

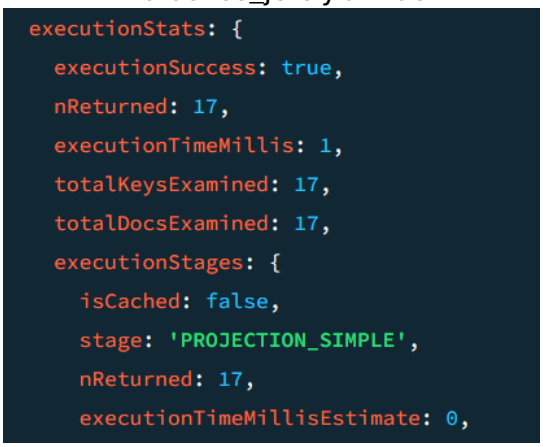


Ilustración 24. Consulta 1 con índice es_jefe y almas

Si bien parece que tarda 1 ms más que en la versión sin índice, se examinan solo 17 documentos, en contraposición a los 119 originales.

4.1.2.2. *Consulta 2*

Jugadores cuyo nombre contenga 'x' (mayúsculas o minúsculas), que tienen equipado un arma de fuego y están invadiendo a alguien.

```
db.jugadores.find({
$and:[
    {"nombre": { $regex : /[x]/, $options:"i"} },
    {"lista_armas.elemento":"Fire"},
    {"invasion": {$ne:null}}
  ]},
{nombre:1, lista_armas: { $elemMatch: { elemento: "Fire" } }, invasion: 1})
```

Ilustración 25. Jugadores cuyo nombre contenga 'x' (mayus o minus), que tienen equipado un arma de fuego y están invadiendo a alguien.

Resultado de la consulta:

SIN ÍNDICES

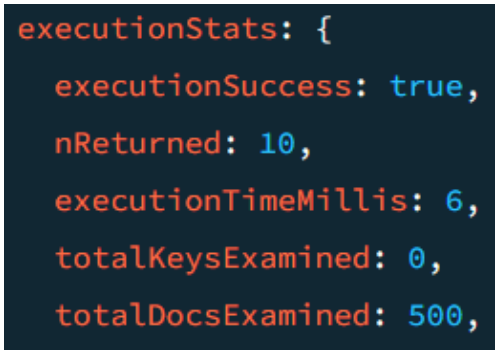
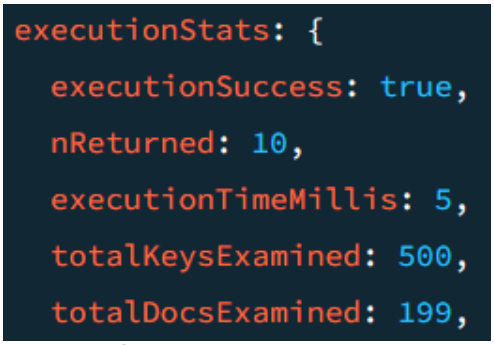


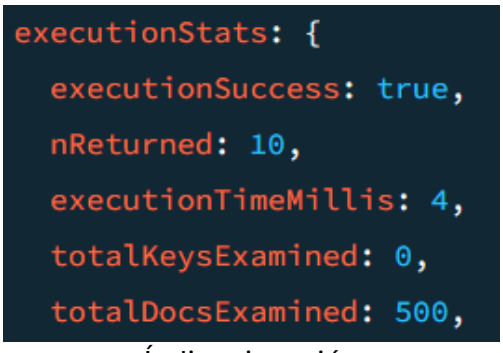
Ilustración 26. Consulta 2 sin índices

CON ÍNDICES

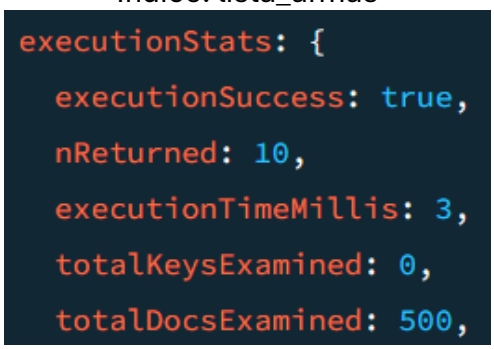
Índice: nombre



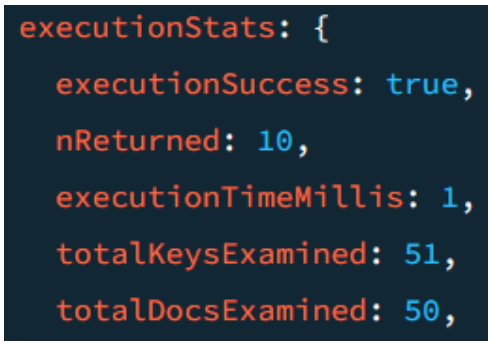
Índice: lista_armas.element



Índice: lista_armas



Índice: invasión



Índice: nombre, lista_armas e invasión simultáneamente

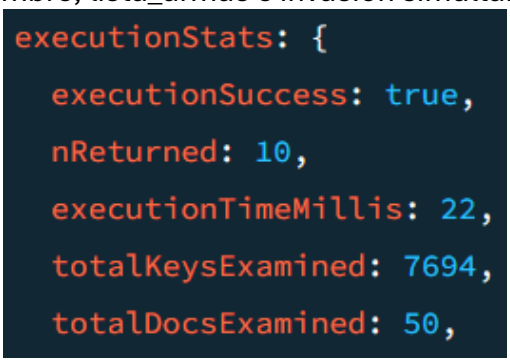


Ilustración 27. Consulta 2 con índices

Con la experimentación realizada en el caso de esta consulta, podemos concluir que el uso de un índice de invasiones nos da el resultado óptimo. Esto es gracias a que el índice que crea permite que la consulta acceda directamente a los jugadores invasores, podando gran parte de los documentos que necesita examinar. Como se puede ver, en el caso de un índice por nombres, la mejora no es muy grande ya que igualmente tiene que examinar 500 índices, que es el mismo número de jugadores. En el caso de lista_armas y lista_armas.element, no examina índices por lo que tiene que consultar todos los documentos igualmente. Por último, crear un índice con los tres elementos, hace que se creen una cantidad exageradamente alta de índices, haciendo que examinarlos tarde entre 4 y 5 veces más del tiempo original.

4.1.2.3. *Consulta 3*

Zonas con 3 o más tipos de enemigos diferentes y menos de 3 hogueras.

```
db.zonas.find(
    {
        $and: [
            { lista_enemigos: { $type: "array" } },
            { lista_hogueras: { $type: "array" } },
            { $expr: { $gt: [ {$size: "$lista_enemigos"}, 2 ] } },
            { $expr: { $lt: [ {$size: "$lista_hogueras"}, 3 ] } }
        ]
    },
    {
        nombre: 1
    }
)
```

Ilustración 28. Zonas con 3 o más tipos de enemigos diferentes y menos de 3 hogueras.

