Zaawansowane Systemy Baz Danych – Etap 3 „MongoDB”

Michał Ankiersztajn 311171

## Zbiór Danych

Wybieram zbiór danych z <https://apidocs.cheapshark.com/>, skorzystam tam z danych Deals i Shops.

Dlaczego? Ponieważ:

* Jest to publiczne API, w którego USAGE wchodzi jedynie to, aby zakup gier odbywał się poprzez linki, które są dostarczane przez cheapshark
* Pobierane dane są w formacie JSON
* Choć nie mają zagnieżdżeń to łatwo je wprowadzić
* Deals mają odnośniki do StoreID, co pozwoli mi spełnić wymaganie referencji

W razie gdyby zbiór okazał się zbyt mały/zbyt mało skomplikowany API udostępnia też Games z których będę mógł skorzystać, aby dodać większą liczbę referencji i zagnieżdżeń. Początkowo nie korzystam z Games, ponieważ wymagało by to zrobienia paru tysięcy requestów poprzez API, które ma wprowadzony rate limiting.

Wezmę 3150 za pomocą skryptu, który jest dostępny w załączonych plikach.

## Konfiguracja MongoDB i wczytanie danych

Wcześniej stworzone dane nie zawierają zagnieżdżeń, więc sam je dodam modyfikując plik json za pomocą skryptu **Nestify.py**, który jest w załączniku.

Bazę skonfiguruję poprzez dockera. Plik **docker-compose.yaml** w załącznikach. Buduję i łączę się z nim kolejno poprzez:

**docker-compose pull**

**docker-compose build**

**docker-compose up**

**docker exec -it 19a0035f6b4a mongosh -u admin -p admin**

Wczytanie danych i stworzenie kolekcji:

Obraz zawierający tekst, zrzut ekranu, Czcionka, Jaskrawoniebieski

Opis wygenerowany automatycznie

Korzystanie z dokumentów poprzez zapytania:

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

Teraz połączę tak, aby Deal zawierał również dane z stores:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

Bardziej złożone zapytania pokażę przy logice bizensowej w kolejnym kroku.

Niestety zauważyłem to dopiero podczas pisania logiki biznesowej, ale:

* dealID to kopia \_id, więc je usuwam:

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

* storeID to kopia \_id, więc również je usuwam z tym, że tutaj update’uje id w kolekcji deals (było to dosyć skomplikowane jak na pierwsze spotkanie z MongoDB):



Obraz zawierający tekst, zrzut ekranu, wyświetlacz, oprogramowanie

Opis wygenerowany automatycznie

Początkowo błędnie zmigrowałem dane i przypisałem cały sklep do storeID zamiast samego \_id, na szczęście dało się to dosyć prosto odkręcić (w końcu posiadałem już storeID w wymaganym obiekcie):



Niestety nie mogę pokazać wyniku (zostawiłem print w migracji i zaśmieciłem terminal), ale mogę pokazać rezultat:

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

Teraz usuwam storeID z stores:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

## Logika biznesowa

Wszystkie funkcje znajdują się również w pliku **functions.js**

* Dodawanie okazji:

Mockowy deal tworzę chatemGPT: <https://chatgpt.com/share/6738d502-dac8-8011-a64a-0f5b4a82b4d2>

Funkcja + test:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

* Wyszukiwanie najlepszych okazji (takich, które mają najwyższy dealRating):

Obraz zawierający tekst, zrzut ekranu, Czcionka, design

Opis wygenerowany automatycznie

Wynik uruchomienia (kolejne deale mają coraz niższy rating):

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

Na screenie wyżej 9.1, a niżej 9.0

* Wyszukiwanie okazji po nazwie gry:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Rezultat szukania po „When ski”:

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

Dalsze testy…

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

* Wyszukiwanie okazji po filtrze na max salePrice

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

* Zaawansowane wyszukiwanie uwzględniające max cenę, to że gra jest na przecenie, tytuł gry, większy lub równy steam rating oraz sortujące po najlepszym dealRating i połączone z danymi sklepu:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Test:

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

* Usuwanie deali

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Usuwam deal z poprzedniego advancedSearch i testuję 2-krotną próbę usunięcia tego samego deala:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

* Wyszukiwanie okazji po wysokiej ocenie Steam
* Wyszukiwanie okazji po wysokiej ocenie Metacritic
* Dodawanie sklepów

Co ciekawe, można teoretycznie pisać kod w SQL i transformować go na MongoDB działające na JSONie. Dodatkowo praca z zagnieżdżeniami jest bardzo wygodna, bo mamy możliwość wprowadzenia dodatkowego poziomu abstrakcji ułatwiającego działanie na danych.

