UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu

NoSQL MongoDB

Brazilský Ecommerce

Lukáš Kovář

V Hradci Králové
dne 19. 5. 2025

Obsah

Seminární práce z předmětu NoSQL databáze

Úvod

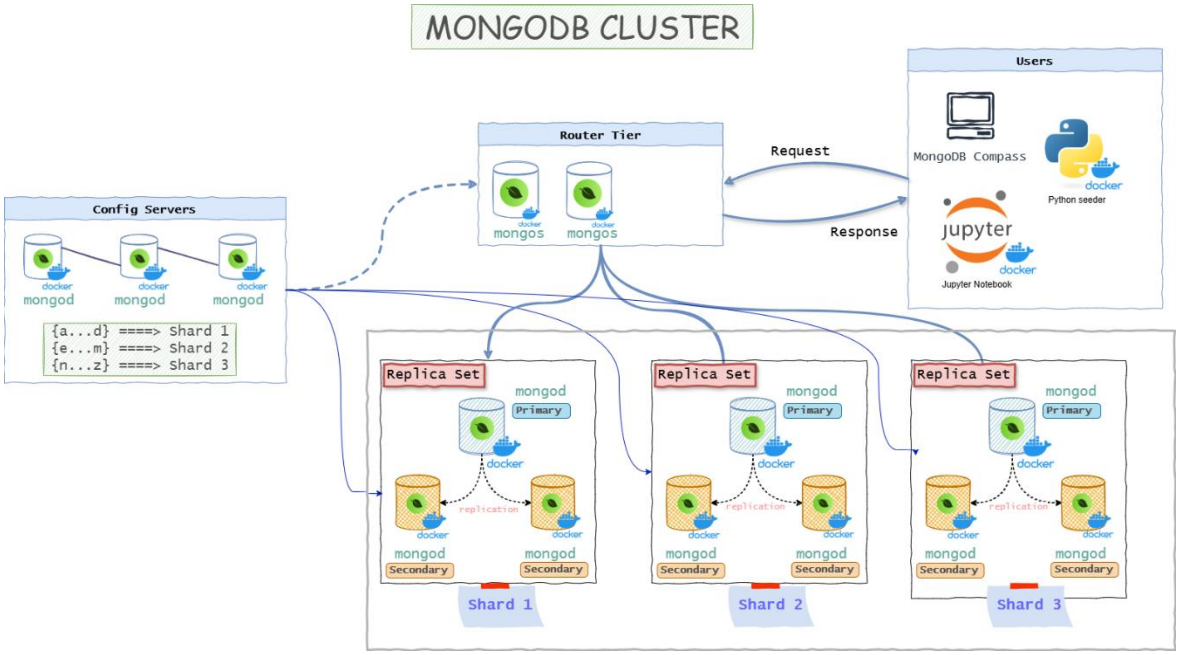
Tématem mé semestrální práce je návrh a implementace NoSQL databáze MongoDB v prostředí e-commerce systému. Cílem bylo vytvořit distribuované a škálovatelné řešení, které umožní efektivní ukládání a analýzu dat o zákaznících, objednávkách a produktech. Kromě samotného nasazení databáze jsem se zaměřil také na zabezpečení, automatizaci a práci s daty pomocí dotazů a Pythonu.

V práci popisuji architekturu MongoDB clusteru vytvořeného pomocí Docker Compose, včetně jednotlivých komponent (config servery, shardy, routery). Dále podrobně rozebírám konfiguraci systému – způsob shardingu, replikace, distribuce dat a zabezpečení pomocí RBAC a keyfile.

Použitá verze databáze je MongoDB 6.0.2, která splňuje požadavek práce s verzí maximálně tři generace zpětně (nejnovější je 8 ale ta způsobovala problémy), s novějšími verzemi jsem měl problémy se změnami týkající nastaveni nodu.

Projekt se zaměřuje čistě na backendové řešení, API rozhraní nebo nasazení do cloudu nejsou jeho součástí, i když by ve skutečném systému měly své místo.

1 Architektura



Obrázek 1 - Schema clusteru – převzato, upraveno [1]

Cluster je implementován pomocí Docker Compose a zahrnuje config servery, shardy a router. Každá komponenta je navržena pro specifickou roli v distribuovaném systému.

1.1 Schéma a popis architektury

Architektura implementovaného MongoDB sharded clusteru je vizualizována na přiloženém schématu. Toto schéma znázorňuje propojení mezi jednotlivými komponentami a tok dat v systému.

1.2 Specifika konfigurace

Projekt implementuje plně funkční MongoDB sharded cluster pomocí Docker Compose. Architektura obsahuje:

- Config Server Replica Set (3 uzly)
- 3 Shardy, každý implementován jako replica set (3 uzly každý a,b,c)
- 2 Routery (mongos) pro směrování dotazů

1.2.1 CAP teorém

V naší implementaci MongoDB clusteru jsme se zaměřili na CP (Consistency + Partition Tolerance) garance Brewerova CAP teorému. Toto rozhodnutí **je** učiněno s ohledem na povahu e-commerce aplikace, kde je kriticky důležitá konzistence dat, zejména u objednávek a jejich stavů.

1.2.2 Cluster

Pro naše řešení jsme zvolili jeden cluster s názvem "ecommerce"

Proč jeden cluster?

Jednoduchost správy: Jeden cluster je jednodušší na správu a monitoring

Efektivní distribuce zátěže: Routery automaticky distribuují zátěž mezi shardy

Centralizovaná konfigurace: Config servery udržují konzistentní konfiguraci celého clusteru

1.2.3 Uzly

Cluster obsahuje 11 uzlů, které jsou rozděleny následovně:

- **Routery (2):**
  - router01 (port 27117)
  - router02 (port 27118)
- **Config servery (3):**
  - configsvr01 (port 27119)
  - configsvr02 (port 27120)
  - configsvr03 (port 27121)
- **Shardy (3), každý s 3 replikami:**
  - Shard 1: shard01-a, shard01-b, shard01-c
  - Shard 2: shard02-a, shard02-b, shard02-c
  - Shard 3: shard03-a, shard03-b, shard03-c

1.2.4 Sharding

Máme Shardy 1,2,3 a jejich záložní repliky a,b,c.

Sharding je implementován s 3 shardy, každý jako samostatný replica set. Toto umožňuje horizontální škálování dat napříč více servery a zlepšuje výkon pro velké datasety.

1.2.5 Replikace

Každý shard i config server jsou implementovány jako replica sety s primary/secondary modelem. Toto zajišťuje vysokou dostupnost a odolnost proti výpadkům jednotlivých uzlů.

1.2.6 Persistence dat

Data jsou ukládána na persistentní Docker volumes (`/data/db`, `/data/configdb`).

MongoDB používá write-ahead log (WAL journal) pro zajištění perzistence i při výpadku.

Primární paměť (RAM) slouží pro cache, sekundární (disk) pro trvalé uložení. V clusteru toto nijak nekonfigurujeme, je použito defaultní nastavení.

Případné zálohy by se následně měly dělat na jiný stroj, např do cloudu na S3.

