Sprawozdanie Etap 3 - MongoDB

Zaawansowane systemy baz danych

Krzysztof Dąbrowski (293101)
GR6 (Poniedziałek 8:15)

Spis treści

Spis rycin

Spis tabel

Spis wykazów

Kod projektu

Kod projektu i sprawozdań jest dostępny na GitHub BoardBase

1 Wybrany zbór danych

W liceum zacząłem grać w grę karcianą Magic: The Gathering (MTG), która do dziś jest moim hobby. Gra MTG jest bardzo złożona, do tego stopnia, że za pomocą kart i mechanik *można zbudować maszynę Turinga* [1].
Gra zawiera bardzo wiele kart, z których każda ma zestaw parametrów zależnych od jej typu. Poza cechami kart związanymi bezpośrednio z rozgrywką, karty mają też dodatkowe cechy takie jak sety, w których zostały wydane, grafikę, która może różnić się między wydaniami, czy też przynależność do talii lub legalność w danych formatach gry. Istotnym cechą kart są też ich ceny, których zmienność prowadzi nawet do spekulacji cenowych i inwestycji niektórych graczy.

Z uwagi na moje zainteresowanie MTG, oraz złożoność danych o kartach i relacji między nimi zdecydowałem się użyć zbioru na ten temat. W tym celu skorzystałem z projektu MTGJSON [2], który jest otwarto źródłowym projektem katalogującym wiele informacji o MTG w przenośnych formatach jak na przykład JavaScript Object Notation (JSON).

W ramach projektu dostępne są tysiące zbiorów danych, ponieważ dla każdej oficjalnej talii czy set'u dostępne są dedykowane zbiory. Postanowiłem się skupić na formacie standard [3] i jednym wybranym secie i talii. Wybrałem więc zbiory:

- AllPricesToday - aktualne ceny każdego wydania dla każdej karty
- DeckList - metadane każdej oficjalnej talii
- SetList - metadane każdego dodatku
- Keywords - słowa kluczowe z kart
- CardTypes - typy kart
- Standard - podzbiór wszystkich kart legalnych w formacie standard [3]
- StandardAtomic - podstawowe informacje o kartach legalnych w formacie standard (bez rozróżnienia na różne wydania tej samej karty)
- set Edge of Eternities (endpoint do pobrania informacji o wybranym dodatku)
- deck World Shaper - (endpoint do pobrania informacji o wybranej talii)

Dane zawierają złożone struktury, wiele pól i referencje na dane z innych zbiorów.

2 Instalacja MongoDB i wczytanie danych

Przygotowanie bazy MongoDB [4].

2.1 Instalacja MongoDB

Wybrałem wersję MongoDB Community, ponieważ chciałem skonfigurować lokalną wersję bazy i nie płacić za licencję wersji Enterprise. Rozważałem też użycie MongoDB Atlas [5], ale uznałem, że wolę lokalnie działającą wersję.

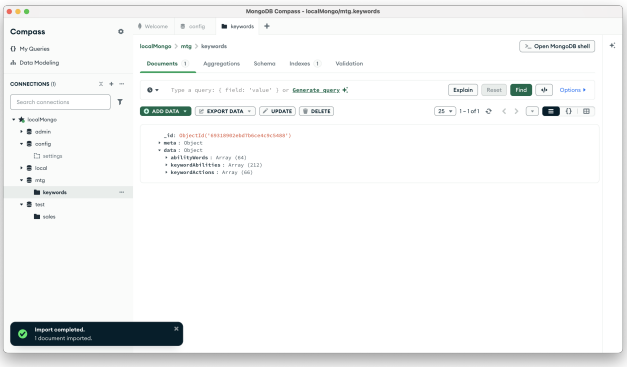
Ponieważ w poprzednich etapach dobrze sprawdzała mi się instalacja jako obraz Docker tym razem też się na to zdecydowałem. Skonfigurowałem uruchomienie obrazu w pliku `docker-compose.yml`, kierując się oficjalnym poradnikiem Install MongoDB Community Edition [6].
Niestety oficjalna strona obrazu wersji community [7] nie opisuje jakie są możliwe tagi, ani jak rozumieć zastosowane w nich skróty. Na szczęście strona do wersji open source obrazu MongoDB [8] jest lepiej opisana.
Nie byłem pewien, który obraz wybrać, ale sugerując się wątkiem na reddit [9] pozostałem przy obrazie `mongodb/mongodb-community-server`.
Dowiedziałem się, że jest on dostępny w wersji bazującej na Redhat i Ubuntu. Wybrałem wersję na Ubuntu, ponieważ mam większe doświadczenie z tą dystrybucją, choć pewnie nie będzie to miało znaczenia.

Po uruchomieniu kontenera połączyłem się do niego `mongosh` i pobrałem informacje o bazie poleceniem `db.runCommand({hello:1})` jak w samouczku [6].
Zobaczyłem JSON z informacjami o bazie, co potwierdza, że działą ona poprawnie.