SIN ÍNDICES

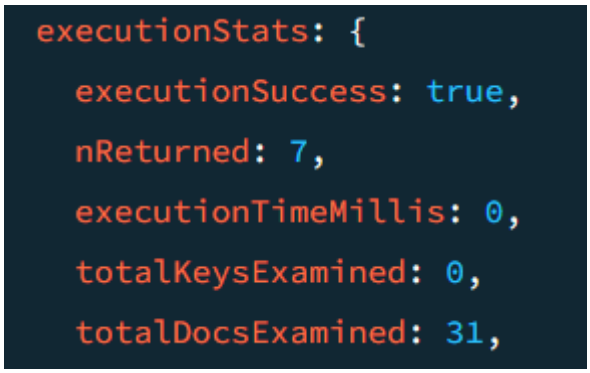


Ilustración 29. Consulta 3 sin índices

CON ÍNDICES	
Índice: num_hogueras	Índice: num_enemigos y num_hogueras

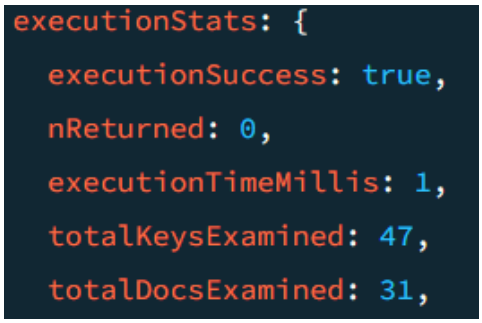


Ilustración 30. Consulta 3 con índice num_hogueras

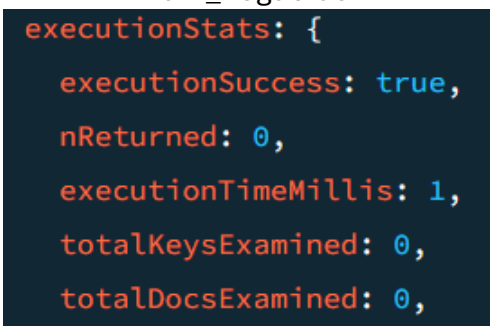


Ilustración 31. Consulta 3 con índice num_enemigos y num_hogueras

Se tienen que leer los 31 documentos de zonas y aplicar el filtro en cada uno. Como son pocos, el coste es de 0ms. Con el índice de hogueras se recorren las 47 entradas del índice con num_hogueras < 3 pero al no incluír num_enemigos, se acaban recuperando los 31 documentos que se escanearían aun no teniendo ningún índice.

Al crear un índice compuesto de enemigos y hogueras se pueden resolver ambos filtros en el árbol por lo que no se examinan ni claves ni documentos y se optimiza al máximo la consulta.

4.1.2.4. *Consulta 4*

Enemigos que dan muchas almas (1000) y tienen poca vida (300)

```
db.enemigos.find(
    {
        $expr: {
            $and: [
                { $gte: [{ $toInt: "$almas" }, 1000] },
                { $lte: [{ $toInt: "$vida" }, 300] }
            ]
        }
    },
    {
        _id: 0,
        nombre: 1,
        almas: 1,
        vida: 1
    }
)
```

Resultados

SIN ÍNDICES

Ilustración 32. Consulta 4 sin índices

CON ÍNDICES

Índice: almas

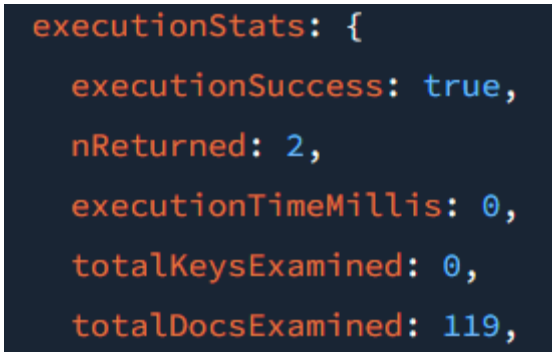


Ilustración 33. Consulta 3 con índice almas

Índice: almas y vida

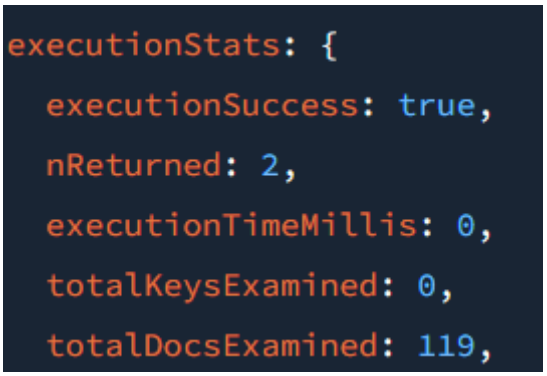


Ilustración 34. Consulta 3 con índice almas y vida

4.2. Consultas de ordenación

4.2.1.1. *Consulta de ordenación 1*

Las 5 armas con mayor durabilidad ordenadas de mayor a menor, mostrar nombre, tipo y durabilidad máxima

```
db.objetos.aggregate([
    {
        $match: {
            _arma_id: { $regex: /.*/i }
        }
    },
    {
        $addFields: {
            durabilidad_int: { $toInt: "$durabilidad_max" }
        }
    },
    {
        $sort: { durabilidad_int: -1 }
    },
    {
        $limit: 5
    },
    {
        $project: {
            _id: 0,
            nombre: 1,
            tipo: 1,
            durabilidad_max: 1
        }
    }
])
```

Sin índices

```
Execution time (VERSION 1): 5 ms
{
  nombre: 'Dark Hand',
  tipo: 'Fist Weapon',
  durabilidad_max: '999'
}
```

Ilustración 35. Estadísticas pre-índice consulta ordenacion 1

Con índices

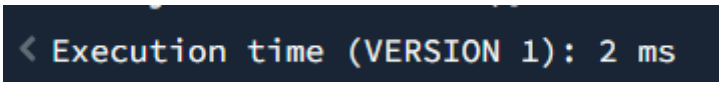


Ilustración 36. Estadísticas con indice en durabilidad_max

4.2.1.2. *Consulta de ordenación 2*

Los jugadores que más muertes tienen ordenados de mayor a menor

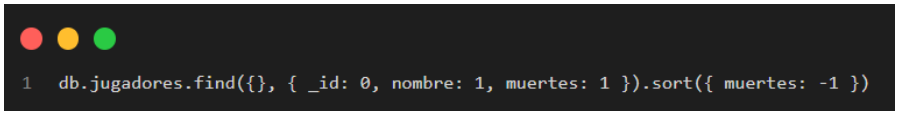


Ilustración 37. Los jugadores que más muertes tienen ordenados de mayor a menor

Resultado de la consulta:

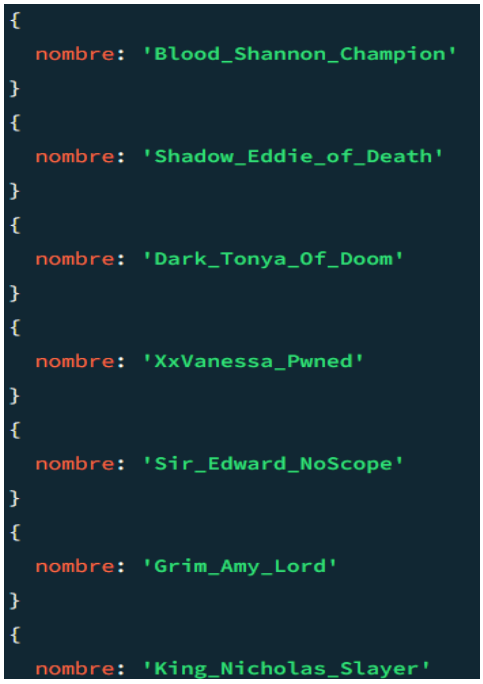


Ilustración 38. Resultado de consulta 3 de ordenación

SIN ÍNDICES

Ilustración 39. Consulta 3 de ordenación sin índices

CON ÍNDICES

Índice: muerte

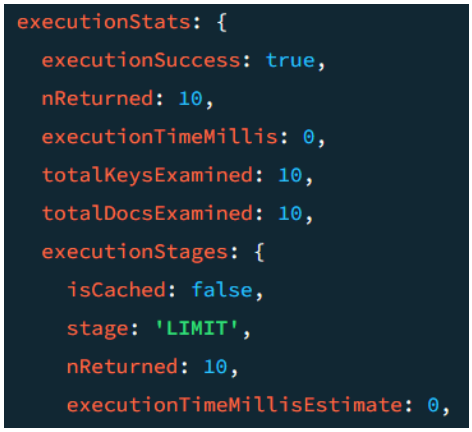


Ilustración 40. Consulta 3 de ordenación con índice de muerte

Índice: muerte y nombre

Ilustración 41. Consulta 3 de ordenación con índice de muerte y nombre

Sin índices se tienen que leer los 500 documentos, ordenarlos y extraer los 10 primeros. Con índice sobre muertes, se crea el B-tree descendente y se toman las 10 primeras claves. Se leen 10 claves y 10 documentos en lugar de 500 y se tardan 0 ms en comparación con las 5 iniciales. Creando un índice compuesto de muertes y nombre, se tarda 1 ms y se examinan 0 documentos pero 500 claves. No supone una mejora con respecto a usar el índice de muerte porque ocupa más espacio y no es necesario ya que se está ordenando simplemente por muerte.

4.2.1.3. *Consulta de ordenación 3*

El nombre de cada jugador, sus almas y el nombre del pacto al que pertenecen, ordenado por el nombre del jugador alfabéticamente

```
db.jugadores.find({}, {_id: 0, nombre: 1, almas: 1, pacto: 1}).sort({nombre: 1})
```

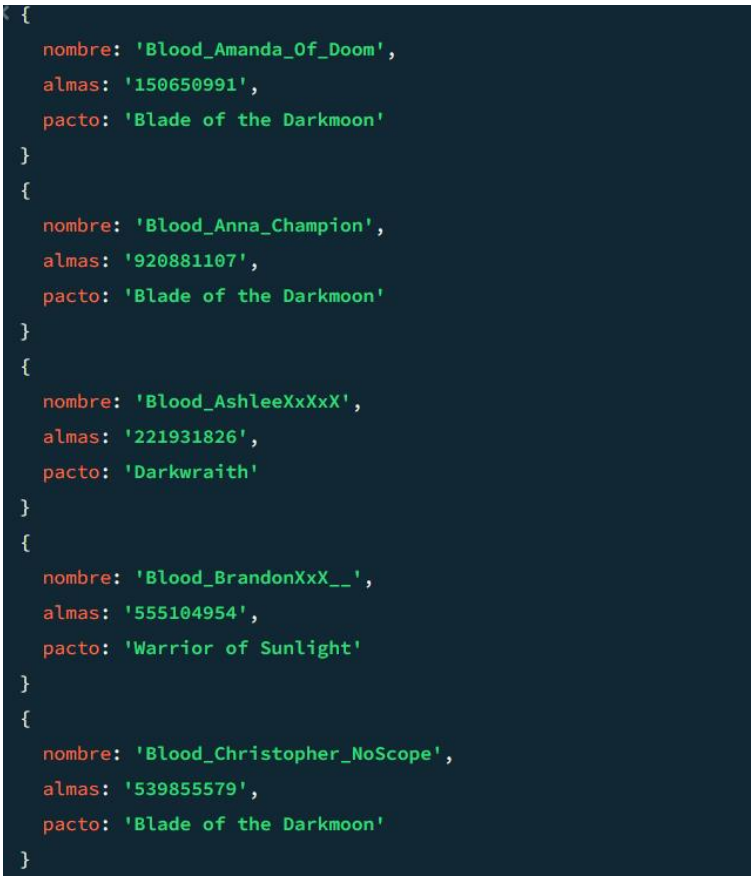


Ilustración 42. Primeros resultados de consulta 4.4.2

La lista continúa, pero con esta muestra se ven los campos que aparecen y el orden alfabético. En cuanto a estadísticas se tienen las siguientes:

Sin índices

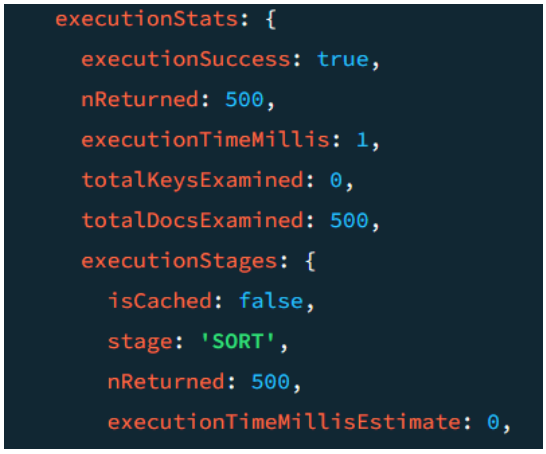


Ilustración 43. Estadísticas pre-índice consulta 4.4.2

Como se puede ver se acceden a 500 documentos y tarda 1 ms. Para mejorar esto se crea el índice de nombres y se vuelve a realizar la consulta usando el .hint. Como la consulta devolverá todos los jugadores, se va a crear el índice con los datos necesarios.
db.jugadores.createIndex({ nombre: 1, almas: 1, pacto: 1})