1.2.7 Distribuce dat

Používáme sharding pro distribuci dat. Klíčové pro distribuci jsou:

- Sharding key – určuje, jak jsou data rozdělena mezi shardy
- Router – Node, který zajišťuje rovnoměrné rozdělení dat na základě sharding keys

```
// Povolení shardingu pro databázi
sh.enableSharding("ecommerce")

// Nastavení shardování pro kolekce
sh.shardCollection("ecommerce.orders", { "_id": "hashed" });
sh.shardCollection("ecommerce.products", { "_id": "hashed" });
sh.shardCollection("ecommerce.customers", { "_id": "hashed" });
```

Obrázek 2 - Sharding klíče se nastavují ve scriptu create-indexes-and-shards.js

Po konfiguraci a nahrání dat je clusteru stav:

```
use("ecommerce");
// Distribuce dat zakazniku
db.customers.getShardDistribution()

// Výstup
{
      "Shard     rs-shard-03     at     rs-shard-03/shard03-a:27017,shard03-
b:27017,shard03-c:27017": {
    "data": "7.69MiB",
    "docs": 33157,
    "chunks": 2,
    "estimated data per chunk": "3.84MiB",
    "estimated docs per chunk": 16578
  },
      "Shard     rs-shard-01     at     rs-shard-01/shard01-a:27017,shard01-
b:27017,shard01-c:27017": {
    "data": "7.69MiB",
    "docs": 33145,
    "chunks": 2,
    "estimated data per chunk": "3.84MiB",
    "estimated docs per chunk": 16572
  },
      "Shard     rs-shard-02     at     rs-shard-02/shard02-a:27017,shard02-
b:27017,shard02-c:27017": {
    "data": "7.69MiB",
```

```
    "docs": 33139,
    "chunks": 2,
    "estimated data per chunk": "3.84MiB",
    "estimated docs per chunk": 16569
  },
  "Totals": {
    "data": "23.07MiB",
    "docs": 99441,
    "chunks": 6,
    "Shard rs-shard-03": [
      "33.34 % data",
      "33.34 % docs in cluster",
      "243B avg obj size on shard"
    ],
    "Shard rs-shard-01": [
      "33.33 % data",
      "33.33 % docs in cluster",
      "243B avg obj size on shard"
    ],
    "Shard rs-shard-02": [
      "33.32 % data",
      "33.32 % docs in cluster",
      "243B avg obj size on shard"
    ]
  }
}
```


Obrázek 3 - Stav distribuce dat po nahrání dat

1.2.8 Zabezpečení

Cluster je zabezpečen několika vrstvami autorizace:

1. **Sdílený keyfile pro vnitřní autentizaci**:

```
FROM mongo:6.0.2
COPY auth/mongodb-keyfile /data
RUN chmod 400 /data/mongodb-keyfile
RUN chown 999:999 /data/mongodb-keyfile
```

Obrázek 4 - Dockerfile s keyfile

   - Všechny uzly clusteru používají sdílený keyfile pro ověření své identity vůči ostatním členům clusteru
   - Tento keyfile je distribuován na všechny config servery, shardy a routery
   - Zajišťuje, že pouze autorizované instance MongoDB se mohou připojit k replica setu

2. **Role-based Access Control (RBAC)**:

   - Implementován systém uživatelů s různými úrovněmi oprávnění
   - Uživatelé jsou vázáni na konkrétní databázi, kde mají oprávnění

   1**. Administrativní uživatel** – **admin_lukas** s heslem 123 (pro produkci doporučeno silnější) v databázi admin. Role root poskytuje plný přístup ke všem databázím a funkcím clusteru. Používá se pro administraci, monitorování, údržbu a vytváření nových uživatelů.

   2. **Aplikační uživatel s právy pro zápis** – **ecommerce_user** s heslem ecommerce123 v databázi ecommerce. Role readWrite umožňuje čtení a zápis pouze v databázi ecommerce. Používán pro běžné operace aplikace.

   3. **Uživatel pouze pro čtení** – **ecommerce_reader** s heslem reader123 v databázi ecommerce. Role read umožňuje pouze čtení dat z databáze ecommerce. Vhodný pro reporty, analýzy a dashboard aplikace.

   Samotná inicializace uživatelů probíhá v rámci procesu clusteru pomocí skriptu auth.sh. Nejprve je vytvořen administrátorský účet, kterým jsou pak vytvořeni aplikační uživatelé. Tento přístup zajišťuje princip nejnižších potřebných oprávnění a odděluje administrátorské a aplikační funkce.

2 Funkční řešení

2.1 Struktura

Hlavní adresářová struktura obsahuje:

- docker-compose.yml - konfigurace Docker kontejnerů
- Makefile – skripty pro správu clusteru
- scripts/ - skripty pro inicializaci a správu clusteru
- auth/ - keyfile
- Dockerfile – základní Docker image pro MongoDB (rozšířený o keyfile)
- seeder.Dockerfile - Docker image pro seeder (python image s balicky pro vkladani dat do MongoDB)

```
services:

  ## Router
  router01:
    build:
      context: .
    container_name: router-01
          command:    mongos    --port    27017    --configdb    rs-config-
server/configsvr01:27017,configsvr02:27017,configsvr03:27017 --bind_ip_all
--keyFile /data/mongodb-keyfile
    ports:
      - 27117:27017
    restart: always
    volumes:
      - ./scripts:/scripts
      - mongodb_cluster_router01_db:/data/db
      - mongodb_cluster_router01_config:/data/configdb
…

  ## Config Servers
  configsvr01:
    build:
      context: .
    container_name: mongo-config-01
    command: mongod --port 27017 --configsvr --replSet rs-config-server --
keyFile /data/mongodb-keyfile
    volumes:
      - ./scripts:/scripts
      - mongodb_cluster_configsvr01_db:/data/db
      - mongodb_cluster_configsvr01_config:/data/configdb
    ports:
      - 27119:27017
    restart: always
```

```
    links:
      - shard01-a
      - shard02-a
      - shard03-a
      - configsvr02
      - configsvr03

...

  shard01-a:
    build:
      context: .
    container_name: shard-01-node-a
    command: mongod --port 27017 --shardsvr --replSet rs-shard-01 --keyFile
/data/mongodb-keyfile
    volumes:
      - ./scripts:/scripts
      - mongodb_cluster_shard01_a_db:/data/db
      - mongodb_cluster_shard01_a_config:/data/configdb
    ports:
      - 27122:27017
    restart: always
    links:
      - shard01-b
      - shard01-c

volumes:
  mongodb_cluster_router01_db:
...
```

Obrázek 5 - Náhled souboru docker-compose.yml

2.2 Instalace

K jednoduchému spouštění byl vytvořen Makefile, kde lze volat bud sloučené skupiny příkazů nebo příkaz **make setup**, který celý cluster nastaví včetně inicializace shardu, vytvoření indexů, partision keys a nahrání dat.

```
# MongoDB cluster management inicializace
setup:
    $(MAKE) docker-down-clean
    $(MAKE) docker-up; \
    sleep 5; \
    $(MAKE) docker-init-configserver; \
    sleep 5; \
    $(MAKE) docker-init-shard01; \
    sleep 1; \
    $(MAKE) docker-init-shard02; \
    sleep 1; \
```

```
    $(MAKE) docker-init-shard03; \
    sleep 1; \
    $(MAKE) docker-init-router; \
    sleep 5; \
    $(MAKE) docker-auth; \
    sleep 3; \
    $(MAKE) docker-create-indexes-and-shards
    $(MAKE) docker-insert-data

# v originálním souboru jsou následně skupiny příkazů
…
```

K používání je třeba mít nainstalovaný Docker a Openssl pro vygenerování keyfile.

