Zaawansowane Systemy Baz Danych – Etap 3 „MongoDB”

Michał Ankiersztajn 311171

1. 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.

2. 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:

```
PS C:\Users\Michał\Desktop\ZSBD\Etap 3 - Mongo> docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                      NAMES
06cf8a8b60ce   mongo:latest   "docker-entrypoint.s…"   10 seconds ago   Up 8 seconds    0.0.0.0:27017->27017/tcp   etap3-mongo-mongo-1
PS C:\Users\Michał\Desktop\ZSBD\Etap 3 - Mongo> docker cp nested_deals.json 06cf8a8b60ce:/deals.json
PS C:\Users\Michał\Desktop\ZSBD\Etap 3 - Mongo> docker cp stores.json 06cf8a8b60ce:/stores.json
root@06cf8a8b60ce:/# mongoimport --collection='deals' --file='deals.json' -u admin -p admin --authenticationDatabase admin --jsonArray
2024-11-15T13:50:19.709+0000    error parsing command line options: unknown option "authenticationDatabase"
2024-11-15T13:50:19.709+0000    try 'mongoimport --help' for more information
root@06cf8a8b60ce:/#
root@06cf8a8b60ce:/# mongoimport --collection='deals' --file='deals.json' -u admin -p admin --authenticationDatabase admin --jsonArray
2024-11-15T13:51:08.267+0000    connected to: mongodb://localhost/
2024-11-15T13:51:08.401+0000    3000 document(s) imported successfully. 0 document(s) failed to import.
root@06cf8a8b60ce:/# mongoimport --collection='stores' --file='stores.json' -u admin -p admin --authenticationDatabase admin --jsonArray
2024-11-15T13:51:23.116+0000    connected to: mongodb://localhost/
2024-11-15T13:51:23.137+0000    35 document(s) imported successfully. 0 document(s) failed to import.
root@06cf8a8b60ce:/#
```

Korzystanie z dokumentów poprzez zapytania:

```
test> db.getCollectionNames();
[ 'deals', 'stores' ]
test> db.deals.findOne();
{
  _id: ObjectId('673751cc02805c6855ff12ac'),
  steam: {
    steamRatingText: 'Very Positive',
    steamRatingPercent: '87',
    steamRatingCount: '490',
    steamAppID: '638000'
  },
  metacritic: {
    metacriticLink: '/game/when-ski-lifts-go-wrong/',
    metacriticScore: '80'
  },
  pricing: {
    salePrice: '1.28',
    normalPrice: '14.99',
    isOnSale: '1',
    savings: '91.460974'
  },
  game: {
    internalName: 'WHENSKILIFTSGOWRONG',
    title: 'When Ski Lifts Go Wrong',
    gameID: '194665'
  },
  dealID: 'icVa7vS43yj2VW2ZTPVRrpVBr9mtcvSlVpwhHSohEDs%3D',
  storeID: '3',
  releaseDate: 1548201600,
  lastChange: 1731580694,
  dealRating: '9.5',
  thumb: 'https://shared.fastly.steamstatic.com/store_item_assets/steam/apps/638000/capsule_sm_120.jpg?t=1668073718'
}
test> db.stores.findOne();
{
  _id: ObjectId('673751db13a9b185c17dd9ca'),
  storeID: '2',
  storeName: 'GamersGate',
  isActive: 1,
  images: {
    banner: '/img/stores/banners/1.png',
    logo: '/img/stores/logos/1.png',
    icon: '/img/stores/icons/1.png'
  }
}
```