Skonfigurowałem login i hasło użytkownika `root`, żeby ograniczyć dostęp do bazy. Rozważałem też ustawienie innych opcji [10], ale uznałem, że na tym etapie domyślne ustawienia mi pasują.

2.2 Wczytanie danych

Na początku spróbowałem wczytać dane za pomocą MongoDB Compass (Compass) [11]. Zorientowałem się jednak, że ponieważ moje dane są pojedynczymi plikami JSON to do kolekcji został dodany tylko jeden dokument zawierający wszystkie dane (Rysunek 1).



Rysunek 1: Wczytanie pliku za pomocą MongoDB Compass

W dokumentacji przeczytałem, że dobrym sposobem na wczytanie danych z JSON jest użycie narzędzia `mongoimport` razem z `jq` do transformacji danych [12]. Skrypty wczytujące dane umieściłem w katalogu `mongo/importData/`.

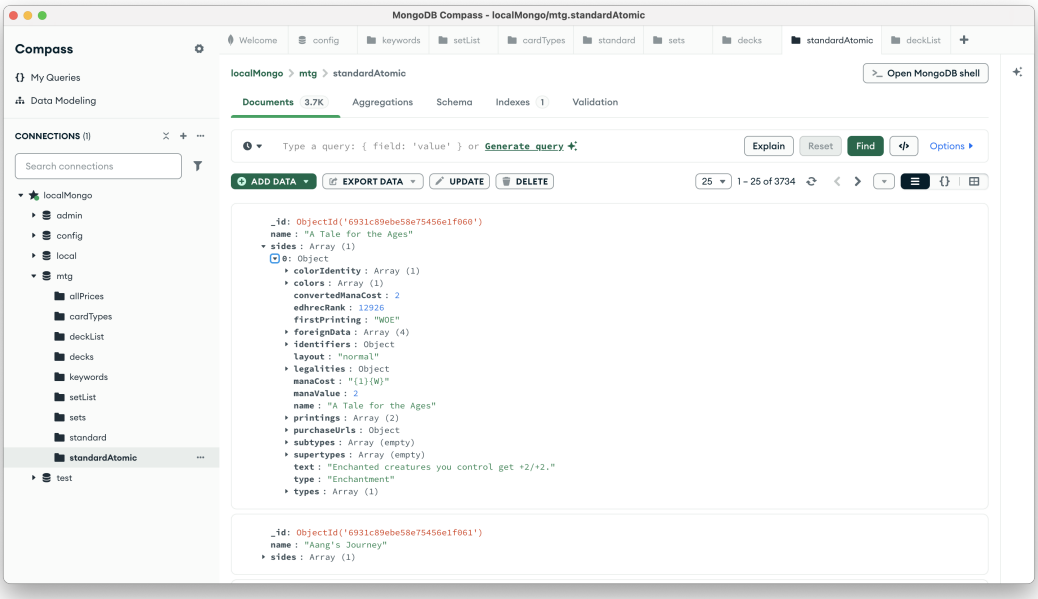
Na początku miałem problem z uwierzytelnieniem. Udało mi się go rozwiązać kodując hasło w connection string używając URL encoding, zgodnie z przykładem z poradnika Decoding Encoded URLs in Linux [13].

Po wczytaniu danych zauważyłem, że wszystkie pola mają typ `string`. Przeczytałem, że w przypadku importu z JSON wszystkie pola są zapisywane jako `string` [14]. Przygotowałem więc skrypty `.mongodb.js` w `mongo/importData/`, które dostosowują typy we wczytanych dokumentach.

2.3 Podział na kolekcje

Ponieważ projekt MTGJSON [2] już rozsądnie grupuje różne rodzaje danych, to w mojej bazie załadowałem każdy ze zbiorów danych do osobnej kolekcji. W przypadku niektórych zbiorów podzieliłem dane na mniejsze dokumenty. Zrobiłem tak na przykład w `importKeywords.sh`.
Gdybym używał danych wielu tali lub setów umieściłbym je wszystkie w kolekcjach `decks`

i `sets`. Jednak, ponieważ mam już wiele zestawów danych, to wybrałem tylko po jednej tali i secie. Utworzone kolekcje są widoczne na Rysunek 2.