Instalace byla testována na virtuálním stroji i v prostředí WSL (Linux subsystem pro Windows).

3 Případy užití a případové studie

Tato kapitola popisuje vhodné případy užití MongoDB, konkrétní důvody volby této databáze v projektu a případové studie z praxe.

3.1 Pro jaké účely je MongoDB vhodná?

- Práce s velkými objemy nestrukturovaných nebo semi-strukturovaných dat (např. JSON, dokumenty)
- Horizontální škálování (sharding) při vysokých nárocích na zápis/čtení
- Systémy s požadavkem na vysokou dostupnost a odolnost (replikace)
- Rychlý vývoj (schema-less model, snadná změna struktury dat)
- Analytické systémy, logování, e-commerce, IoT, real-time aplikace

3.2 Proč byla MongoDB zvolena pro tento projekt?

Popularita, z daných databází má nejsilnější komunitu. Znamená i nejvíce dokumentace.

Dataset obsahuje velké množství záznamů s různorodou strukturou (e-commerce data), je zde potenciál zpřetrhat nějaké M to N vazby do vnořených dokumentů (v mém případě order_items).

3.3 Proč nebyla zvolena jiná NoSQL databáze?

Redis – primárně pro caching, složitější pro analytická data

Cassandra – vhodná pro time-series, méně flexibilní pro složité dotazy

Elasticsearch – výborný pro fulltext, ne pro transakční/analytická data

MongoDB je pro tento typ dat (e-commerce, dokumentová struktura, analytika) z vybraných NoSQL databází nejjednodušší na použití.

3.4 Případové studie

3.4.1 Ebay, vyhledávání návrhů

Společnost eBay čelila výzvě efektivního škálování databáze pro poskytování rychlých a relevantních návrhů vyhledávání uživatelům. Tradiční relační databáze nebyly schopny splnit požadavek na dobu odezvy pod 60–70 milisekund. Proto eBay implementovala MongoDB, dokumentově orientovanou databázi, která umožnila dosažení průměrné doby odezvy pod 1,4 milisekundy. [2]

eBay strukturovala seznam návrhů vyhledávání jako dokumenty v MongoDB, které byly indexovány podle předpony slova a doplňkových metadat, jako je kategorie produktu. Díky vícenásobným indexům bylo možné rychle a flexibilně vyhledávat návrhy. Data byla uložena v paměti RAM, což dále zrychlilo přístup. Použití jednoho replikačního setu eliminovalo potřebu shardingu. [2]

Tato implementace umožnila eBay poskytovat uživatelům rychlé a relevantní návrhy vyhledávání, čímž se zvýšila uživatelská spokojenost a efektivita vyhledávání. Případová studie eBay demonstruje, jak lze využít MongoDB pro řešení problémů s výkonem a škálovatelností v prostředí s velkým objemem dat.

3.4.2 Vývoj databázové architektury ve společnosti Discord

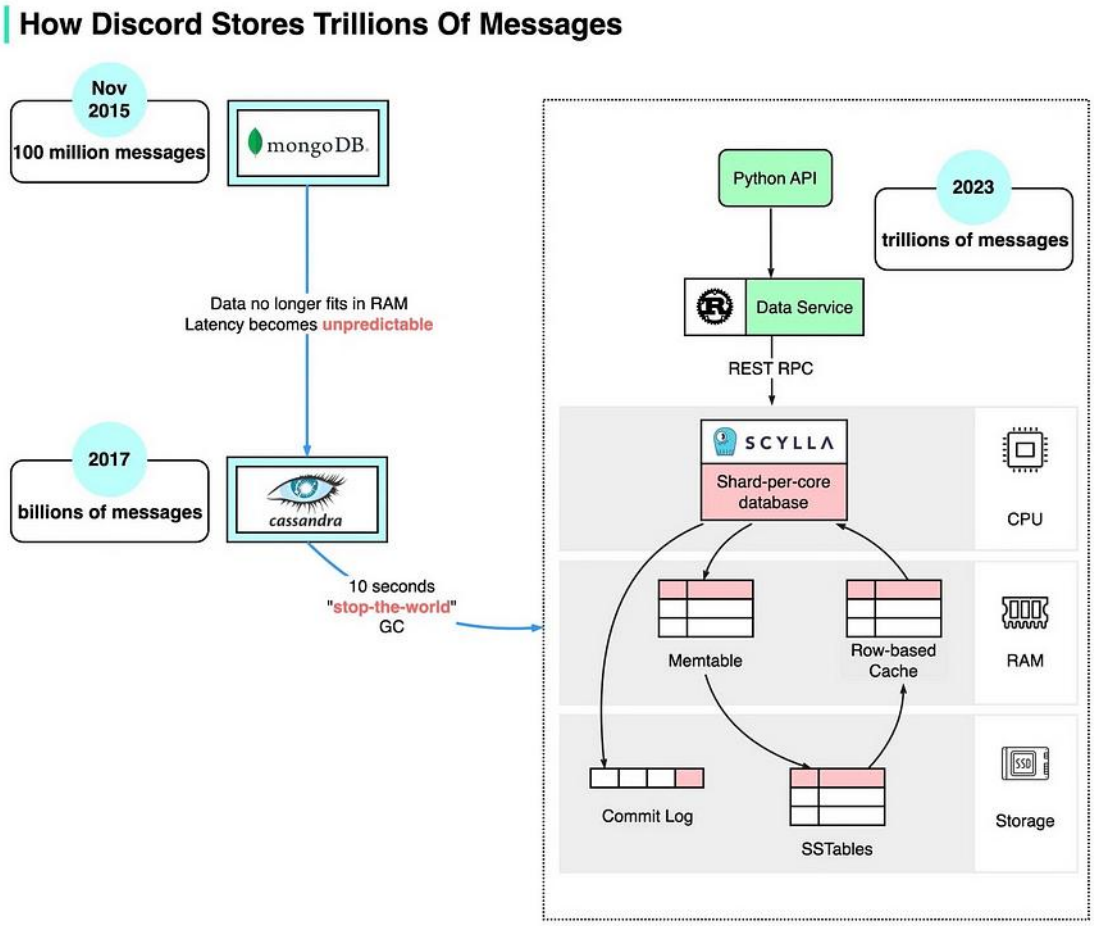
Společnost Discord je příkladem rychle rostoucé technologické firmy, která musela v průběhu let několikrát zásadně přepracovat svou databázovou infrastrukturu kvůli rostoucím požadavkům na škálovatelnost a výkon. V tomto vývoji sehrály roli tři klíčové technologie – MongoDB, Apache Cassandra a ScyllaDB. [3]

Počáteční řešení: MongoDB

V počátcích, okolo roku 2015, Discord využíval MongoDB pro ukládání zpráv. MongoDB poskytovalo jednoduché rozhraní a rychlý vývoj, což odpovídalo tehdejším potřebám startupu. S růstem počtu uživatelů však nastaly problémy: jakmile objem dat přesáhl velikost operační paměti, došlo ke zpomalení systému kvůli přístupům na disk. MongoDB nebylo schopno efektivně škálovat s přibývajícím množstvím zpráv a uživatelů.