Con índices

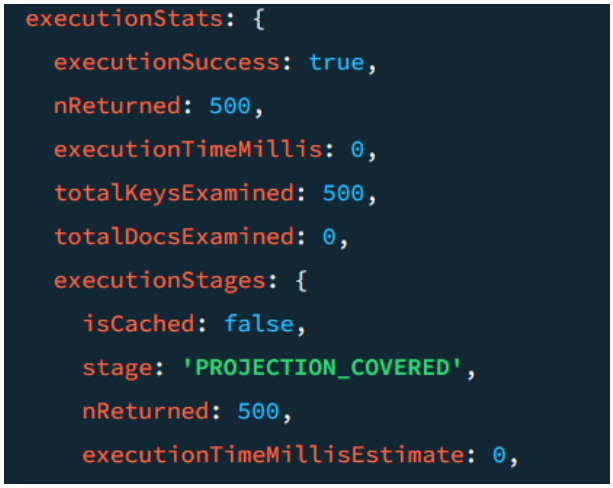


Ilustración 44. Estadísticas post-índice consulta 4.4.2

Así se puede reducir el tiempo ya que solo se accede al índice.

4.3. Consultas nuevas con Aggregation Framework

4.3.1. Consulta 1 de Aggregation Framework

Tiempos entre muertes, hacer la media por cada jugador y ordenarlos

La agregación para esta consulta se compone de 6 fases, 5 de proyección y 1 de ordenación. Cada fase realiza la siguiente operación para lograr obtener la información deseada de forma incremental:

1º. El campo de tiempo de la muerte, que se codifica en tiempo de juego almacenado como una cadena de caracteres con el formato HHH:MM:SS.MS, se toma para calcular con él un valor numerico en milisegundos (desde el comienzo de la partida, al crear el personaje) del momento en el que murió el jugador. Esto se hace para todos los tiempos de muerte de todas las manchas de sangre de todos los jugadores.

```
// Tiempo de muerte convertido a ms de juego de cada jugador
  {
    $project: {
      nombre: 1,
      tiempos_muerte_ms: {
        $map: {
          input: "$manchas_sangre",
          as: "mancha",
          in: {
            $let: {
              vars: { parts: { $split: ["$$mancha.tiempo_muerte", ":"] } },
              in: {
                $add: [
                  { $multiply: [{ $toInt: { $arrayElemAt: ["$$parts", 0] } }, 3600000] }, // hours → ms
                  { $multiply: [{ $toInt: { $arrayElemAt: ["$$parts", 1] } }, 60000] },   // minutes → ms
                  { $toInt: { $multiply: [{ $toDouble: { $arrayElemAt: ["$$parts", 2] } }, 1000] } } // seconds (with decimals)
                ]
              }
            }
          }
        }
      }
    }
  }
```

2º. Los valores calculados previamente “tiempo_muerte_ms” se utilizan para calcular para cada jugador, todos los intervalos de tiempo transcurridos entre cada tiempo de muerte registrado. Así se averigua cuanto tiempo pasó vivo el jugador, en unidades de tiempo de juego, sin contar la inactividad.

```
// Intervalo de tiempo entre cada muerte de cada jugador
  {
    $project: {
      nombre: 1,
      intervalos_muerte: {
        $map: {
          input: { $range: [0, { $size: "$tiempos_muerte_ms" }, 2] },
          as: "index",
          in: { $slice: ["$tiempos_muerte_ms", "$$index", 2] }
        }
      }
    }
  }
```

3º. Todos los intervalos de muerte calculados para cada jugador se ponderan para averiguar cuánto tiempo de juego aguanta sin morir ese jugador.

```
// Media de los intervalos de tiempo entre muertes de cada jugador
  {
    $project: {
      nombre: 1,
      tiempo_vida_medio: {
        $avg: {
          $map: {
            input: "$intervalos_muerte",
            as: "par_muertes",
            in: { $subtract: [{ $arrayElemAt: ["$$par_muertes", 1] }, { $arrayElemAt: ["$$par_muertes", 0] }] }
          }
        }
      }
    }
  }
```

4º. El tiempo de vida medio, que sigue siendo una cantidad de tiempo expresada en milisegundos, se reinterpreta al formato original de HH:MM:SS.MS, para que la información sea más significativa y fácil de leer.

```
// Tiempo medio de vida bien formateado
  {
    $project: {
      nombre: 1,
      tiempo_vida_medio: 1,
      tiempo_vida_medio_formateado: {
        $let: {
          vars: {
            total_ms: "$tiempo_vida_medio",
            horas: { $floor: { $divide: ["$tiempo_vida_medio", 3600000] } },
            minutos: { $floor: { $divide: [{ $mod: ["$tiempo_vida_medio", 3600000] }, 60000] } },
            segundos_decimales: { $divide: [{ $mod: ["$tiempo_vida_medio", 60000] }, 1000] }
          },
          in: {
            $concat: [
              // Horas padded to 2 digits
              { $cond: [{ $lte: ["$$horas", 9] }, { $concat: ["0", { $toString: "$$horas" }] }, { $toString: "$$horas" }] },
              ":",
              // Minutos padded
              { $cond: [{ $lte: ["$$minutos", 9] }, { $concat: ["0", { $toString: "$$minutos" }] }, { $toString: "$$minutos" }] },
              ":",
              {
                $let: {
                  vars: {
                    seg: { $floor: "$$segundos_decimales" },
                    // padding trick
                    ms: { $substrBytes: [{ $concat: [{ $toString: { $round: [{ $mod: ["$$segundos_decimales", 60] }, 3] } }, "000"] }, 0, 6] }
                  },
                  in: { $cond: [{ $lte: ["$$seg", 9] }, { $concat: ["0", "$$ms"] }, "$$ms"] }
                }
              }
            ]
          }
        }
      }
    }
  }
```

5º. Se ordenan los jugadores según el campo de tiempo de vida medio (en milisegundos), para crear un ranking de la longevidad de los jugadores.

6º. Finalmente se seleccionan los campos relevantes para mostrar en el ranking, el nombre y el tiempo de vida medio (ya con formato legible) de cada jugador.

```
db.jugadores.aggregate([
  // Tiempo de muerte convertido a ms de juego de cada jugador
  {
    $project: { ... }
  },
  // Intervalo de tiempo entre cada muerte de cada jugador
  {
    $project: { ... }
  },
  // Media de los intervalos de tiempo entre muertes de cada jugador
  {
    $project: { ... }
  },
  // Tiempo medio de vida bien formateado
  {
    $project: { ... }
  },
  // Jugadores ordenados segun su tiempo de vida medio (orden descendente)
  { $sort: { tiempo_vida_medio: -1 } },
  // _id: 0, --> Si queremos esconder los ids de los jugadores
  { $project: { _id: 0, nombre: 1, tiempo_vida_medio_formateado: 1 } }
])
```

Al tratarse de una consulta tan elaborada, su rendimiento es, previsiblemente, algo malo, ya que requiere de muchos pasos intermedios que implican operaciones concretas que manipulan muchos datos como el cambio de formato y el calculo de intervalos y medias. Por ello se ha realizado un estudio en profundidad sobre 3 posibles implementaciones alternativas (V1, V1.5 y V2), y se ha medido su rendimiento tanto sin como con índices para averiguar cuál es la mejor opción.