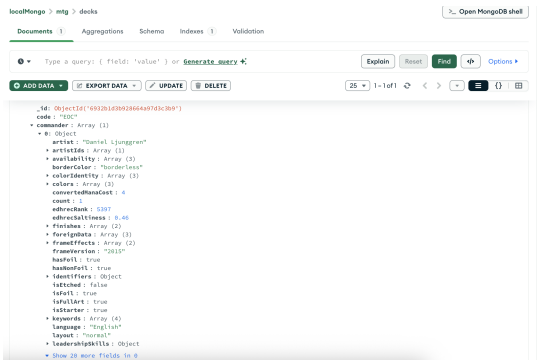
Rysunek 2: Utworzone kolekcje

2.4 Relacje

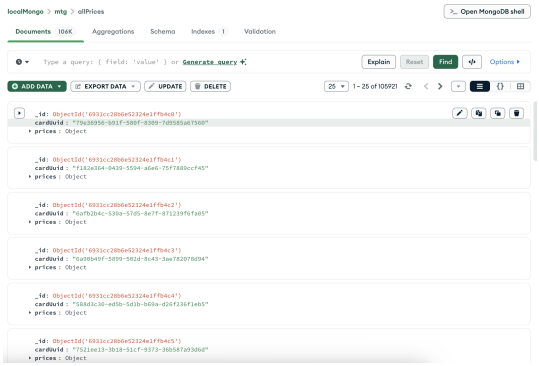
Wybrane przeze mnie zbiory posiadają już za równo zagniezdzone dokumenty jak i referencje (Rysunek 3). Nie musiałem więc modyfikować ich struktury, poza zmianami, na które zdecydowałem się przy wczytywaniu danych (Sekcja 2.2).

Różnica między zagnieżdżeniem a referencją polega na tym, że w przypadku zagniezdzenia całe dane są przechowywane w bazowym dokumencie, a referencja przechowuje tylko identyfikator danych w innym dokumencie.
Referencja przypomina bardziej znormalizowany model relacyjny. Stosując MongoDB zalecane jest jednak zagniezdzanie danych, gdy są one często czytane razem, ponieważ znacznie przyśpiesza to regularne działanie bazy [15].

Byłem ciekaw jakie są najczęstsze słowa kluczowe w kartach z dodatku *Edge of Eternities*. W tym celu napisałem zapytanie w pliku `usingNestedData.mongodb.js`, które to wylicza.
Wynik zapytania z filtrem na 3 najczęstsze słowa kluczowe widać w Wykaz 1.



(a) Przykład zagniezdzonych danych



(b) Przykład referencji

Rysunek 3: Przykładowe zagniezdzenie i referencja

Do pokazania relacji napisałem zapytanie `mostExpensiveStandardCards.mongodb.js` wyświetlające najdroższe karty legalne w standardzie. Kolekcja `standard` zawiera informacje o kartach, ale ich aktualne ceny są w kolekcji `allPrices`. Kolekcja `allPrices` zawiera jedynie id kart i dane o cenach.

Zapytanie działało bardzo wolno dla 6440 kart. Prawdopodobnie dodanie indeksów przyśpieszyłoby działanie, ponieważ indeksy są tematem innego etapu, to na razie ograniczyłem liczbę przetwarzanych kart do 100.
Wyniki zapytania dla 3 najdroższych kart przedstawiam w Wykaz 2.

3 Logika biznesowa

Do implementacji logiki biznesowej zdecydowałem się użyć zewnętrznych funkcji w MongoDB Playgrounds [16].
Rozważałem użycie `$function` [17], jednak uznałem, że oferują zbyt małe możliwości do solidnej implementacji logiki biznesowej. Dodatkowo użycie `$function` nie jest rekomendowane [17].

Napisany kod umieściłem w katalogu `mongo/businessLogic/`.

- `addKeyword.mongodb.js` - dodaje słowo kluczowe do odpowiedniego dokumentu. Sprawdzana jest poprawność typu słowa kluczowego, a następnie jest ono dodawane do odpowiedniej listy, jeśli jeszcze go tam nie ma.
- `addCardmarketPrice.mongodb.js` - dodanie ceny danej karty jako cena z Cardmarket. Funkcja znajduje `uuid` karty na podstawie jej nazwy i dodaje podaną cenę z aktualną datą.
- `addDeckList.mongodb.js` - dodanie informacji o talii. Funkcja waliduje typ talii oraz automatyczne generuje `fileName` na podstawie nazwy i kodu talii.

Wykaz 1 Top 3 słów kluczowych w dodatku Edge of Eternities

```
[
  {
    "_id": "Flying",
    "count": 59
  },
  {
    "_id": "Warp",
    "count": 52
  },
  {
    "_id": "Station",
    "count": 47
  }
]
```

Na początku implementując `addCardmarketPrice` nadpisywałem cały dokument w `$set`. Traciłem przez to ceni z poprzednich dni. Na forum [18] znalazłem rozwiązanie, że należy użyć *Dot Notation* [19], żeby zaktualizować tylko wybrane pola.

Uznałem, że w moim przypadku **sekwencje do autoinkrementacji** nie są potrzebne. Szczególnie w środowiskach rozproszonych używanie tego typu sekwencji może sprawiać problemy z synchronizacją wartości. Na przykład w sytuacji równoległego dodania dwóch dokumentów. Gdybym w przyszłości potrzebował sekwencji, to zaimplementowałbym je bazując na artykule *MongoDB Auto-Increment* [20].