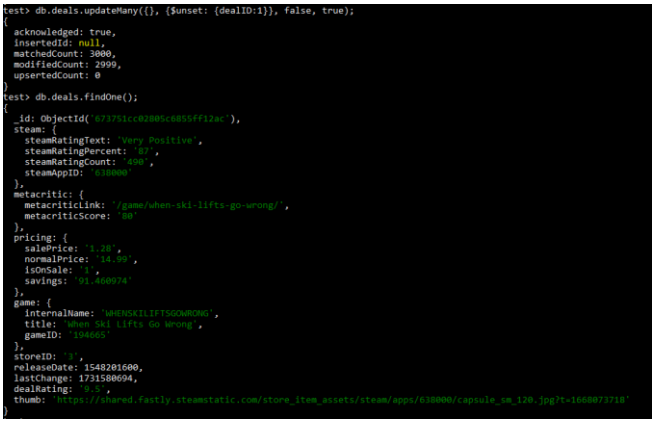
Teraz połączę tak, aby Deal zawierał również dane z stores:

```
test> db.deals.aggregate([ { $lookup: { from: "stores", localField: "storeID", foreignField: "storeID", as: "store" } }, { $limit: 1 } ])
[
  {
    _id: ObjectId('673751cc02805c6855ff12ac'),
    steam: {
      steamRatingText: 'Very Positive',
      steamRatingPercent: '87',
      steamRatingCount: '490',
      steamAppID: '638000'
    },
    metacritic: {
      metacriticLink: '/game/when-ski-lifts-go-wrong/',
      metacriticScore: '80'
    },
    pricing: {
      salePrice: '1.28',
      normalPrice: '14.99',
      isOnSale: '1',
      savings: '91.460974'
    },
    game: {
      internalName: 'WHENSKILIFTSGOWRONG',
      title: 'When Ski Lifts Go Wrong',
      gameID: '194665'
    },
    dealID: 'icVa7vS43yj2VW2ZTPVRrpVBr9mtcvSlVpwhHSohEDs%3D',
    storeID: '3',
    releaseDate: 1548201600,
    lastChange: 1731580694,
    dealRating: '9.5',
    thumb: 'https://shared.fastly.steamstatic.com/store_item_assets/steam/apps/638000/capsule_sm_120.jpg?t=1668073718',
    store: [
      {
        _id: ObjectId('673751db13a9b185c17dd9cc'),
        storeID: '3',
        storeName: 'GreenManGaming',
        isActive: 1,
        images: {
          banner: '/img/stores/banners/2.png',
          logo: '/img/stores/logos/2.png',
          icon: '/img/stores/icons/2.png'
        }
      }
    ]
  }
]
```

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:



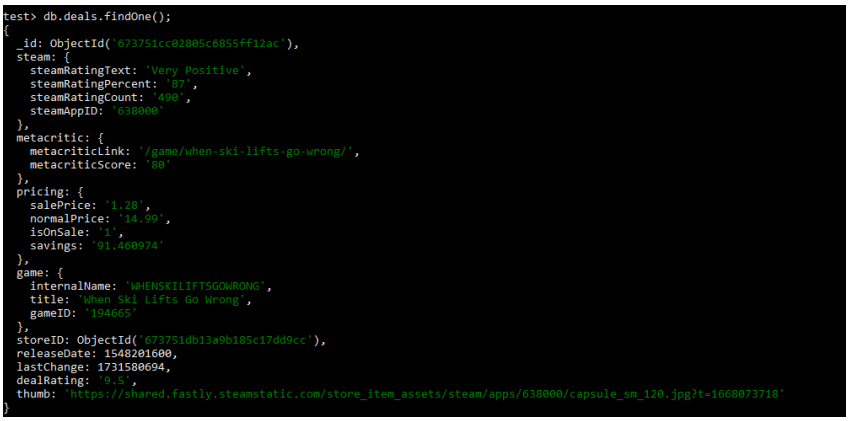
- storeID to kopia _id, więc również je usuwam z tym, że tutaj update'uje id w kolekcji deals (było to dosyć skomplikowane jak na pierwsze spotkanie z MongoDB):

```
test> db.deals.find().forEach((deal) => { const storeId = db.stores.findOne({ storeID: deal.storeID }); db.deals.updateOne( { _id: deal._id }, { $set: { storeID: storeId } } ); });
```

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):

```
test> db.deals.find().forEach((deal) => { const store = db.stores.findOne({ _id: ObjectId(deal.storeID._id) }); print(store); db.deals.updateOne( { _id: deal._id }, { $set: { storeID: store._id } } ); });
```

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



Teraz usuwam storeID z stores:

```
test> db.stores.updateMany({}, {$unset: {storeID:1}}, false, true);
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 35,
  modifiedCount: 35,
  upsertedCount: 0
}
test> db.stores.findOne();
{
  _id: ObjectId('673751db13a9b185c17dd9ca'),
  storeName: 'GamersGate',
  isActive: 1,
  images: {
    banner: '/img/stores/banners/1.png',
    logo: '/img/stores/logos/1.png',
    icon: '/img/stores/icons/1.png'
  }
}
```