Přechod na Apache Cassandra

Aby firma vyřešila problémy s výkonem, přešla na distribuovanou databázi Apache Cassandra, která lépe zvládala zápisy a horizontální škálování. Cassandra umožnila Discordu ukládat miliardy zpráv a provoz v clusterech o desítkách uzlů. I přesto se začaly objevovat nové výzvy, zejména složitá správa databáze, latence způsobená LSM strukturou a náročné operace jako kompakce SSTable souborů. Tyto faktory omezovaly schopnost systému spolehlivě reagovat v reálném čase.



Obrázek 6 - Diagram postupné změny databází a struktur v čase [3]

Optimalizace pomocí ScyllaDB

V roce 2022 Discord migroval z Cassandry na ScyllaDB – binárně kompatibilní databázi, přepsanou v C++ s důrazem na výkon. Tato změna vedla ke snížení latencí a potřebného počtu uzlů (ze 177 na 72), čímž se zjednodušila správa a zvýšila efektivita systému. ScyllaDB také umožnila lépe předvídat výkon a vyrovnat se s požadavky miliardových objemů zpráv bez degradace odezvy. [3]

Tato případová studie ukazuje, jak zásadní roli hraje správná volba databázové technologie při budování rozsáhlých distribuovaných systémů. Discord se svým vývojem stal významným příkladem postupného překonávání omezení NoSQL řešení na cestě k vysoké škálovatelnosti a robustnosti. Na druhou stranu, toto je příklad extrémní zátěže, který drtivá většina systémů řešit nemusí. MongoDB je stále velmi dobrá databáze.

3.4.3 L'Oréal – Zrychlení vývoje a výkonu aplikací s MongoDB

Společnost L'Oréal, globální lídr v oblasti kosmetiky, čelila výzvám spojeným s výkonem a udržovatelností svých digitálních aplikací. Při hledání řešení se rozhodla pro přechod na MongoDB Atlas, plně spravovanou cloudovou databázovou platformu. [4]

Během fáze proof-of-concept zaznamenala společnost výrazné zlepšení: latence databázových operací se snížila téměř 40násobně. Tato změna vedla k jednodušší údržbě backendového kódu, lepší škálovatelnosti a vyšší efektivitě vývoje. Vývojový tým ocenil také funkce MongoDB Atlas, jako je automatické vytváření indexů a simulátor agregací, které přispěly k optimalizaci výkonu aplikací. [4]

Studie demonstruje, jak může přechod na moderní NoSQL databázové řešení, jako je MongoDB Atlas, přinést významné výhody v oblasti **výkonu**, **škálovatelnosti** a **vývojové efektivity**, což je klíčové pro globální společnosti operující v digitálním prostředí.

4 Výhody a nevýhody

4.1 Výhody/Nevýhody obecné

MongoDB přináší několik výhod oproti klasickým relačním databázím:

- Horizontální škálování (sharding) – vhodné pro velké objemy dat
- Vysoká dostupnost díky replikaci
- Flexibilní schéma (schema-less)
- Rychlé dotazy díky indexům
- Snadná integrace s moderními jazyky a nástroji (Python, BI, Jupyter)

Díky schema-less přístupu nemusíme řešit zámky v databázi v momentě přidávání sloupečků a dalších změn schémat. Hlídat je třeba pouze validaci dat, a to buď na aplikační úrovni nebo validačními schématy v Mongodb.

Zároveň má ale i nevýhody:

- Vyšší nároky na hardware při větších clusterech (zejména na pamět)
- Nechodí se pro silně transakční operace
- joiny a relační integrita je omezená

Každá databáze má svoje výhody a nevýhody, avšak MongoDB přináší velmi zajímavý balanc. [5]

4.2 Výhody/Nevýhody v tomto cluster řešení

Výhody

Dobré škálování. Sharding umožňuje vytížení distribuovat mezi více strojů, zde je použit jeden stroj, takže výhody v tomto nejsou. Avšak např pomocí Docker Swarm by tento projekt šel relativně jednoduše rozběhnout v produkčním prostředí.

Povedlo se zpřetrhat M:N vazbu order_items přímo do objednávek, výhodou tak je

Nevýhody

Použitý dataset příliš nečerpá výhod nestrukturovaných dat. Data na e-shopech nejsou příliš často modifikovaná. Zároveň kvůli minimálním zanoření v dokumentech nevyužívá výhod schema-less. Integrita je zde taky důležitá. Pro lepší využití by se hodilo data duplikovat na více míst. Např na zvážení jsou uživatelské reviews. Měli by být zanořené v dokumentu produktu? Nebo mít prvních 10 v dokumentu pro zjednodušení dotazu a zbytek v separátní kolekci? Zároveň některé dotazy které vyžadují aggregaci skrz více shardů co nejsou sharding key nejde realizovat

5 Další specifika

Kolekce užitečných příkazů v Makefile. Přes make docker-setup-data se vymažou kolekce a znovu nahrají schémata a data, pro rychlejší debugging.

Jupyter je součástí docker composu a připojuje se do clusteru.

Seeder python kontejner pro nahrávání dat a validačního schémata po startu clusteru.

Zbytek je standartní.

6 Data

Data v mojí práci pocházejí z veřejně dostupného datasetu "Brazilian E-Commerce Public Dataset by Olist", který je k dispozici na platformě Kaggle. [6]

Vybral jsem si ho z několika důvodů:

1. Obsahuje reálná data z e-commerce platformy
2. Má dostatečný rozsah (více než 5 000 záznamů v každém souboru)
3. Obsahuje různé typy dat (textové, numerické, časové)
4. Data jsou vzájemně propojena, což umožňuje komplexní analýzu a tvorbu zajímavých datových struktur.

6.1 Popis dat

Pro mou databázi jsem si vybral následující datové soubory:

olist_orders_dataset.csv: Obsahuje informace o objednávkách.

olist_order_items_dataset.csv: Detailní položky pro každou objednávku. Tento dataset budu v MongoDB vnořovat do příslušných objednávek.

olist_customers_dataset.csv: Informace o zákaznících.

olist_products_dataset.csv: Informace o produktech (i když nebyl explicitně v zadání jako jeden ze tří, je vhodny pro komplexní pohled skrz M ku N vazbu, protože order_items se na něj odkazuje).