Agregację danych wykonałem w zapytaniach testujących relacje. Przygotowałem też nowe zapytanie wyliczające podstawowe statystyki dla danej talii w pliku `deckStatistics.mongodb.js`.

- `usingNestedData.mongodb.js` - wyszukanie najczęstszych słów kluczowych
- `mostExpensiveStandardCards.mongodb.js` - wyszukanie najdroższych kart legalnych w standardzie
- `deckStatistics.mongodb.js` - wyliczenie podstawowych statystyk talii

Analogiczne elementy, które warto dodać w pełnej wersji aplikacji:

- funkcje do dodawania cen dla innych sklepów i cen wersji online
- funkcję do dodawania i aktualizacji typów kart
- funkcję do dodawania nowych kart lub aktualizacji istniejących
- funkcję do dodawania i aktualizacji setów
- funkcję do dodawania i aktualizacji szczegółowych informacji o taliach
- analiza trendów cen danych kart

Wykaz 2 Top 3 najdroższych kart legalnych w standardzie


```
[
  {
    "cardUuid": "73030017-9483-5f73-b4be-0660e1a5c1eb",
    "cardName": "Simulacrum Synthesizer",
    "rarity": "mythic",
    "type": "Artifact",
    "setCode": "BIG",
    "cardMarketPrice": "80.48 EUR"
  },
  {
    "cardUuid": "1ff3d49d-979d-5237-b2dd-28f4da08401a",
    "cardName": "Vaultborn Tyrant",
    "rarity": "mythic",
    "type": "Creature - Dinosaur",
    "setCode": "BIG",
    "cardMarketPrice": "61.62 EUR"
  },
  {
    "cardUuid": "c3cfc938-9b15-5ca1-9361-e89523d310dd",
    "cardName": "Simulacrum Synthesizer",
    "rarity": "mythic",
    "type": "Artifact",
    "setCode": "BIG",
    "cardMarketPrice": "47.21 EUR"
  }
]
```


- wyliczenie wartości kolekcji podanych kart
- rozbudowanie wyliczania statystyk dla danej talii o więcej parametrów

4 Indeksy

Używając zapytania wyszukującego najdroższe karty, zauważyłem, że było ono bardzo wolne. Przypuszczam, że wynika to z operacji `$lookup`. Żeby to zweryfikować użyłem metody `explain`. Pozwala ona przeanalizować dane o wykonaniu zapytania, takie jak czas poszczególnych etapów [21].
Za pomocą `explain("executionStats")` zbadałem czas wykonania poszczególnych etapów zapytania (Tabela 1).

Tabela 1: Czas zapytania przed dodaniem indeksu

Etap	Czas wykonania etapu (ms)	Całkowity czas (ms)
$unwind	10	10
$limit	0	10
$project	0	10
$lookup	7047	7057
$addFields	0	7057
$sort	0	7057
$project	0	7057

Widać, że największy narzut pochodzi z etapu `$lookup` (Tabela 1). Ponieważ `$lookup` wyszukuje dokumenty w kolekcji `allPrices` po polu `uuid`, to dodanie indeksu powinno to przyśpieszyć.

Dodałem indeks na polu `cardUuid` w kolekcji `allPrices` za pomocą skryptu `createIndexes.mongodb.js`.

Tabela 2: Czas zapytania po dodaniu indeksu

Etap	Czas wykonania etapu (ms)	Całkowity czas (ms)
$unwind	5	5
$limit	0	5
$project	0	5
$lookup	27	32
$addFields	0	32
$sort	0	32
$project	0	32

Po ponownym przeanalizowaniu zapytania widać, że czas etapu `$lookup` zmniejszył się z 7047 ms do 27 ms (Tabela 2). Jest to bardzo znacząca poprawa wydajności.
Podobnie jak w przypadku kluczy obcych w bazach relacyjnych, indeksy są kluczowe dla zaczytywania danych z innych kolekcji.

Żeby uniknąć wpływu cache na testy wydajności usuwałem kontener z bazą i tworzyłem go ponownie pomiędzy testami.

Zachęcony poprawą wydajności usunąłem ograniczenie brania tylko 100 pierwszych kart i przeanalizowałem ceny wszystkich 6440 kart ze standardu. Zapytanie, które wcześniej wykonywało się tak długo, że nie chciałem na nie czekać, teraz wykonało się poniżej 1 sekundy!

Gracze często szukają kart po ich nazwach. Żeby to przyśpieszyć dodałem indeksy na nazwach kart. W kolekcji `standardAtomic` dodałem zwykły indeks *Single Field* [22] na

polu `name`, a w kolekcji `standard` na polu `cards.name` indeks *Multikey* [23]. Dodałem podobne indeksy do wyszukiwania innych elementów po ich nazwach lub kodach.
Tego typu indeksy testowałem analizując statystyki wykonania z włączonym i wyłączonym indeksem w pliku `cardSearchByName.mongodb.js`.