3. 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:

```
test> function insertDeal(deal) {
...     const storeExists = db.stores.findOne({ _id: deal.storeID });
...     if (!storeExists) {
...         throw new Error("Store with given ID doesn't exist");
...     }
...     const result = db.deals.insertOne(deal);
...     return result.insertedId;
... }
[Function: insertDeal]
test> insertDeal(exampleDealWrongShop);
Error: Store with given ID doesn't exist
test> insertDeal(exampleDeal);
ObjectId('6738d66090f44df5b1fe6916')
```

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

```
test> function getBestDeals() {
...     return db.deals.aggregate([{
...         $lookup: {
...             from: "stores",
...             localField: "storeID",
...             foreignField: "_id",
...             as: "store"
...         }
...     }, {
...         $sort: {
...             dealRating: -1
...         }
...     }])
... }
[Function: getBestDeals]
```

Wynik uruchomienia (kolejne deale mają coraz niższy rating):

```
  },
  storeID: ObjectId('673751db13a9b185c17dd9cc'),
  releaseDate: 1599696000,
  lastChange: 1731580245,
  dealRating: '9.1',
  thumb: 'https://shared.fastly.steamstatic.com/store_item_assets/steam/apps/609920/capsule_sm_120.jpg?t=1728915008',
  store: [
    {
      _id: ObjectId('673751db13a9b185c17dd9cc'),
      storeName: 'GreenManGaming',
      isActive: 1,
      images: {
        banner: '/img/stores/banners/2.png',
        logo: '/img/stores/logos/2.png',
        icon: '/img/stores/icons/2.png'
      }
    }
  ]
 },
 {
  _id: ObjectId('673751cc02805c6855ff1307'),
  steam: {
    steamRatingText: 'Very Positive',
    steamRatingPercent: '83',
    steamRatingCount: '327',
    steamAppID: '246960'
  },
  metacritic: {
    metacriticLink: '/game/giana-sisters-twisted-dreams-rise-of-the-owlver/',
    metacriticScore: '84'
  },
  pricing: {
    salePrice: '0.99',
    normalPrice: '4.99',
    isOnSale: '1',
    savings: '80.160321'
  },
  game: {
    internalName: 'GIANASISTERSTWISTEDDREAMSRISEOFTHEOWLVERLORD',
    title: 'Giana Sisters: Twisted Dreams - Rise of the Owlverlord',
    gameID: '101631'
  },
  storeID: ObjectId('673751db13a9b185c17dd9d3'),
  releaseDate: 1380153600,
  lastChange: 1731613911,
  dealRating: '9.0',
  thumb: 'https://shared.fastly.steamstatic.com/store_item_assets/steam/apps/246960/capsule_sm_120.jpg?t=1729076649',
  store: [
    {
      _id: ObjectId('673751db13a9b185c17dd9d3'),
      storeName: 'Humble Store',
      isActive: 1,
      images: {
        banner: '/img/stores/banners/10.png',
        logo: '/img/stores/logos/10.png',
        icon: '/img/stores/icons/10.png'
      }
    }
  ]
 }
]
Type "it" for more
test>
```

Na screenie wyżej 9.1, a niżej 9.0

- Wyszukiwanie okazji po nazwie gry:

```
function findGameDeals(title) {
    const regex = new RegExp(title, 'i');

    const deals = db.deals.find(
     { "game.title": { $regex: regex } },
     { pricing: 1, storeID: 1, dealRating: 1, game: 1 }
    );

    return deals;
}
```

Rezultat szukania po „When ski”:

Dalsze testy...

```
test> findGameDeals("Shadow of mordor");

test> findGameDeals("Mordor");

test>
```

- Wyszukiwanie okazji po filtrze na max salePrice

```
function findGamesCheaperThan(maxSalePrice) {
    return db.deals.find(
      { "pricing.salePrice": { $lte: maxSalePrice } },
      { pricing: 1, storeID: 1, dealRating: 1, game: 1 }
    );
}
```