*El código previamente presentado es la consulta V1, que resultó ser la versión con mejor rendimiento, junto con el índice *manchas_sangre_tiempo_muerte_1*.


```
[
  {
    nombre: 'Master_Lauraxx',
    tiempo_vida_medio_formateado: '00:19:19.437'
  },
  {
    nombre: 'Lord_Debra_Reaper',
    tiempo_vida_medio_formateado: '00:18:52.538'
  },
  {
    nombre: 'XxJordan666',
    tiempo_vida_medio_formateado: '00:18:43.122'
  },
  {
    nombre: 'Night_Rebecca_Slayer',
    tiempo_vida_medio_formateado: '00:18:09.5030'
  },
  {
    nombre: 'Sir_Steven_Of_Doom',
    tiempo_vida_medio_formateado: '00:18:07.3550'
  },
  {
    nombre: 'Blood_TheresaXxXxXx',
    tiempo_vida_medio_formateado: '00:18:01.6450'
  },
  {
    nombre: 'Master_Micheal_420BlazeIt',
    tiempo_vida_medio_formateado: '00:17:58.292'
  },
  ... 420 more items
]
```


Ilustración 45. Resultado de la consulta de longevidad de los jugadores.

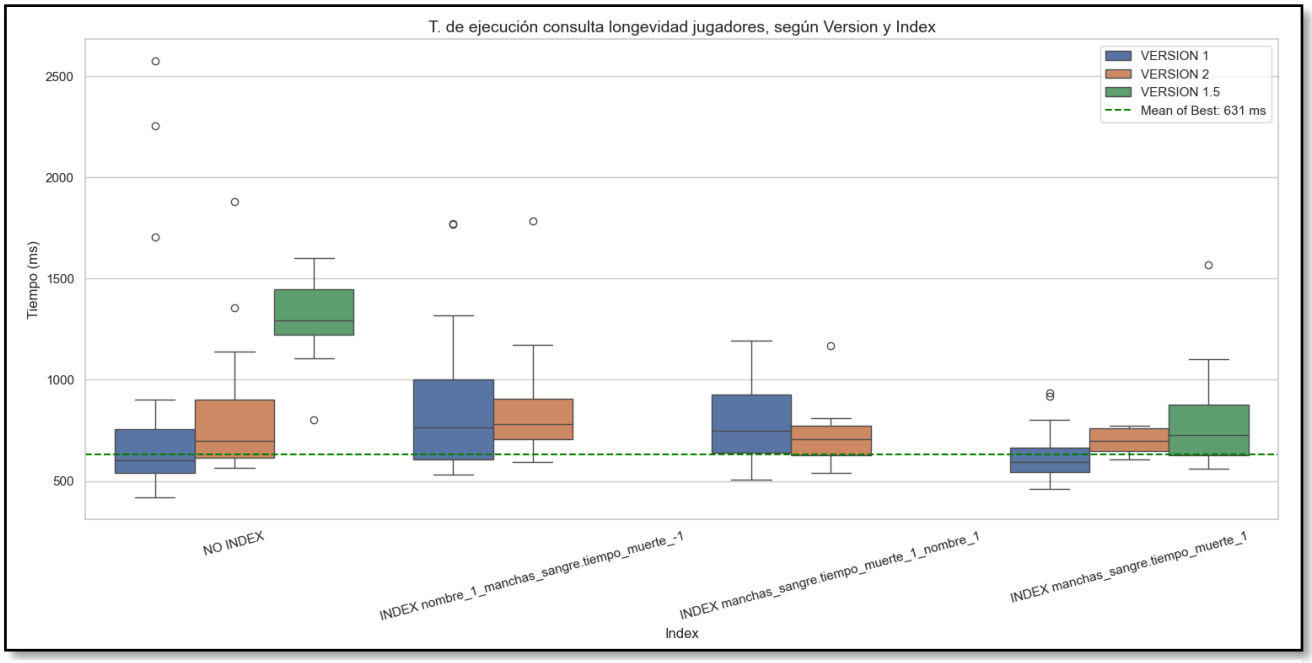


Ilustración 46. Medidas de rendimiento de distintas versiones e índices para la consulta de longevidad.

```
DarkSoulsBBDD> db.jugadores.createIndex({"manchas_sangre.tiempo_muerte": 1})
```

Ilustración 47. Índice creado para optimizar el rendimiento de la consulta de longevidad.

4.3.2. Consulta 2 de Aggregation Framework

Jugadores y sus equipaciones (mostrar sus nombres) ordenados por el número de muertes del jugador (orden ascendente)

```
db.jugadores.aggregate([
  { $lookup : {
    from: "objetos",localField: "arma_equipada",foreignField: "_arma_id",as: "arma" }
  },
  { $lookup : {
    from: "objetos",localField: "hechizo_equipado",foreignField: "_hechizo_id",as: "hechizo" }
  },
  { $lookup: {
    from: "objetos",localField: "casco_equipado",foreignField: "_pieza_armadura_id",as: "casco"}
  },
  { $lookup: {
    from: "objetos", let: { idBuscado: "$torso_equipado" }, pipeline: [ { $match: { $expr: { $and:[
      { $eq: ["$slot_armadura", "Torso"] },
      { $eq: ["$_pieza_armadura_id", "$$idBuscado"] }
    ] } } } ], as: "torso"}
  },
  { $lookup: {
    from: "objetos",let: { idBuscado: "$pernera_equipada" }, pipeline: [ { $match: { $expr: { $and: [
      { $eq: ["$slot_armadura", "Piernas"] },
      { $eq: ["$_pieza_armadura_id", "$$idBuscado"] }
    ] } } } ],as: "pernera"}
  },
  { $lookup: {
    from: "objetos",localField: "guantes_equipado",foreignField: "_pieza_armadura_id",as: "guantes"}
  },
  { $lookup: {from: "objetos",localField: "item_equipado",foreignField: "_item_id",as: "item"}
  },
  { $addFields: { "numero_muertes": {$size: "$manchas_sangre"} } },
  {
    $project: { jugador: "$nombre", arma: {$arrayElemAt:["$arma.nombre",0]} , hechizo: {$arrayElemAt:["$hechizo.nombre",0]},armadura: {
      casco: {$arrayElemAt:["$casco.nombre",0]}, torso: {$arrayElemAt:["$torso.nombre",0]}, pernera: {$arrayElemAt:["$pernera.nombre",0]},
      guantes: {$arrayElemAt:["$guantes.nombre",0]} }, items: {$arrayElemAt:["$item.nombre",0]}, numero_muertes: "$numero_muertes"}
  },
  { $sort: { "numero_muertes": 1} },
])
```