Gracze często szukają kart do swoich talii bazując na kolorze karty i jej koszcie many. Żeby przyśpieszyć tego typu wyszukiwania utworzyłem *Compound Index* [24]. Ponieważ tylko przednia strona karty jest istotna przy tego typu wyborze, ustawiłem indeks na pierwszym elemencie listy stron kart (Wykaz 3).
Indeksy na tego typu zagniezdzonych wartościach nie są możliwe do utworzenia w relacyjnych bazach danych, gdzie indeksuje się całe kolumny.

Wykaz 3 Utworzenie indeksu złożonego

```
1 db.standardAtomic.createIndex({ "sides.0.colors": 1 }, {
  ↪  "sides.0.manaValue": 1 });
```

W indeksach złożonych kolejność pól ma istotne znaczenie. Kierowałem się zasadą ESR [25], najpierw wybierając kolor, po którym użytkownicy będą zazwyczaj bezpośrednio filtrować, a potem dając koszt many, który będzie częściej wyszukiwany w zakresach.
W ten sposób dane o wartościach many w ramach danego koloru są **ułożone obok siebie w indeksie**, zapytania tego typu są wydajne.
Testując zapytania tego typu w pliku `cardForDeckSlot.mongodb.js` ustaliłem, że indeks znacząco przyśpiesza wyszukiwanie.

Ponieważ z czasem w MTG pojawiają się nowe formaty, na obiekcie opisującym legalność karty ustawiłem indeks *Wildcard* [26] poleceniem z Wykaz 4.
Dzięki temu rodzajowi indeksu nie muszę jawnie podawać pól, które będą indeksowane. Indeks obejmie wszystkie pola zagnieżdżone pod `sides.legalities`, nawet gdy z czasem dojdzie nowy rodzaj formatu.
Indeksy tego rodzaju raczej nie występują w bazach relacyjnych.

Wykaz 4 Utworzenie indeksu typu Wildcard

```
1 db.standardAtomic.createIndex({ "sides.legalities.$**": 1 });
```

Pozostałe indeksy, które założyłem są w pliku `createIndexes.mongodb.js`.

5 Transakcje

Transakcje w MongoDB działają tylko, gdy system składa się z więcej niż jednej repliki [27]. Żeby je przetestować skonfigurowałem dodatkowe instancje w `docker-compose.yml`.

Moją konfigurację bazowałem na artykule *Create a replica set in MongoDB with Docker Compose* [28]. Uprościłem konfigurację sieciową, w stosunku do proponowanej w artykule, korzystając z domyślnej sieci tworzonej przez Docker Compose [29]. Zautomatyzowałem też inicjalizację połączenia replik skryptem `replica-set-init.sh` za pomocą Docker Compose lifecycle hook'u `post_start`. Konfiguracja sugerowana w artykule wymagała ręcznego uruchomienia skryptu po uruchomieniu kontenerów.

Zauważałem, że po skonfigurowaniu replik moje połączenia z Playgrounds [16] i Compass przestały działać. Po przeczytaniu wątku na forum *Cannot connect to replica set via mongo compass* [30] zrozumiałem, że muszę dostosować connection string.
Na początku wypróbowałem `directConnection=true`, co sprawiło, że połączenie zadziałało. W tej konfiguracji jednak łączyłem się tylko z główną repliką, co w sytuacji potencjalnej awarii neguje zalety używania replik.
Idąc za radami z forum [30] dodałem nowe routing do `/etc/hosts` (Wykaz 5). Po tej zmianie udało mi się połączyć connection string'iem `mongodb://admin:*****@mongodb:27017,mongodb-secondary1:27018,mongodb-secondary2:27019/?replicaSet=rsmtg` do wielu replik.

Wykaz 5 Routing do replik MongoDB w /etc/hosts

```
127.0.0.1 mongodb
127.0.0.1 mongodb-secondary1
127.0.0.1 mongodb-secondary2
```

Poprawne działanie replik widać między innymi uruchamiając `db.runCommand({ hello: 1 })` (Wykaz 6).

Przykładową transakcję przygotowałem w pliku `updateSetName.mongodb.js`. Aktualizuje ona nazwę dodatku w kilku kolekcjach.
Dla transakcji skonfigurowałem odpowiedni `readConcern` i `writeConcern` [31] [32].
Po skonfigurowaniu replik, za równo przy połączeniu z `directConnection` jak i listą replik, transakcja działała poprawnie. Jednak to drugi rodzaj połączenia uważam za prawdziwe zastosowanie rozproszonych transakcji [33].

6 Zaawansowane funkcjonalności

Wybrane zaawansowane funkcjonalności MongoDB, które pasują do mojego zbioru danych.