## Indeksy

Początkowo sprawdzam, jakie już istnieją indeksy:

Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie

Zostawiam je, ponieważ \_id jest często wyszukiwanym polem. Dodatkowo potrzebuję dodać indeksy dla kolekcji „deals” na game.title, steam.steamRatingPercent, pricing.salePrice, storeID:

Dla testów dodaję funkcję, która 10\_000 razy odpala daną komendę dla lepszej statystyki:

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Wiem, że istnieje ‘explan()’, ale mam problem z jego długością, a nie udało mi się sprawić, aby scroll w terminalu działał. Teoretycznie mógłbym tworzyć pliki z explain i potem je analizować jako .txt, ale byłoby to bardzo czasochłonne, jak na takie jednorazowe zadanie.

Najpierw testuję bez dodatkowych indeksów:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Dodanie indeksu na game.title:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Co ciekawe, pogoszyło to czas dla szukania po nazwie, ale polepszyło w przypadku zaawansowanego szukania, sprawdźmy jeszcze jak działa indeks na „text”:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Zadziałał on gorzej niż zwyczajny indeks, być może wynika to z tego, że indeks ten jest bardziej zoptymalizowany dla pojedynczych słów niż zdań.

Dodanie indeksu na steam.steamRatingPercent:

Obraz zawierający zrzut ekranu, Czcionka, tekst

Opis wygenerowany automatycznie

Lekko polepszył on czas wyszukiwania.

Dodanie indeksu na storeID:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Czas wyszukiwania nieznacznie się poprawił

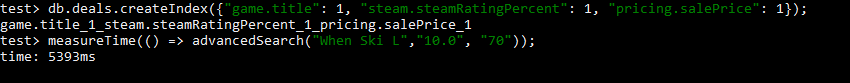
Dodanie na pricing.salePrice:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Przy prostym szukaniu po samej cenie wynik się poprawł, ale przy zaawansowanym z większą liczbą filtrów – pogorszył się.

Sprawdźmy działanie indeksu kompozytowego zamiast pojedynczych dla game.title, pricing.salePrice i steam.steamRatingPercent:



Rozwiązanie to okazało się nieznacznie gorsze niż, gdy występują pojedyncze indeksy.

Baza danych mogła być zbyt mała, aby w pełni wiarygodnie ocenić wydajność indeksów. Nie występowały w niej miliony dokumentów, a jedynie parę tyś.

Indeksy w mongoDB mają kierunek, który może nawet zwolnić działanie query, dlatego należy korzystać z niego w odpowiedni sposób, ma to znaczenie zwłaszcza dla indeksów kompozytowych.

MongoDB ma prościej nazwane indeksy np. TEXT zamiast GIN i GIST, co na wysokim poziomie abstrakcji jest wygodne, ale na niskim i specjalistyczne może nas ugryźć, bo musimy znać algorytmy pod spodem zamiast czytać ich nazwy

## Transakcje

W mongodb występują transakcje:

* Jednodokumentowe:

Gwarancja operacji atomowej na 1 dokumencie. Albo zmiany zostaną zaaplikowane albo cofnięte.

* Wielodokumentowe:

Jest to transakcja na wielu dokumentach na raz, mongoDB traktuje to tak, jakby wykonywana była ona jako jedna atomowa jednostka pracy z tym, że jest ona nie na 1, a wielu dokumentach.