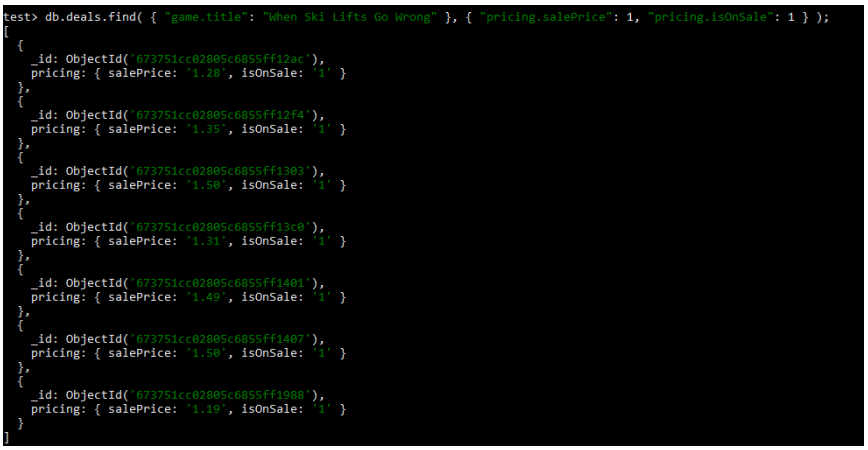
```
test> findGamesCheaperThan("0.01");
[
  {
    _id: ObjectId('673751cc02805c6855ff1650'),
    pricing: {
      salePrice: '0.00',
      normalPrice: '0.99',
      isOnSale: '1',
      savings: '100.000000'
    },
    game: {
      internalName: 'LYNNHDWALLPAPER',
      title: 'Lynn , HD WallPaper',
      gameID: '269806'
    },
    storeID: ObjectId('673751db13a9b185c17dd9cf'),
    dealRating: '7.7'
  }
]
```

- 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:

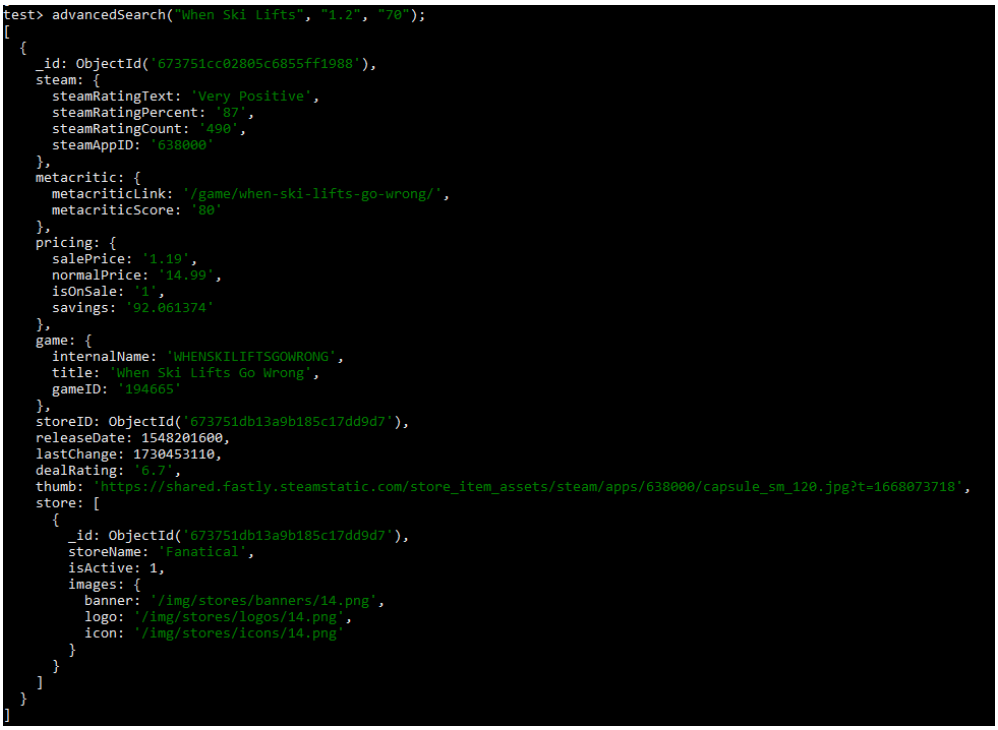
```
function advancedSearch(title, maxSalePrice, minSteamRating) {
    const regex = new RegExp(title, 'i');

    return db.deals.aggregate([{
        $lookup: {
            from: "stores",
            localField: "storeID",
            foreignField: "_id",
            as: "store"
        }
    }, {
        $sort: {
            dealRating: -1
        }
    }, {
        $match: {
            "game.title": { $regex: regex },
            "pricing.salePrice": { $lte: maxSalePrice },
            "steam.steamRatingPercent": { $gte: minSteamRating },
            "pricing.isOnSale": "1"
        }
    }])
}
```