Ilustración 48. Jugadores y sus equipaciones(mostrar sus nombres) ordenados por el número de muertes del jugador (orden ascendente)

Resultado de la consulta:

```
{
  _id: ObjectId('68251c891390d1c6abb91474'),
  jugador: 'Sir_Steven_Of_Doom',
  arma: 'Shotel',
  hechizo: 'Chaos Storm',
  armadura: {
    casco: 'Six-Eyed Helm of the Channelers',
    torso: 'Eastern Armor',
    pernera: 'Sorcerer Boots',
    guantes: 'Giant Gauntlets'
  },
  items: 'Large Magic Ember',
  numero_muertes: 50
}
```

SIN ÍNDICES

```
print(`Execution time: ${end-start} ms`);
Execution time: 1554 ms
```

Ilustración 49. Consulta 2 aggregation framework sin índices

CON ÍNDICES

Índice: Nombre

```
print(`Execution time: ${end-start} ms`);
Execution time: 1559 ms
```

Índice: Manchas de sangre

```
print(`Execution time: ${end-start} ms`);
Execution time: 1568 ms
```

Índice: Arma equipada

```
print(`Execution time: ${end-start} ms`);
Execution time: 1516 ms
```

Índice: Hechizo equipado

```
print(`Execution time: ${end-start} ms`);
Execution time: 1508 ms
```

Índice: Casco equipado

```
print(`Execution time: ${end-start} ms`);
Execution time: 1574 ms
```

Índice: Torso equipado

```
print(`Execution time: ${end-start} ms`);
Execution time: 1583 ms
```

Índice: Pernera equipada

```
print(`Execution time: ${end-start} ms`);
Execution time: 1543 ms
```

Índice: Guantes equipados

```
print(`Execution time: ${end-start} ms`);
Execution time: 1660 ms
```

Índice: Ítem equipado

```
print(`Execution time: ${end-start} ms`);
Execution time: 1653 ms
```

Índice: Arma id (objetos)

```
print(`Execution time: ${end-start} ms`);
Execution time: 1458 ms
```

Índice: Hechizo id

```
print(`Execution time: ${end-start} ms`);
Execution time: 1436 ms
```

Índice: Pieza armadura id

```
print(`Execution time: ${end-start} ms`);
Execution time: 1049 ms
```

Índice: Item id

```
print(`Execution time: ${end-start} ms`);
Execution time: 1422 ms
```

Índice: Ids arma, hechizo, armadura e item

```
print(`Execution time: ${end-start} ms`);
Execution time: 1417 ms
```

Ilustración 50. Consulta 2 aggregation framework con índices

No podemos comprobar el número de documentos a los que se accede, pero calculando el tiempo que tarda podemos comparar las distintas opciones de los índices. Se ha hecho una larga experimentación de tiempos para esta consulta y se ha podido concluir que la creación de índices para los campos que se utilizan del jugador no supone ninguna mejora (al hacer varias veces esta consulta, como es muy compleja, los tiempos fluctúan bastante, por lo que algunos tiempos han salido peores que el que no usaba índices, pero en realidad no significa que haya empeorado el tiempo de la consulta).

En el caso de crear índices de los objetos (sus ids son los que permiten diferenciar el tipo de objeto del que se trata), podemos comprobar que hay una ligera mejora, de unos 100 ms aproximadamente, a excepción de las piezas de armadura, que reducen el tiempo de la consulta hasta medio segundo. Esto es debido a que gran parte de la carga de trabajo de esta consulta es por la agregación de las cuatro piezas de armadura.

Por lo tanto, en este caso, la manera óptima de reducir el tiempo en esta consulta es creando un índice de las piezas de armadura.

4.4. Consultas de Agregación y combinación

4.4.1.1. *Consulta 1 de agregación y combinación*

Los 7 jugadores con más magias en su posesión, mostrar su nombre, id y cantidad de magias

```
db.jugadores.aggregate([
{$project: {_id: 1, nombre: 1, cantidad_magias: { $size: "$lista_hechizos" }}},
  { $sort: { cantidad_magias: -1 } },
  { $limit: 7 }
```

Las etapas son las siguientes:

- Project construye una vista temporal de la colección de jugadores con solo tres elementos, el nombre, el id y el número de hechizos, que se calcula obteniendo el tamaño del array lista_hechizos.
- Después se ordenan todos los jugadores con sort según su número de hechizos de manera inversa.
- Por último, se limita el número de jugadores que se muestran a 7 usando limit.

Ilustración 51. Primeros 5 jugadores con más hechizos

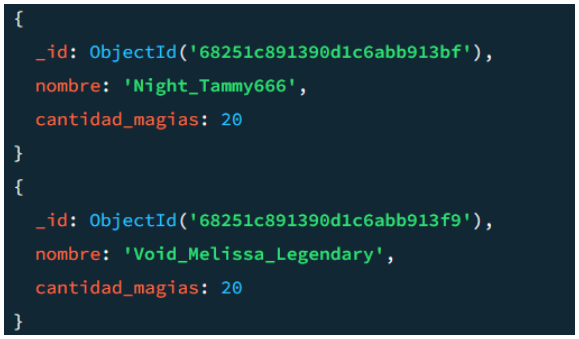


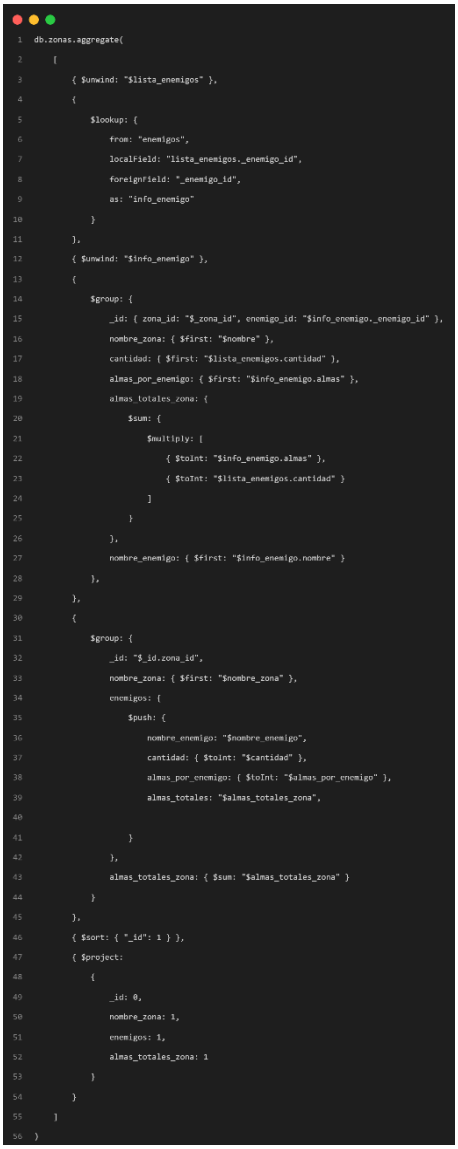
Ilustración 52. Jugadores 6º y 7º con más hechizos

Como el sort se realiza sobre un campo computado, no va a ser de utilidad usar un índice en esta consulta. Otra opción sería crear un campo en los jugadores que tenga el número de hechizos, y luego el índice de esto. Esto mejoraría la eficiencia si se diese el caso de que esta es una consulta frecuente.