Data z CSV souborů jsou transformována a ukládána do MongoDB ve formátu BSON (Binary JSON) pomoci python scriptu. Každý záznam z CSV souboru (řádek) je typicky převeden na jeden dokument v MongoDB.

6.2 Proč jsem zvolil dané datové struktury (schéma) v rámci MongoDB?

Při návrhu datového modelu pro mou MongoDB databázi **ecommerce** jsem zvažoval několik přístupů k reprezentaci vztahů mezi entitami (objednávky, položky objednávek, zákazníci, produkty). Cílem bylo najít optimální rovnováhu mezi výkonem pro typické dotazy (např. zobrazení celé objednávky), konzistencí dat a jednoduchostí správy, přičemž jsem se řídil doporučenými postupy pro modelování dat v MongoDB. Rozhodl jsem se pro hybridní přístup, který kombinuje vnořování dokumentů a používání referencí.

6.3 Rozsah dat a jejich kvalita

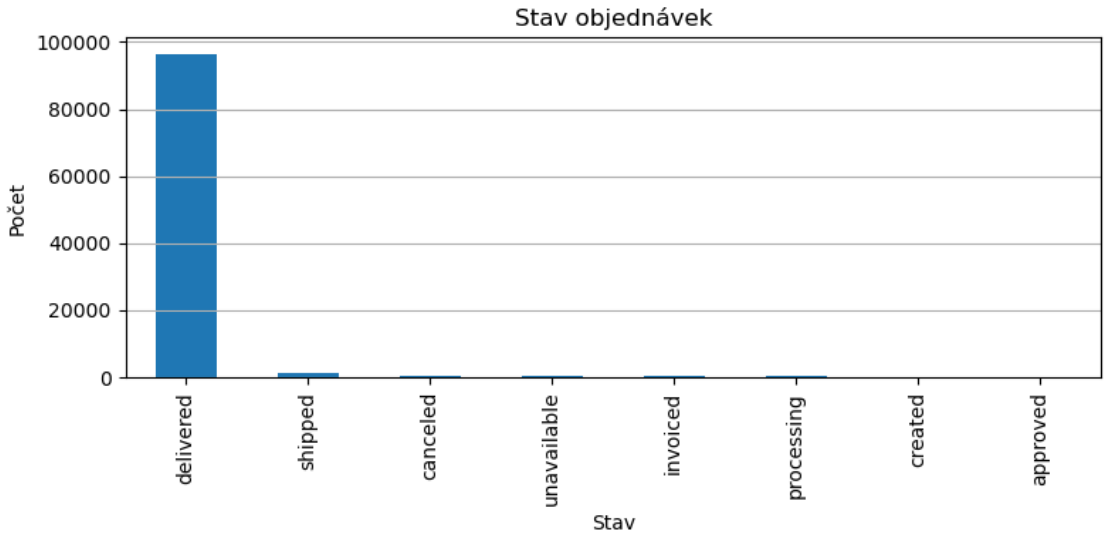
Největší tabulka olist_orders_dataset.csv obsahuje přes **99 tisíc záznamů**. Datové soubory obsahují různý počet řádků:

Dataset	Počet řádků	Počet sloupců
Orders	99 441	8
Order Items	112 650	7
Customers	99 441	5
Products	32 951	9

Chybějící hodnoty byly zejména v souboru **orders.csv**

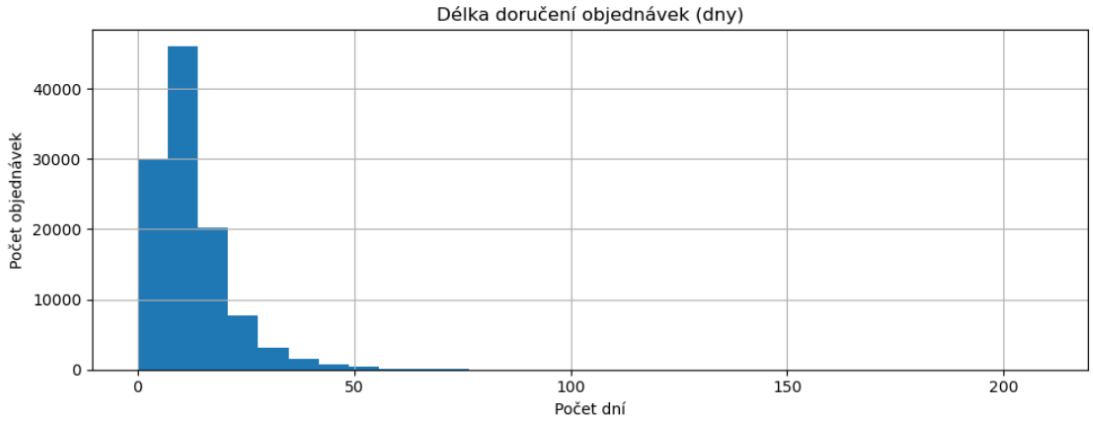
Sloupec	Počet chybějících hodnot
order_delivered_customer_date	2 965
order_delivered_carrier_date	1 783
order_approved_at	160

Některé sloupce obsahují prázdné hodnoty, typicky ve sloupcích s datem potvrzení nebo doručení objednávky – to odpovídá tomu, že některé objednávky nemusí být dokončené.



Obrázek 7 - rozložení typu stavu doručení v orders.csv

Jak můžeme vidět, tak drtivá většina objednávek je ve stavu doručeno. Z 99 441 záznamů mělo 96 % objednávek stav delivered. Ostatní stavy jako shipped, canceled, nebo processing tvoří pouze malé procento celkového objemu.