Test:



```
test> db.deals.find( { "game.title": "When Ski Lifts Go Wrong" }, { "pricing.salePrice": 1, "pricing.isOnSale": 1 } );
[
  {
    _id: ObjectId('673751cc02805c6855ff12ac'),
    pricing: { salePrice: '1.28', isOnSale: '1' }
  },
  {
    _id: ObjectId('673751cc02805c6855ff12f4'),
    pricing: { salePrice: '1.35', isOnSale: '1' }
  },
  {
    _id: ObjectId('673751cc02805c6855ff1303'),
    pricing: { salePrice: '1.50', isOnSale: '1' }
  },
  {
    _id: ObjectId('673751cc02805c6855ff13c0'),
    pricing: { salePrice: '1.31', isOnSale: '1' }
  },
  {
    _id: ObjectId('673751cc02805c6855ff1401'),
    pricing: { salePrice: '1.49', isOnSale: '1' }
  },
  {
    _id: ObjectId('673751cc02805c6855ff1407'),
    pricing: { salePrice: '1.50', isOnSale: '1' }
  },
  {
    _id: ObjectId('673751cc02805c6855ff1988'),
    pricing: { salePrice: '1.19', isOnSale: '1' }
  }
]
```



```
test> advancedSearch("When Ski Lifts", "1.2", "70");
[
  {
    _id: ObjectId('673751cc02805c6855ff1988'),
    steam: {
      steamRatingText: 'Very Positive',
      steamRatingPercent: '87',
      steamRatingCount: '490',
      steamAppID: '638000'
    },
    metacritic: {
      metacriticLink: '/game/when-ski-lifts-go-wrong/',
      metacriticScore: '80'
    },
    pricing: {
      salePrice: '1.19',
      normalPrice: '14.99',
      isOnSale: '1',
      savings: '92.061374'
    },
    game: {
      internalName: 'WHENSKILIFTSGOWRONG',
      title: 'When Ski Lifts Go Wrong',
      gameID: '194665'
    },
    storeID: ObjectId('673751db13a9b185c17dd9d7'),
    releaseDate: 1548201600,
    lastChange: 1730453110,
    dealRating: '6.7',
    thumb: 'https://shared.fastly.steamstatic.com/store_item_assets/steam/apps/638000/capsule_sm_120.jpg?t=1668073718',
    store: [
      {
        _id: ObjectId('673751db13a9b185c17dd9d7'),
        storeName: 'Fanatical',
        isActive: 1,
        images: {
          banner: '/img/stores/banners/14.png',
          logo: '/img/stores/logos/14.png',
          icon: '/img/stores/icons/14.png'
        }
      }
    ]
  }
]
```

- Usuwanie deali

```
function deleteDeal(dealID) {
    const objectID = new ObjectId(dealID);
    const result = db.deals.deleteOne({ _id: objectID });
    if (result.deletedCount == 1) {
        print("Deal deleted successfully")
    } else {
        throw Error("Deal with given id doesn't exist");
    }
}
```

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

```
test> deleteDeal("673751cc02805c6855ff1988");
Deal deleted successfully

test> advancedSearch("When Ski Lifts", "1.2", "70");

test> deleteDeal("673751cc02805c6855ff1988");
Error: Deal with given id doesn't exist
```

- 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.