Igualmente se ha medido el tiempo usando new Date( ) y la consulta tarda 36 ms.

4.4.1.2. *Consulta 2 de agregación y combinación*

Suma del número total de almas que te da cada tipo enemigo a lo largo de cada zona.



```
db.zonas.aggregate(
    [
        { $unwind: "$lista_enemigos" },
        {
            $lookup: {
                from: "enemigos",
                localField: "lista_enemigos._enemigo_id",
                foreignField: "_enemigo_id",
                as: "info_enemigo"
            }
        },
        { $unwind: "$info_enemigo" },
        {
            $group: {
                _id: { zona_id: "$_zona_id", enemigo_id: "$info_enemigo._enemigo_id" },
                nombre_zona: { $first: "$nombre" },
                cantidad: { $first: "$lista_enemigos.cantidad" },
                almas_por_enemigo: { $first: "$info_enemigo.almas" },
                almas_totales_zona: {
                    $sum: {
                        $multiply: [
                            { $toint: "$info_enemigo.almas" },
                            { $toInt: "$lista_enemigos.cantidad" }
                        ]
                    }
                },
                nombre_enemigo: { $first: "$info_enemigo.nombre" }
            },
        },
        {
            $group: {
                _id: "$_id.zona_id",
                nombre_zona: { $first: "$nombre_zona" },
                enemigos: {
                    $push: {
                        nombre_enemigo: "$nombre_enemigo",
                        cantidad: { $toint: "$cantidad" },
                        almas_por_enemigo: { $toInt: "$almas_por_enemigo" },
                        almas_totales: "$almas_totales_zona",

                    }
                },
                almas_totales_zona: { $sum: "$almas_totales_zona" }
            }
        },
        { $sort: { "_id": 1 } },
        { $project:
            {
                _id: 0,
                nombre_zona: 1,
                enemigos: 1,
                almas_totales_zona: 1
            }
        }
    ]
)
```

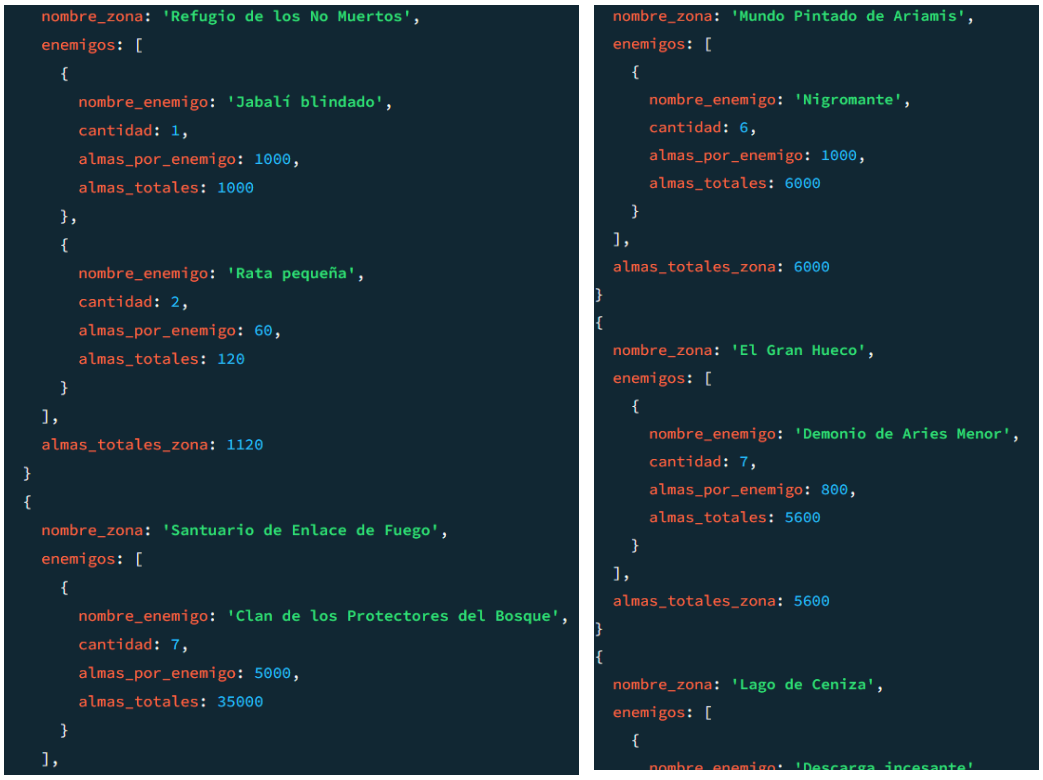
Ilustración 53. Suma del número total de almas que te da cada tipo enemigo a lo largo de cada zona.

Se establecen 7 etapas en el pipeline de agregación:

- Unwind: Se deconstruye el array lista_enemigos de cada documento de zonas, generando un documento independiente por cada elemento del array. Tras esto, cada documento tendrá un campo lista enemigos que es un objeto con enemigo_id y cantidad.
- Lookup hace join con la colección enemigos.
- Unwind deconstruye el array info_enemigo, dejando en el campo info_enemigo directamente el objeto con los datos del enemigo.
- Group: para agrupar cada par zona-enemigo y calcular cuantas almas genera ese tipo de enemigo en esa zona.
- Group: segunda agrupación, por zona. Calcula la suma total de almas que se pueden conseguir en la zona.

- Sort: Ordena los documentos resultantes por el campo id (que es la zona_id) en orden ascendente.
- Project: para generar el documento final.