Wykaz 6 Odpowiedź z polecenia hello po skonfigurowaniu replik

```
{
  "topologyVersion": {
    "processId": {
      "$oid": "69357bfa954b1bf67799cd35"
    },
    "counter": 7
  },
  "hosts": [
    "mongodb:27017",
    "mongodb-secondary1:27017",
    "mongodb-secondary2:27017"
  ],
  "setName": "rsmtg",
  "setVersion": 1,
  "isWritablePrimary": true,
  "secondary": false,
  "primary": "mongodb:27017",
  "me": "mongodb:27017",
  "electionId": {
    "$oid": "7fffffff0000000000000007"
  },
// Skrócony wynik ...
}
```

6.1 Przetwarzanie grafów

W MTG z danymi dodatkami wypuszczane są dodatkowe produkty, takie jak zestawy promocyjne kart z innymi grafikami. Do reprezentacji tej relacji w mojej bazie wykorzystywane jest pole `parentSet` w kolekcji `setList`.

Powiązania produktu innymi produktami mogą być wielopoziomowe. Przykładowo dodatek *Outlaws of Thunder Junction* ma rozszerzenie *The Big Score*, do którego został wydany zestaw promocyjny *The Big Score Promos*. Z tego powodu chcąc zobaczyć wszystkie suplementarne produkty nie wystarczy wyszukać `{parentSet: baseSetCode}`.

Sądzę, że do analizy tego typu relacji idealnie nadaje się mechanizm przetwarzania grafów [34]. Przejście wielu poziomów relacji można bardzo prosto zrealizować `$graphLookup`. Zapytanie, które wykorzystuje przetwarzanie grafów do znalezienia produktów powiązanych z zadanym napisałem w pliku `graphQueries.mongodb.js`

6.2 Indeks klastrowany

Tworząc kolekcję można ustawić `_id` jako indeks klastrowany. Dokumentacja zaleca to podejście, ponieważ pozwala ono często na wyelimniowanie jednego zwykłego indeksu, zmniejsza rozmiar kolekcji i przyśpiesza zapytania [35]. Postanowiłem to przetestować.

Wybrałem kolekcję `allPrices`, ponieważ mierzyłem już czytanie z niej szukając najdroższych kart w Sekcja 4.
Ponieważ indeks kastrowany można utworzyć tylko przy tworzeniu kolekcji [35] usunąłem ją i utworzyłem ponownie `clusteredCollection.mongodb.js`. Zmieniłem wczytywanie danych do kolekcji `allPrices`, umieszczając w `_id` wartość `cardUuid` `importAllPricesToday.sh`. Żeby nie musieć przerabiać poprzednich zapytań zostawiłem też jawne pole `cardUuid`.

Po skonfigurowaniu indeksu klastrowanego zmieniłem zapytanie wyszukujące nadroższe karty, żeby z niego skorzystać `mostExpensiveStandardCards-withClusteredIndex.mongodb.js`.
Następnie zbadałem czas wykonania zapytań.

Dla spólnego porównania ograniczyłem liczbę przetwarzanych kart do 100. Czas zapytania spadł do 9 ms (Tabela 3). Przy stosowaniu zwykłego indeksu zapytanie trwało 32 ms (Tabela 2).
Przetestowałem też czas wykonania zapytania dla wszystkich 6440 kart. Dla indeksu klastrowanego czas wyniósł 410 ms, a dla zwykłego indeksu trwał około 900 ms.

Tabela 3: Wyszukanie najdroższych kart z indeksem klastrowanym (tylko pierwsze 100 kart)

Etap	Czas wykonania etapu (ms)	Całkowity czas (ms)
$unwind	1	1
$limit	0	1
$project	0	1
$lookup	8	9
$addFields	0	9
$sort	0	9
$project	0	9

Stosowanie indeksu klastrowanego znacząco przyśpieszyło moje zapytanie, nawet w stosunku do zwykłego indeksu. Zdecydowanie będę korzystał z tej funkcjonalności w przyszłości.
Ciekawe było też dla mnie to, że przy analizie zapytania korzystającego z indeksu klastrowanego `explain("executionStats")` pokazuje, że nie został użyty żaden indeks `"indexesUsed": [],`. Jestem jednak pewien, że indeks klastrowany został użyty, ponieważ w przeciwnym razie wykonanie trwałoby znacznie dłużej.

6.3 Map Reduce

Byłem ciekaw ile w danym roku zostało wydanych setów w ramach danych bloków (rodzajów). Chciałem też zobaczyć jak zmienia się to na przestrzeni lat.

Wykorzystałem do tego podejście map reduce w pliku `mapReduce.mongodb.js`
Najpierw zmapowałem każdy dokument mający `releaseDate` i `block` na rok i nazwę bloku. Następnie w funkcji redukującej wyliczyłem liczbę setów dla danego roku i bloku. Na koniec, już poza map reduce, posortowałem wyniki po roku.