4. Indeksy

Początkowo sprawdzam, jakie już istnieją indeksy:

```
test> db.deals.getIndexes();
[ { v: 2, key: { _id: 1 }, name: '_id_' } ]
test> db.stores.getIndexes();
[ { v: 2, key: { _id: 1 }, name: '_id_' } ]
```

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:

```
function measureTime(command) {
    const t1 = new Date();
    for (let i = 0; i < 10_000; i++) {
        command();
    }
    const t2 = new Date();
    print("time: " + (t2 - t1) + "ms");
}
```

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:

```
test> measureTime(() => getBestDeals());
time: 4519ms

test> measureTime(() => findGameDeals("Shadow of"));
time: 4193ms

test> measureTime(() => findGamesCheaperThan("10.0"));
time: 4134ms

test> measureTime(() => advancedSearch("When Ski L","10.0", "70"));
time: 5295ms
```

Dodanie indeksu na game.title:

```
test> db.deals.createIndex({"game.title":1});
game.title_1
test> measureTime(() => findGameDeals("Shadow of"));
time: 4280ms

test> measureTime(() => advancedSearch("When Ski L","10.0", "70"));
time: 5077ms
```

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

```
test> db.deals.dropIndex("game.title_1")
{ nIndexesWas: 2, ok: 1 }
test> db.deals.createIndex({"game.title": "text"});
game.title_text
test> measureTime(() => findGameDeals("Shadow of"));
time: 4305ms

test> measureTime(() => advancedSearch("When Ski L","10.0", "70"));
time: 5161ms
```

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:

```
test> measureTime(() => advancedSearch("When Ski L","10.0", "70"));
time: 5091ms
```

Lekko polepszył on czas wyszukiwania.

Dodanie indeksu na storeID:

```
test> measureTime(() => advancedSearch("When Ski L","10.0", "70"));
time: 5078ms

test> measureTime(() => getBestDeals());
time: 4392ms
```

Czas wyszukiwania nieznacznie się poprawił

Dodanie na pricing.salePrice:

```
test> measureTime(() => advancedSearch("When Ski L","10.0", "70"));
time: 5242ms

test> measureTime(() => findGamesCheaperThan("10.0"));
time: 4058ms
```

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:

```
test> db.deals.createIndex({"game.title": 1, "steam.steamRatingPercent": 1, "pricing.salePrice": 1});
game.title_1_steam.steamRatingPercent_1_pricing.salePrice_1
test> measureTime(() => advancedSearch("When Ski L","10.0", "70"));
time: 5393ms
```

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

5. 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):

```
function insertDealTransaction(deal) {
    const session = db.getMongo().startSession();
    try {
        session.startTransaction();

        const storeExists = db.stores.findOne({ _id: deal.storeID });
        if (!storeExists) {
            throw new Error("Store with given ID doesn't exist");
        }

        const result = db.deals.insertOne(deal);

        session.commitTransaction();

        return result.insertedId;
    } catch (error) {
        session.abortTransaction();
        throw error;
    } finally {
        session.endSession();
    }
}
```

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.

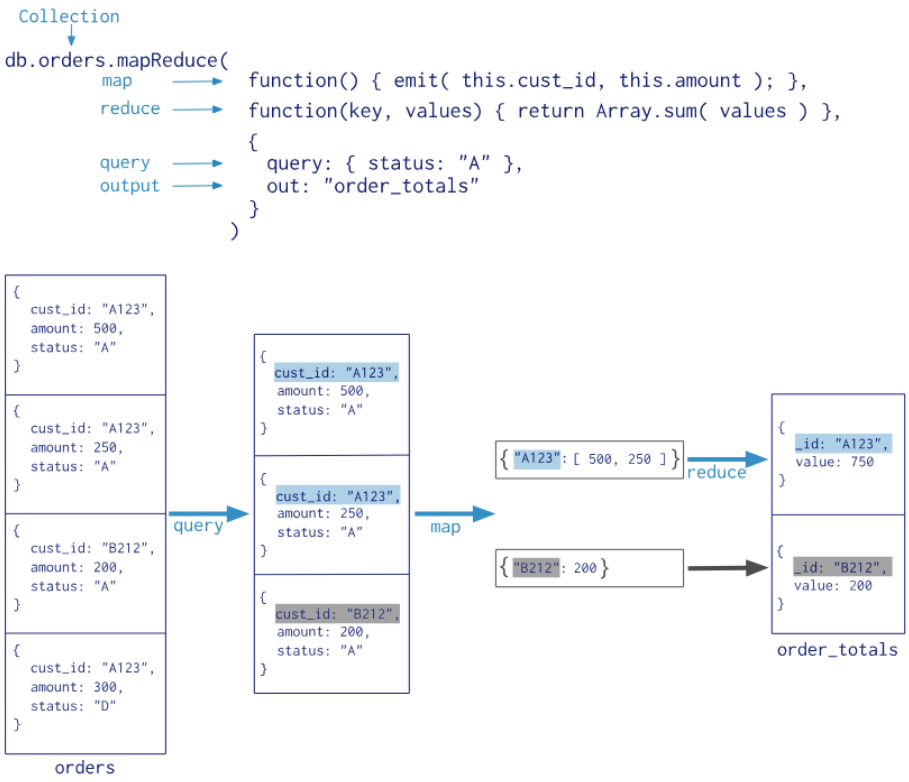
Dodatkowo w ramach transakcji jesteśmy w stanie tworzyć dodatkowe kolekcje, jak i indeksy.