Resultado de la consulta:



```
  nombre_zona: 'Refugio de los No Muertos',
  enemigos: [
    {
      nombre_enemigo: 'Jabalí blindado',
      cantidad: 1,
      almas_por_enemigo: 1000,
      almas_totales: 1000
    },
    {
      nombre_enemigo: 'Rata pequeña',
      cantidad: 2,
      almas_por_enemigo: 60,
      almas_totales: 120
    }
  ],
  almas_totales_zona: 1120
}
{
  nombre_zona: 'Santuario de Enlace de Fuego',
  enemigos: [
    {
      nombre_enemigo: 'Clan de los Protectores del Bosque',
      cantidad: 7,
      almas_por_enemigo: 5000,
      almas_totales: 35000
    }
  ],
```

```
  nombre_zona: 'Mundo Pintado de Ariamis',
  enemigos: [
    {
      nombre_enemigo: 'Nigromante',
      cantidad: 6,
      almas_por_enemigo: 1000,
      almas_totales: 6000
    }
  ],
  almas_totales_zona: 6000
}
{
  nombre_zona: 'El Gran Hueco',
  enemigos: [
    {
      nombre_enemigo: 'Demonio de Aries Menor',
      cantidad: 7,
      almas_por_enemigo: 800,
      almas_totales: 5600
    }
  ],
  almas_totales_zona: 5600
}
{
  nombre_zona: 'Lago de Ceniza',
  enemigos: [
    {
      nombre_enemigo: 'Descarga incesante'
```


Ilustración 54. Resultado de la consulta 2 de agregación y combinación

```
Tiempo de ejecución: 34 ms
```

4.4.1.3. *Consulta 3 de agregación y combinación*

Mostrar el número total de cada enemigo contando de zona en zona

```
var start = Date.now();
var resultado = db.zonas.aggregate([
  { $unwind: "$lista_enemigos" },
  { $group: {
      _id: { zona: "$nombre", enemigoId: "$lista_enemigos._enemigo_id" },
      total: { $sum: { $toInt: "$lista_enemigos.cantidad" } }
  }},
  { $lookup: {
      from:       "enemigos",
      localField: "_id.enemigoId",
      foreignField:"_enemigo_id",
      as:         "detalle"
  }},
  { $unwind: "$detalle" },
  { $project: { _id:0, zona:"$_id.zona", enemigo:"$detalle.nombre", total:1 }}
]).toArray();
var end = Date.now();
printjson({ tiempo_ms: end - start });
printjson(resultado);
```

Ilustración 55. Mostrar el número total de cada enemigo contando de zona en zona

Se establecen 5 etapas en el pipeline de agragación:

- Unwind para dividir la lista de enemigos en documentos por cada elemento.
- Group para agrupar los documentos de la etapa anterior desdoblados por las claves zona: “$nombre” y enemigoId: “$lista_enemigos._enemigo_id”;.
- Lookup para buscar el documento cuyo _enemigo_id coincida con el enemigoId del grupo y añadirlo al campo “detalle”.
- Unwind de nuevo para sacar el objeto “detalle” del array y poder mostrar sus campos.
- Project para construir el documento final y mostrarlo tomando zona, enemigo y la suma total.

Resultado de la consulta:

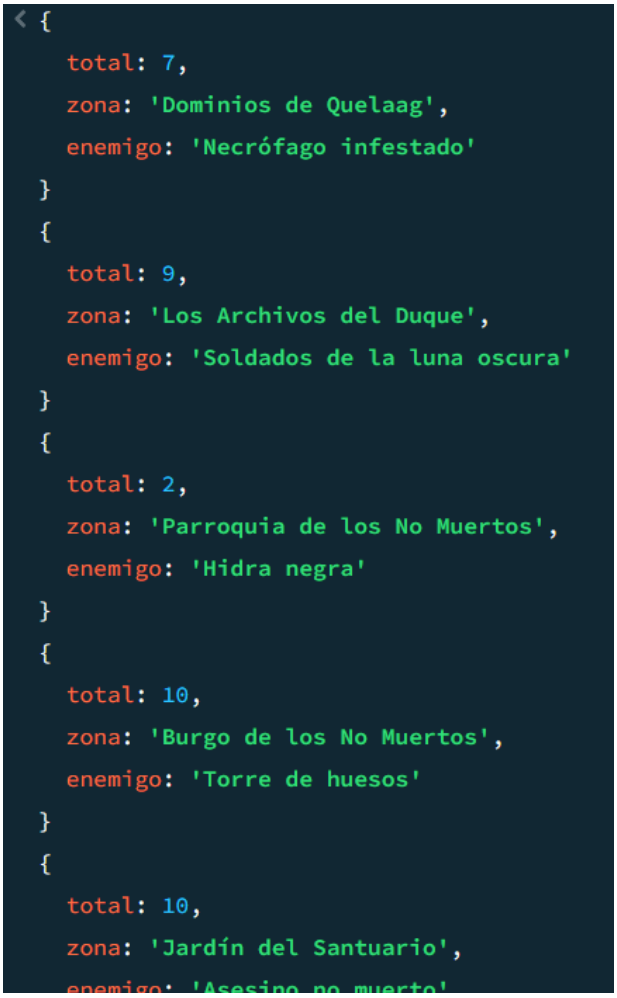


Ilustración 56. Consulta 3 de agregación y combinación

5. Conclusiones

A lo largo de este trabajo se ha adaptado la base de datos relacional de las anteriores dos prácticas al programa de Bases de datos NoSQL MongoDB, que tomaba como referencia el videojuego Dark Souls. Esta adaptación ha supuesto un rediseño, que ha permitido optimizar tanto el almacenamiento como la eficiencia de acceso a los datos, gracias a decisiones clave como la unificación de entidades (jugadores, partidas e invasiones), que han simplificado las consultas y mejorado la coherencia del modelo. Además, se ha introducido la entidad “mancha de sangre”, no solo para reflejar una mecánica fundamental del juego, sino que también para hacer un mejor uso de las posibilidades de MongoDB y así poder realizar un análisis más detallado de la información de los jugadores.

En cuanto al diseño de operaciones CRUD, aunque algunas consultas originales no se pudieron replicar exactmente, se han adaptado correctamente al modelo documental de MongoDB. El trabajo con consultas más complejas ha permitido comprobar la capacidad del modelo para realizar rápidamente las que combinan múltiples condiciones, filtros por subdocumentos y operaciones de conteo y agrupación. El rediseño del esquema y la utilización de índices se ha visto reflejado en el rendimiento de las consultas, especialmente en colecciones grandes como la de jugadores. Asimismo, la comparación del tiempo de ejecución entre consultas con y sin índices ha permitido obtener conclusiones relevantes sobre su impacto real en el rendimiento.

Para finalizar, este trabajo, junto con los anteriores, ha permitido el profundo aprendizaje del funcionamiento de una base de datos real en un entorno tan complejo como el de un videojuego. Gracias al desarrollo y a lo aprendido en cada una, se ha podido realizar los cambios de enfoque necesarios para una correcta implementación de la Base de Datos final. El resultado es un muy buen ejemplo de una Base de Datos con una estructura que ofrece flexibilidad y escalabilidad, sentando una base sólida para un hipotético desarrollo posterior. Además, la realización de las distintas consultas ha servido no solo para validar el modelo, sino también para aprender buenas prácticas en el uso de MongoDB.

6. Bibliografía

- Documentación de la librería element tree
- Wiki Dark Souls
- Manual de operaciones CRUD de MongoDB