Obrázek 8 - doba doručení

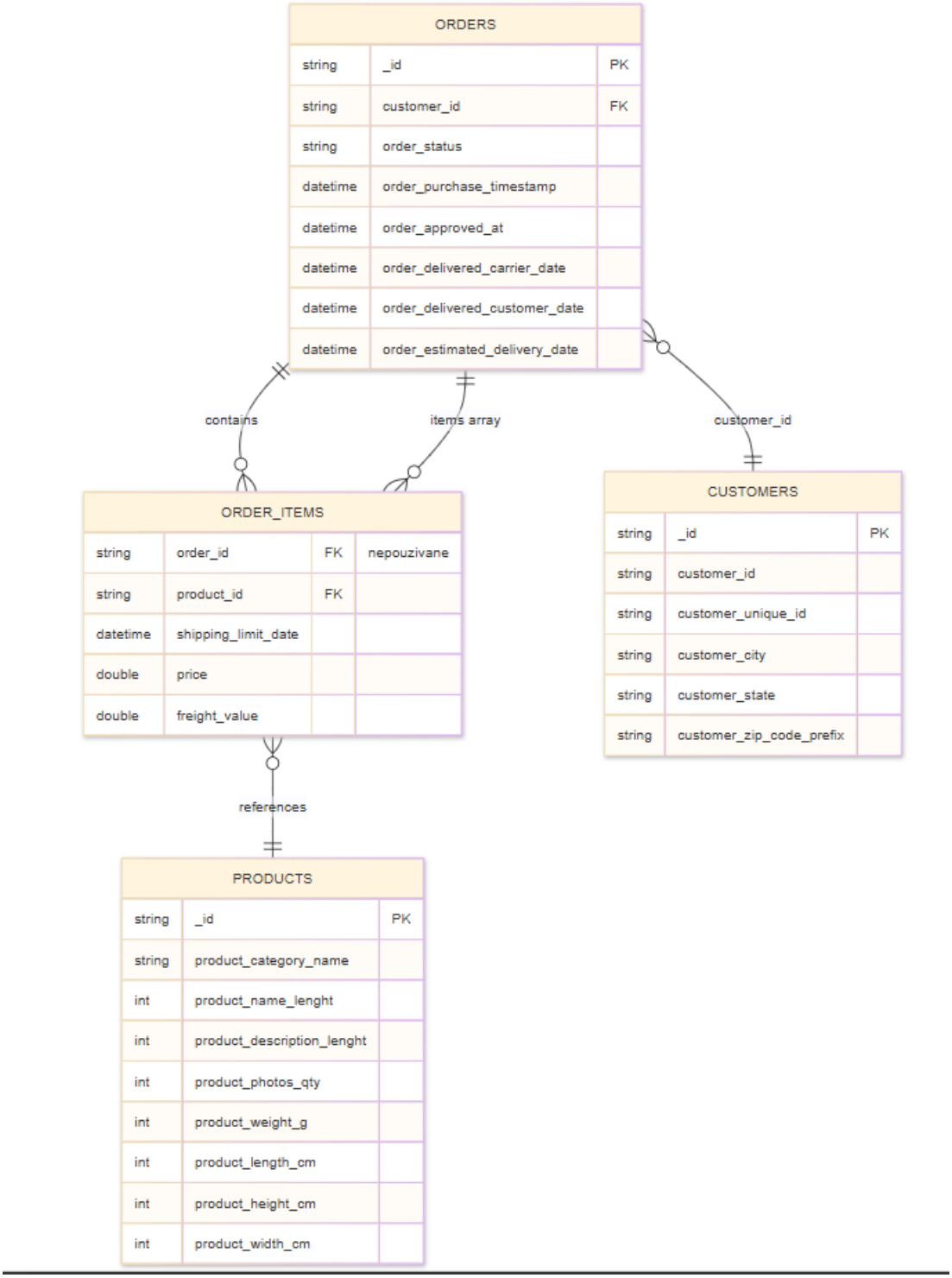
Vypočítal jsem také rozdíl mezi časem nákupu a doručení. Většina objednávek byla doručena do 10 dnů, s typickou hodnotou kolem 8 dní. Další vizualizace lze nalézt v přiloženém jupyter notebooku. Většina zajímavých potencionálních vizualizací by však byla možná udělat v sekci Dotazy.

6.4 Úpravy a čištění dat

Při analýze jsem provedl následující úpravy:

- převod sloupců s daty na správný formát datetime,
- kontrola a odstranění záznamů s příliš velkým množstvím prázdných hodnot,
- zanoření tabulky order_items a orders podle order_id pro další analýzu.

6.5 Schéma dat



Obrázek 9 - schéma struktury kolekcí v databázi – vlastní zpracování

Následující diagram ilustruje strukturu kolekcí v použité databázi MongoDB, včetně vztahů mezi nimi. Vzhledem k tomu, že MongoDB je dokumentově orientovaná databáze, je datový model navržen s využitím (embedding / nested dokumenty), kde je vhodné, a referencování, kde je potřeba udržovat vazby mezi samostatnými kolekcemi.

6.5.1 Struktura kolekcí

- **orders**
  Hlavní kolekce reprezentující objednávky. Každý dokument obsahuje informace o stavu objednávky, časech nákupu, doručení, schválení a také identifikátor zákazníka (customer_id).
  Klíčovým prvkem je pole **items**, které obsahuje **zanořené položky objednávky**. Každá položka v tomto poli obsahuje údaje jako product_id, shipping_limit_date, price a freight_value. Tímto způsobem jsou všechna relevantní data o objednávce uložena pohromadě.

- **customers**
  Samostatná kolekce uchovávající informace o zákaznících, včetně jejich unikátního ID, města a federálního státu. Dokumenty v orders odkazují na tuto kolekci pomocí pole customer_id.

- **products**
  Kolekce s podrobnostmi o produktech, které se mohou vyskytovat v jednotlivých objednávkách. Obsahuje údaje o kategorii, rozměrech, hmotnosti a dalších vlastnostech produktů. Produkty nejsou zavnitřněny v objednávkách, ale jsou **referencovány pomocí product_id** uvnitř položek v poli items.

Vztahy a struktura:

- orders.items je **pole zavnitřněných dokumentů** (embedded array), což je efektivní pro operace, kdy potřebujeme pracovat s celou objednávkou včetně všech jejích položek.

- orders.customer_id → customers._id je **referenční vztah**, sloužící k propojení s detaily o zákazníkovi.

- orders.items.product_id → products._id je **referenční vztah**, díky němuž lze dohledat informace o daném produktu.

Výhody použité struktury:

- Umožňuje snadno získat celou objednávku včetně položek jedním dotazem.

- Vhodné pro systémy, kde jsou položky úzce spojené s objednávkou a nemají samostatný životní cyklus.

- Minimalizuje potřebu složitých lookup operací, protože items jsou rovnou součástí dokumentu objednávky.

7 Dotazy

Veškeré tyto dotazy jsou také v projektu v souboru **dotazy/queries.js**.

7.1 Práce s Daty

V této části jsou příklady základních operací, jako je vkládání produktů a objednávek, úprava atributů (např. stav objednávky nebo město zákazníka), mazání produktů s nulovou hmotností či označení „drahých“ objednávek pomocí podmínky $expr. Tyto dotazy jsou vhodné pro základní správu dat v systému.

```
// přidá dvě nové objednávky
db.orders.insertMany([
    {
        _id: "111111565e28c3e0d756192f84d8731f",
        customer_id: "C100",
        order_status: "created",
        order_purchase_timestamp: new Date(),
        items: [
                        {   product_id:   "8b41fbc2b984a12030090112324d1bc4",
shipping_limit_date: new Date(), price: 199.9, freight_value: 15 }
        ]
    },
    {
        _id: "zzzzzz565e28c3e0d756192f84d8731f",
        customer_id: "C101",
        order_status: "created",
        order_purchase_timestamp: new Date(),
        items: [
                        {   product_id:   "8c92109888e8cdf9d66dc7e463025574",
shipping_limit_date: new Date(), price: 300.9, freight_value: 15 }
        ]
    }
]);


// 2 – přidá 1 položku do existující objednávky
use("ecommerce");
db.orders.updateOne(
    { _id: "bfe42c22ecbf90bc9f35cf591270b6a7" },
    {
        $push: {
            items: {
                product_id: "8b41fbc2b984a12030090112324d1bc4",
                shipping_limit_date: new Date(),
                price: 299.9,
                freight_value: 20
            }
```

```
        }
    }
);

// 3 – změní stav jedné objednávky a vrátí počty ovlivněných dokumentů
use("ecommerce");
db.orders.updateOne(
    { _id: "9d531c565e28c3e0d756192f84d8731f" },
    { $set: { order_status: "delivered", order_delivered_customer_date: new
Date() } },
);

// 4 – přejmenuje město zákazníkovi a vrátí novou verzi dokumentu
use("ecommerce");
db.customers.findOneAndUpdate(
    { _id: "ae8db0691449a44352e7d535ddf78c5e" },
    { $set: { customer_city: "Brno-město" } },
    { returnDocument: "after" }
);

// 5 – hromadně smaže produkty s nulovou váhou
use("ecommerce");
db.products.deleteMany({ product_weight_g: { $lte: 0 } });

// 6 – Označí „drahé“ objednávky nad 1000 příznakem expensive:true
use("ecommerce");
db.orders.updateMany(
    {
        // $expr dovoluje v podmínce použít agregační výrazy.
        // Následně sečteme ceny všech položek v poli items.
        $expr: { $gt: [{ $sum: "$items.price" }, 1000] }   // celková cena
> 1000 Kč
    },
    [
        { $set: { expensive: true } }
    ]
);
```