Przykład dla wprowadzania nowego deala (chcemy atomowo sprawdzić stores i wpisać dane do deals):

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

Dla lepszego testu chciałem zrobić trigger na wpisywanie deals, ale MongoDB nie posiada takiej funkcjonalności i trzeba by zewnętrznie monitorować zmiany, aby móc coś takiego wprowadzić.

* Rozproszone:

Niezależnie od tego, czy dane są przechowywane w przetwarzanym klastrze, czy innym spełniania jest zasada transakcyjności ACID.

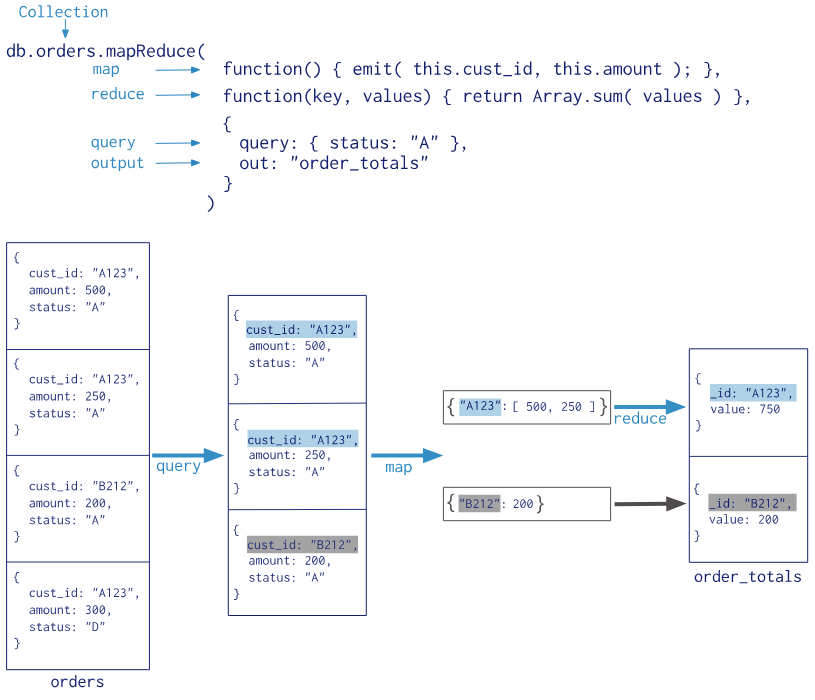
Dodatkowo w ramach transakcji jesteśmy w stanie tworzyć dodatkowe kolekcje, jak i indeksy.

Transakcje powiązane są z sesjami – można mieć max 1 transakcję na sesję i jeśli sesja się skończy to transakcja zostanie przerwana.

## 3 Zaawansowane funkcjonalności

1. MapReduce

Działa to tak, jak na poniższym diagramie:



W ramach kolekcji robimy query -> następnie mapujemy nasze wartości i finalnie dokonujemy redukcji, która jest zwracana jako inny rodzaj dokumentu. Zaprezentuję to na przykładzie, który będzie liczył średnią ceny danej gry:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Rezultat:

Obraz zawierający zrzut ekranu, tekst, obwód

Opis wygenerowany automatycznie

Weryfikacja poprawności cen:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, diagram

Opis wygenerowany automatycznie

Alternatywnie można korzystać z agregacji i jest to raczej preferowany sposób wykorzystywania mapReduce. Dodatkowo tworzę one nową kolekcję.

mapReduce jest dosyć wygodny i czytelny, choć rozumiem czemu stał się ‘deprecated’ na rzecz agregate, które moim zdaniem jest bardziej czytelne. Dużym plusem mapReduce jest możliwość korzystania z funkcji, możemy napisać jedną bardziej skomplikowaną funkcję i korzystać z niej w wielu miejscach. Przykładowo moja funkcja do liczenia średniej mogłaby być zastosowana w również do wyliczania średniej ceny bez przeceny, a nawet w innych bazach danych.

1. Zapytania ad-hoc

Już wcześniej z nich korzystałem, jednak co one oznaczają?