Całkiem wygodnie pisało mi się to zapytanie w formie map reduce. Było to dla mnie łatwiejsze niż napisanie analogicznego zapytania jako `aggregate`, choć myślę, że może to wynikać z mojej małej wprawy w zapytaniach `aggregate`.
W map reduce brakowało mi możliwości posortowania wynków. Zrobiłem je osobno zwykłym `sort` na końcu. Warto uwzględnić, że używanie map reduce nie jest już zalecane [36]. Z tego powodu nie będę raczej wracał do tej funkcjonalności.

7 Wnioski

Implementacja bazy do analizy danych o kartach MTG w MongoDB zawiera kilka istotnych różnic w stosunku do podobnego systemu zrobionego na bazie relacyjnej.

Główną różnicę, którą odczułem jest **inne podejście do modelowania danych**. W bazach relacyjnych stosuje się zazwyczaj 3NF, a w MongoDB typowe jest zagnieżdżanie i duplikacja danych [15]. Takie podejście prowadzi do znacznie **szybszych i prostszych zapytań**. Jednak umożliwia **łatwą utratę spójności bazy**, gdy na przykład zwielokrotnione dane zostaną zmienione tylko w jednym miejscu.

Baza MongoDB odchodzi od architektury ANSI-SPAC. Posiada ona widoki, ale nie daje możliwości ich edycji. Cho prawda w bazach relacyjnych trudno uzyskać działanie edycji widoków, jednak jest to teoretycznie możliwe.

MongoDB nie posiada bezpośrednio wyzwalaczy. Logika zapewniająca spójność musi znajdować się po stronie aplikacji, może to być jednak trudne, gdy różne aplikacje korzystają z tej samej bazy. Atlas oferuje wyzwalacze [37], jednak ja korzystałem z lokalnej bazy. Efekt podobny do wyzwalaczy można osiągnąć korzystając z *Change Streams* [38], jest to jednak bardziej skomplikowane.

Konfiguracja replikacji jest zdecydowanie prostsza w MongoDB niż w relacyjnych bazach danych.

Brak silnego schematu utrudnia to odpytywanie bazy, ponieważ nie można być pewnym jakie pola są dostępne w dokumentach ani nawet, że wszystkie ich wartości będą tego samego typu.

Mimo, że to był mój pierwszy kontakt z MongoDB, to pracowało mi się zdecydowanie przyjemnie. Może to wynika z mojego większego doświadczenia z JavaScriptem niż Structured Query Language (SQL), jednak gdybym miał wybierać z jaką bazą przyjdzie mi pracować, to pewnie wybrałbym MongoDB.

Źródła

1. Churchill A, Biderman S, Herrick A (2019) Magic: The Gathering is Turing Complete

2. MTGJSON - Portable formats for all Magic: The Gathering data. https://mtgjson.com/. Accessed 29 lis 2025

3. MTG Standard Format | Magic: The Gathering. https://magic.wizards.com/en/formats/standard. Accessed 29 lis 2025

4. MongoDB: The World's Leading Modern Database | MongoDB. https://www.mongodb.com/. Accessed 29 lis 2025

5. Atlas Database | MongoDB. https://www.mongodb.com/products/platform/atlas-database. Accessed 29 lis 2025

6. Install MongoDB Community Edition - Database Manual - MongoDB Docs. https://www.mongodb.com/docs/manual/administration/install-community/?operating-system=docker&search-docker=with-search-docker#install-mongodb-community-edition. Accessed 29 lis 2025

7. mongodb/mongodb-community-server - Docker Image. https://hub.docker.com/r/mongodb/mongodb-community-server. Accessed 29 lis 2025

8. mongo - Official Image | Docker Hub. https://hub.docker.com/_/mongo. Accessed 29 lis 2025

9. (2023) Difference between mongo and mongodb/community-server : r/mongodb. https://www.reddit.com/r/mongodb/comments/1658qs8/difference_between_mongo_and/. Accessed 29 lis 2025

10. Self-Managed Configuration File Options - Database Manual - MongoDB Docs. https://www.mongodb.com/docs/manual/reference/configuration-options/#mongodb-setting-auditLog.auditEncryptionKeyIdentifier. Accessed 4 grudz 2025

11. MongoDB Compass | MongoDB. https://www.mongodb.com/products/tools/compass. Accessed 4 grudz 2025

12. mongoimport Example Guide - Database Tools - MongoDB Docs. https://www.mongodb.com/docs/database-tools/mongoimport/mongoimport-guide/#useful-command-line-tools. Accessed 4 grudz 2025

13. Ballard M (2022) Decoding Encoded URLs in Linux | Baeldung on Linux. https://www.baeldung.com/linux/decoding-encoded-urls. Accessed 4 grudz 2025