7.2 Agregační funkce (složitější dotazy)

Agregace v MongoDB umožňují efektivní analýzu dat. Implementoval jsem dotazy jako např.:

- **Top 10 států podle tržeb**
- **Průměrná doba doručení podle státu**
- **Nejdražší objednávka na počet položek**
- **Nejdražší objednávka v každém měsíci**
- **Výpočet marže na položku**

Tyto dotazy využívají operátory jako $group, $lookup, $project, $unwind, $bucketAuto a další. V kombinaci se zákaznickými údaji dokážou vytvořit zajímavé přehledy.

```
// 7 – top 10 států podle tržeb
use("ecommerce");
db.orders.aggregate([
    { $unwind: "$items" },
    // připojíme kolekci customers jelikož na nich jsou státy
    {
        $lookup: {
            from: "customers",
            localField: "customer_id",
            foreignField: "_id",
            as: "c"
        }
    },
    // rozbalí pole customers
    { $unwind: "$c" },
    {
        // seskupí podle státu a spočítá celkovou tržbu
        $group: {
            _id: "$c.customer_state",
            revenue: { $sum: "$items.price" },
            orders: { $addToSet: "$_id" }
        }
    },
    {
        $project: {
            _id: 0,
            state: "$_id",
            revenue: 1,
            orderCount: { $size: "$orders" }
        }
    },
```

```
    { $sort: { revenue: -1 } },
    { $limit: 10 }
]);

// 8 - Celková hodnota objednávek a počet položek na objednávku s detaily
zákazníka
use("ecommerce");
db.orders.aggregate([
    { $unwind: "$items" },
     { $group: { _id: "$_id", customer_id: { $first: "$customer_id" },
totalOrderValue: { $sum: "$items.price" }, totalItems: { $sum: 1 } } },
    { $sort: { totalOrderValue: -1 } },
    { $limit: 10 },
    { $lookup: { from: "customers", localField: "customer_id", foreignField:
"_id", as: "customerDetails" } },
    { $unwind: "$customerDetails" },
    { $project: { _id: 1, totalOrderValue: 1, totalItems: 1, customerCity:
"$customerDetails.customer_city",                          customerState:
"$customerDetails.customer_state" } }
]);

// 9 – průměrná doba doručení (purchase → delivered) podle státu
use("ecommerce");
db.orders.aggregate([
    { $match: { order_delivered_customer_date: { $ne: null } } },
    { $lookup: { from: "customers", localField: "customer_id", foreignField:
"_id", as: "c" } },
    { $unwind: "$c" },
    {
        $project: {
            state: "$c.customer_state",
            days: {
                // prevedeme timestamp na dny
                $divide: [
                          { $subtract: ["$order_delivered_customer_date",
"$order_purchase_timestamp"] },
                    1000 * 60 * 60 * 24
                ]
            }
        }
    },
    { $group: { _id: "$state", avgDays: { $avg: "$days" } } },
    { $sort: { avgDays: 1 } },
    { $limit: 10 }
]);

// 10 - Objednávky, kde VŠECHNY položky byly dodány (tzn.
shipping_limit_date je v minulosti).
```

```
use("ecommerce");
db.orders.find(
    {
        $and: [
            {
                items: {
                    $not: {
                        $elemMatch: {
                            shipping_limit_date: { $gt: new Date() }
                        }
                    }
                }
            },
            // Vyloučí objednávky bez položek
            { items: { $exists: true, $ne: [] } }
        ]
    },
    {
        _id: 1,
        order_status: 1,
        order_purchase_timestamp: 1,
        "items.product_id": 1,
        "items.shipping_limit_date": 1
    }
);

// 11 – nejdražší objednávka vůbec na počet položek v objednávce
use("ecommerce");
db.orders.aggregate([
    {
        // Zajišťuje, že itemCount bude > 0 a vyhneme se dělení nulou při
výpočtu průměru.
        $match: {
            items: { $exists: true, $ne: [], $not: { $size: 0 } },
            order_purchase_timestamp: { $exists: true, $ne: null }
        }
    },
    {
       // Vypočítá totalOrderValue (součet cen položek) a itemCount (počet
položek)
        $addFields: {
            totalOrderValue: { $sum: "$items.price" }, // Sečte ceny všech
položek v poli 'items' daného dokumentu.
             itemCount: { $size: "$items" } // Spočítá počet prvků v poli
'items' daného dokumentu.
        }
    },
    {
```

```
        // Vypočítá průměrnou cenu položky vydělením totalOrderValue počtem
položek.
        $addFields: {
            averageItemPrice: { $divide: ["$totalOrderValue", "$itemCount"]
}
        }
    },
    {
          // Seřadíme objednávky s vypočtenými poli podle průměrné ceny
položky sestupně.
        $sort: { averageItemPrice: -1 }
    },
    {
        // Omezíme výsledek na první dokument, což je objednávka s nejvyšší
průměrnou cenou na položku.
        $limit: 1
    },
    {
         // Přidáme informace o zákazníkovi, spojí nalezenou objednávku s
informacemi o zákazníkovi.
        $lookup: {
            from: "customers",
            localField: "customer_id", // customers._id je cílem reference
z orders.customer_id
            foreignField: "_id",
            as: "customerInfo"
        }
    },
    {
         // preserveNullAndEmptyArrays: true pro případ, že by customer_id
neodkazovalo na existujícího zákazníka (i když by nemělo).
        $unwind: { path: "$customerInfo", preserveNullAndEmptyArrays: true
}
    },
    {
        $project: {
            orderId: "$_id",
               totalOrderValue: { $round: ["$totalOrderValue", 2] }, //
zaokrouhleno na 2 desetiny
            itemCount: "$itemCount",
            averageItemPrice: { $round: ["$averageItemPrice", 2] },
            purchaseTimestamp: 1,
            customerUniqueId: "$customerInfo.customer_unique_id",
            customerCity: "$customerInfo.customer_city",
            customerState: "$customerInfo.customer_state"
        }
    }
]);
```

```
// 12 – nejdražší objednávka v každém měsíci
use("ecommerce");
db.orders.aggregate([
    { $unwind: "$items" },
    {
        $group: {
            _id: { month: { $dateTrunc: { date: "$order_purchase_timestamp",
unit: "month" } }, order: "$_id" },
            total: { $sum: "$items.price" }
        }
    },
    { $sort: { "_id.month": 1, total: -1 } },
    {
        $group: {
            _id: "$_id.month",
            orderId: { $first: "$_id.order" },
            maxTotal: { $first: "$total" }
        }
    },
    { $sort: { "_id.month": 1 } }
]);
```