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

6. 3 Zaawansowane funkcjonalności

1. MapReduce

Działa to tak, jak na poniższym diagramie:



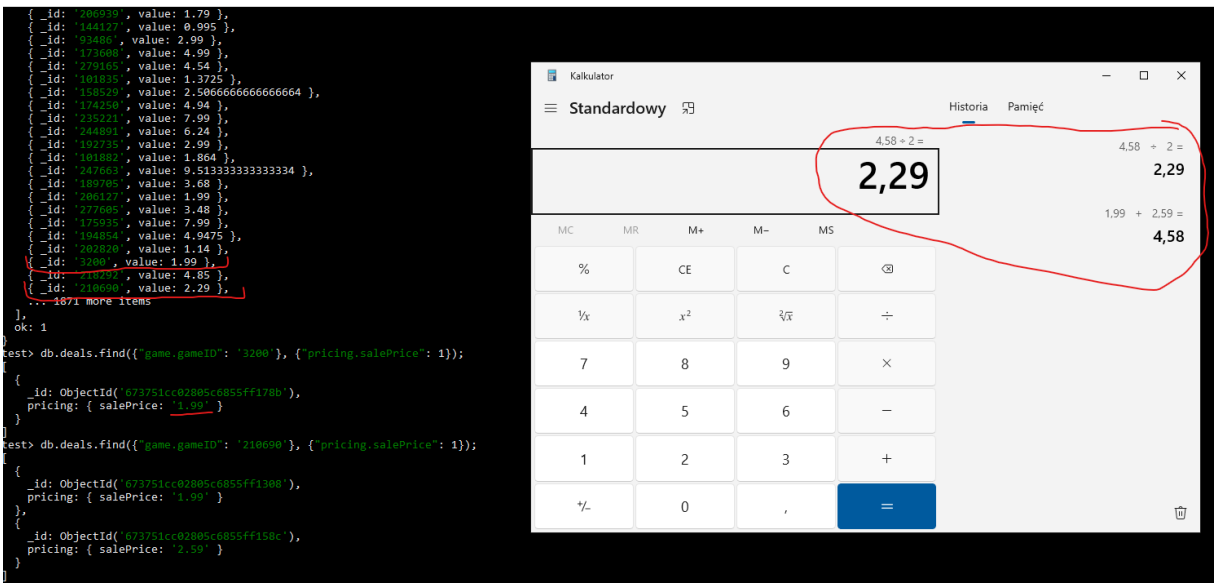
W ramach kolekcji robimy query -> następnie mapujemy nasze wartości i finalnie dokonujemy redukcji, która jest zwracana jako inny rodzaj dokumentu. Zaprezentuję to na przykładzie, który będzie liczył średnią ceny danej gry:

```
// MapReduce
db.deals.mapReduce(
  function () {
    emit(this.game.gameID, parseFloat(this.pricing.salePrice));
  },
  function (key, values) {
    return Array.sum(values) / values.length ;
  },
  { out: { inline: 1 }  }
);
```