14. Importing data via mongoimport and dates - Working with Data / Developer Tools - MongoDB Community Hub. https://www.mongodb.com/community/forums/t/importing-data-via-mongoimport-and-dates/117242/6. Accessed 5 grudz 2025

15. Best Practices for Data Modeling in MongoDB - Database Manual - MongoDB Docs. https://www.mongodb.com/docs/manual/data-modeling/best-practices/#std-label-embedding-vs-references. Accessed 5 grudz 2025

16. Explore Your Data with Playgrounds - VS Code Extension - MongoDB Docs. https://www.mongodb.com/docs/mongodb-vscode/playgrounds/. Accessed 5 grudz 2025

17. $function (expression operator) - Database Manual - MongoDB Docs. https://www.mongodb.com/docs/manual/reference/operator/aggregation/function/#mongodb-expression-exp.-function. Accessed 5 grudz 2025

18. (2024) How to update a collection item without removing non existing values send on the query? - Working with Data / Node.js Frameworks - MongoDB Community Hub. https://www.mongodb.com/community/forums/t/how-to-update-a-collection-item-without-removing-non-existing-values-send-on-the-query/267405. Accessed 5 grudz 2025

19. Documents - Database Manual - MongoDB Docs. https://www.mongodb.com/docs/manual/core/document/#std-label-document-dot-notation. Accessed 5 grudz 2025

20. MongoDB Auto-Increment | MongoDB. https://www.mongodb.com/resources/products/platform/mongodb-auto-increment. Accessed 5 grudz 2025

21. Analyze Query Performance - Database Manual - MongoDB Docs. https://www.mongodb.com/docs/manual/tutorial/evaluate-operation-performance/. Accessed 6 grudz 2025

22. Single Field Indexes - Database Manual - MongoDB Docs. https://www.mongodb.com/docs/manual/core/indexes/index-types/index-single/. Accessed 6 grudz 2025

23. Multikey Indexes - Database Manual - MongoDB Docs. https://www.mongodb.com/docs/manual/core/indexes/index-types/index-multikey/. Accessed 6 grudz 2025

24. Compound Indexes - Database Manual - MongoDB Docs. https://www.mongodb.com/docs/manual/core/indexes/index-types/index-compound/. Accessed 6 grudz 2025

25. The ESR (Equality, Sort, Range) Guideline - Database Manual - MongoDB Docs. https://www.mongodb.com/docs/manual/tutorial/equality-sort-range-guideline/#std-label-esr-indexing-guideline. Accessed 6 grudz 2025

26. Wildcard Indexes - Database Manual - MongoDB Docs. https://www.mongodb.com/docs/manual/core/indexes/index-types/index-wildcard/. Accessed 6 grudz 2025

27. Production Considerations - Database Manual - MongoDB Docs. https://www.mongodb.com/docs/manual/core/transactions-production-consideration/. Accessed 7 grudz 2025

28. TIOGO EC (2021) Create a replica set in MongoDB with Docker Compose. https://blog.tericcabrel.com/mongodb-replica-set-docker-compose/. Accessed 7 grudz 2025

29. Networking | Docker Docs. https://docs.docker.com/compose/how-tos/networking/. Accessed 7 grudz 2025

30. Cannot connect to replica set via mongo compass - Ops and Admin / Replication - MongoDB Community Hub. https://www.mongodb.com/community/forums/t/cannot-connect-to-replica-set-via-mongo-compass/165856. Accessed 7 grudz 2025

31. Read Concern - Database Manual - MongoDB Docs. https://www.mongodb.com/docs/manual/reference/read-concern/. Accessed 7 grudz 2025

32. Write Concern - Database Manual - MongoDB Docs. https://www.mongodb.com/docs/manual/reference/write-concern/. Accessed 7 grudz 2025

33. Transactions - Database Manual - MongoDB Docs. https://www.mongodb.com/docs/manual/core/transactions/. Accessed 7 grudz 2025

34. Using MongoDB As Graph Database: Use Cases | MongoDB. https://www.mongodb.com/resources/basics/databases/mongodb-graph-database. Accessed 7 grudz 2025

35. Clustered Collections - Database Manual - MongoDB Docs. https://www.mongodb.com/docs/manual/core/clustered-collections/. Accessed 7 grudz 2025

36. Map-Reduce - Database Manual - MongoDB Docs. https://www.mongodb.com/docs/manual/core/map-reduce/. Accessed 7 grudz 2025

37. Database Triggers - Atlas App Services - MongoDB Docs. https://www.mongodb.com/docs/atlas/app-services/triggers/database-triggers/. Accessed 7 grudz 2025

38. Change Streams - Database Manual - MongoDB Docs. https://www.mongodb.com/docs/manual/changeStreams/. Accessed 7 grudz 2025

Zastosowane skróty

Compass MongoDB Compass
JSON JavaScript Object Notation
MTG Magic: The Gathering
SQL Structured Query Language