Zapytania ad-hoc pozwalają developerom na przekazywanie zmiennych do zapytania. Oznacza to, że pełne zapytanie znane jest tylko w czasie wykonania (brzmi to jak coś co może spowodować błędy, ale dopiero na produkcji?). Przykład z którego korzystałem już wcześniej:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

query przyjmuje zmienną regex i na jej podstawie porównuje tytuły gier. Wynik testów jest dosyć długi przedstawiony w sekcji 3. Wyszukiwanie okazji po nazwie gry

Ich wadą jest to, że są znane w czasie wykonania i możemy przekazać niekompatybilne typy lub wykonać jakieś niemożliwe działania jak np. dzielenie przez 0.

Dużą zaletą jest ich elastyczność i to, że jesteśmy w stanie przetwarzać dane imperatywnie oraz cache’ować przykładowo często używane Regexy, których budowa jest kosztowna.

1. Capped collections

Są to kolekcje, które nie pozwalają na przekroczenie podanej liczby bajtów. Można z nich korzystać do przykładowo przechowywania dodatkowych logów w bazie. W ten sposób mamy dostęp do dodatkowych informacji potrzebnych przy monitoringu, szukaniu błędów, czy skrajnych sytuacjach nie zapychając w pełni całej bazy. Minusem takiego podejścia jest to, że w pewnym momencie tracimy przeszłe logi. W takich sytuacjach warto skorzystać z TTL, które może usuwać dane po upływie danego czasu (ogólnie indeksy TTL są zalecany bardziej niż capped collections, jednak wszystko zależy od naszej sytuacji).

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Wrzucenie 5 elementu powinno nadpisać 1 w tym przypadku z „id 1”:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Pierwsza wpisana wartość została usunięta, a nowo dodana wpisana na końcu kolekcji.

Widzę wartość capped collections w środowiskach, gdzie płacimy za to ile danych przechowujemy lub korzystanie z TTL jest kosztowne, jednak w większości wypadków użycie TTL będzie po prostu lepsze, ponieważ unikniemy sytuacji w których nasze dane będą usuwane przedwcześnie – gdy możemy jeszcze ich potrzebować.

## Wnioski

* MongoDB przetwarza wszystkie zapytania SQLowe jednak w bardziej „Javascriptowy” sposób.
* Pisanie kodu w MongoDB zajmuje dłużej niż w SQLu(może to wynikać z doświadczenia), ale jest bardziej czytelne
* Dużo łatwiej jest wczytać i migrować zmienne w kolecjach
* Łatwo jest wczytać zbiór danych (wystarczy nam JSON, który jest zwracany przez większość API)
* Zauważyłem, że działa wolniej, jednak może to być jedynie wrażenie, tu jest inna problematyka niż przy bazach SQLowych.
* Brak wsparcia Triggerów w MongoDB

## Bibliografia

<https://www.baeldung.com/linux/mongodb-as-docker-container>

<https://www.mongodb.com/docs/manual/core/databases-and-collections/>

<https://stackoverflow.com/questions/8866041/how-can-i-list-all-collections-in-the-mongodb-shell>

<https://www.mongodb.com/docs/manual/tutorial/project-fields-from-query-results/>

<https://stackoverflow.com/questions/33156703/whats-faster-find-limit1-or-findone-in-mongodb-mongoose>

<https://stackoverflow.com/questions/6851933/how-to-remove-a-field-completely-from-a-mongodb-document>

<https://www.mongodb.com/docs/manual/tutorial/project-fields-from-query-results/#return-the-specified-fields-and-the-_id-field-only>

<https://stackoverflow.com/questions/4481635/in-mongo-how-do-i-display-the-indexes-of-a-collection>

<https://www.mongodb.com/blog/post/performance-best-practices-indexing>

<https://www.mongodb.com/docs/manual/reference/method/db.collection.createIndex/>

<https://stackoverflow.com/questions/14021605/mongodb-how-can-i-see-the-execution-time-for-the-aggregate-command>

<https://stackoverflow.com/questions/10329104/why-does-direction-of-index-matter-in-mongodb>

<https://www.mongodb.com/docs/manual/core/map-reduce/>

<https://www.mongodb.com/docs/manual/reference/command/mapReduce/>

<https://www.mongodb.com/docs/manual/core/capped-collections/>