Rezultat:

```
    { _id: '228550', value: 9.99 },
    { _id: '225182', value: 1.49 },
    { _id: '235239', value: 9.39 },
    { _id: '223406', value: 1.75 },
    { _id: '143926', value: 5.99 },
    { _id: '246605', value: 4.99 },
    { _id: '172131', value: 4.49 },
    { _id: '178363', value: 1.43 },
    { _id: '145041', value: 5.023333333333333 },
    { _id: '250314', value: 5.85 },
    { _id: '223095', value: 6 },
    { _id: '214331', value: 12.49 },
    { _id: '166739', value: 4.99 },
    { _id: '282112', value: 19.24 },
    { _id: '244127', value: 2.71 },
    { _id: '219702', value: 9.99 },
    { _id: '96278', value: 8.49 },
    { _id: '281527', value: 38.08 },
    { _id: '173008', value: 1.59 },
    { _id: '147369', value: 3.99 },
    { _id: '207441', value: 13.99 },
    { _id: '165395', value: 2.99 },
    { _id: '259818', value: 3.39 },
    { _id: '212038', value: 4.37 },
    { _id: '143037', value: 2.7 },
    { _id: '101913', value: 5.99 },
    { _id: '231074', value: 2.6566666666666667 },
    { _id: '143048', value: 1.49 },
    { _id: '86759', value: 1.9633333333333336 },
    { _id: '259611', value: 9.99 },
    { _id: '197150', value: 2.1466666666666665 },
    { _id: '291789', value: 3.24 },
    { _id: '204130', value: 3.74 },
    { _id: '166145', value: 7.49 },
    { _id: '205390', value: 5.99 },
    { _id: '164622', value: 4.49 },
    { _id: '187818', value: 2.07 },
    { _id: '146451', value: 1.74 },
    { _id: '173642', value: 2.99 },
    { _id: '201951', value: 3.39 },
    ... 1871 more items
  ],
  ok: 1
}
```

Weryfikacja poprawności cen:



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.

2. 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:

```
function findGameDeals(title) {
    const regex = new RegExp(title, 'i');

    const deals = db.deals.find(
     { "game.title": { $regex: regex } },
     { pricing: 1, storeID: 1, dealRating: 1, game: 1 }
    );

    return deals;
}
```

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.

3. 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).

```
test> db.createCollection("logs", { capped: true, size: 10000, max: 4 })
{ ok: 1 }
test> show collections
average_cost_per_game
deals
logs
stores
test> db.logs.insertMany([{ info: "Game with id 1 was removed"}, { info: "Game with id 2 was removed"}, { info: "Game with id 3 was removed"}, { info: "Game with id 4 was removed"}])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('6739adc9d67e816e51fe6913'),
    '1': ObjectId('6739adc9d67e816e51fe6914'),
    '2': ObjectId('6739adc9d67e816e51fe6915'),
    '3': ObjectId('6739adc9d67e816e51fe6916')
  }
}
test> db.logs.find()
[
  {
    _id: ObjectId('6739adc9d67e816e51fe6913'),
    info: 'Game with id 1 was removed'
  },
  {
    _id: ObjectId('6739adc9d67e816e51fe6914'),
    info: 'Game with id 2 was removed'
  },
  {
    _id: ObjectId('6739adc9d67e816e51fe6915'),
    info: 'Game with id 3 was removed'
  },
  {
    _id: ObjectId('6739adc9d67e816e51fe6916'),
    info: 'Game with id 4 was removed'
  }
]
```

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

```
test> db.logs.insertOne({ info: "Game with id 5 was removed" })
{
  acknowledged: true,
  insertedId: ObjectId('6739ae16d67e816e51fe691c')
}
test> db.logs.find()
[
  {
    _id: ObjectId('6739ae0ed67e816e51fe6919'),
    info: 'Game with id 2 was removed'
  },
  {
    _id: ObjectId('6739ae0ed67e816e51fe691a'),
    info: 'Game with id 3 was removed'
  },
  {
    _id: ObjectId('6739ae0ed67e816e51fe691b'),
    info: 'Game with id 4 was removed'
  },
  {
    _id: ObjectId('6739ae16d67e816e51fe691c'),
    info: 'Game with id 5 was removed'
  }
]
```

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.

7. 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

8. 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/