7.3 Konfigurace / administrace (kolekcí a clusteru)

Zde jsou uvedeny dotazy pro správu clusteru a kolekcí – např. kontrola stavu shardingu (sh.status()), přístup k validačnímu schématu kolekcí, nastavení logování pomalých dotazů a správa síťového připojení shardů pro simulaci výpadku.

```
// 13 – sharding – krátký výpis stavu
sh.status();

// 14 – informace o kolekci s validačním schématem
use("ecommerce");
db.getCollectionInfos({ name: "orders" });

// 15 – změna zpřísnění validace (přidavat do validace nejde takže se
přepíše):
// povolí jen známé stavy objednávky
use("ecommerce");
db.runCommand({
    collMod: "orders",
    validator: {
        $jsonSchema: {
```

```
            bsonType: "object",
            required: ["order_status"],
            properties: {
                order_status: { enum: ["created", "approved", "shipped",
"delivered", "canceled", "returned"] }
            }
        }
    }
});

// 16 – Distribuce dat zakazniku skrz shardy
use("ecommerce");
db.customers.getShardDistribution()

// 17 - Zobrazení pomalých dotazů a zdrojů připojení
use("ecommerce");
db.adminCommand({
    getLog: "global"
}).log.filter(line => line.includes("command")).slice(0, 50);

// 18 – odpojení a připojení shardu od clusteru
docker network disconnect -f funkcni_reseni_default "shard-02-node-a"
docker network connect funkcni_reseni_default "shard-02-node-a"
```

7.4 Nested dokumenty

Analyzuje věci nad zanořenými dokumenty, zejména s order.items, který obsahuje jednotlivé položky objednávek

```
// 19 – objednávky, kde alespoň jedna položka stála > 100 Kč
use("ecommerce");
db.orders.find(
    { items: { $elemMatch: { price: { $gt: 100 } } } },
    {
        _id: 1,
        order_status: 1,
        order_purchase_timestamp: 1,
        "items.price": 1,
        "items.product_id": 1,
        "items.shipping_limit_date": 1
    }
).limit(5);

// 20 – projekce jen drahých položek (> 200 Kč) – používá $filter
use("ecommerce");
db.orders.aggregate([
    {
```

```
        $lookup: {
            from: "products",
            localField: "items.product_id",
            foreignField: "_id",
            as: "products"
        }
    },
    {
        $project: {
            _id: 1,
            order_status: 1,
            order_purchase_timestamp: 1,
            customer_id: 1,
            expensiveItems: {
                $filter: {
                    input: "$items",
                    as: "it",
                    cond: { $gt: ["$$it.price", 200] }
                }
            },
            products: {
                $filter: {
                    input: "$products",
                    as: "p",
                    cond: { $in: ["$$p._id", "$items.product_id"] }
                }
            }
        }
    }
])

// 21 – výpočet marže na položku
use("ecommerce");
db.orders.aggregate([
    { $unwind: "$items" }, // rozbalí pole veci v objednávce
    {
        $lookup: {
            from: "products", // připojí kolekci products
            localField: "items.product_id",
            foreignField: "_id",
            as: "product"
        }
    },
    { $unwind: "$product" }, // rozbalí pole veci v produktu
    {
        $project: {
            order: "$_id",
            order_status: 1,
```

```
            customer_id: 1,
            // product_id: "$items.product_id",
            // product_category: "$product.product_category_name",
            // product_weight_g: "$product.product_weight_g",
            price: "$items.price",
            freight: "$items.freight_value",
             margin: { $subtract: ["$items.price", "$items.freight_value"]
}
        }
    }
]);

// 22 – update nested pole pomocí pozičního operátoru [$]
// nastaví novou lhůtu dopravy pro první položku konkrétního produktu
use("ecommerce");
db.orders.findOneAndUpdate(
       {   _id:   "05b126974a95c4a8c2ed51e8e0334c6e",   "items.product_id":
"83b9bc6aae6f527ff6aafb9e01d6cbf3" },
    { $set: { "items.$.shipping_limit_date": new Date(Date.now() + 3 * 24
* 60 * 60 * 1000) } },
    {
        returnDocument: "after",
        projection: {
            _id: 1,
            order_status: 1,
            order_purchase_timestamp: 1,
            "items.product_id": 1,
            "items.shipping_limit_date": 1,
            "items.price": 1
        }
    }
);

/*
23 – $reduce uvnitř $addFields
   z každé objednávky udělá malý "statistický balíček" nad embedded polem
items
   počítáme celkové poštovné, kolik položek stálo > 200 Kč a nejnižší cenu
v objednávce
    $reduce prochází položku po položce a kumuluje hodnoty do objektu
$$value             */
use("ecommerce");
db.orders.aggregate([
    {
        $addFields: {
            itemStats: {
                $reduce: {
                    input: "$items",
```

```
                        initialValue: { totalFreight: 0, expensiveCnt: 0,
minPrice: Number.MAX_VALUE },
                    in: {
                        totalFreight: { $add: ["$$value.totalFreight",
"$$this.freight_value"] },
                        expensiveCnt: {
                            $add: ["$$value.expensiveCnt",
                                { $cond: [{ $gt: ["$$this.price", 200] },
1, 0] }]
                        },
                        minPrice: {
                            $cond: [
                                { $lt: ["$$this.price", "$$value.minPrice"]
},
                                "$$this.price",
                                "$$value.minPrice"
                            ]
                        }
                    }
                }
            }
        }
    },
    {
        $project: {
            _id: 1,
            order_status: 1,
            "itemStats.totalFreight": 1,
            "itemStats.expensiveCnt": 1,
            "itemStats.minPrice": 1
        }
    },
    { $limit: 8 }                                                    // výsledků
jen pár, ať je ukázka čitelná
]);

use("ecommerce");
/*
24 – $bucketAuto – dynamické cenové "pásma" objednávek
   sečteme ceny položek uvnitř každé objednávky (unwind → group)
   $bucketAuto podle totalPrice rozdělí všechny objednávky do 4 intervalů
po 10 objednávkách
   pro každý koš vracíme seznam objednávek a průměrný počet položek       */
db.orders.aggregate([
    { $unwind: "$items" },
    {
        $group: {
            _id: "$_id",
```

```
            order_status: { $first: "$order_status" },
            totalPrice: { $sum: "$items.price" },
            itemCnt: { $sum: 1 }
        }
    },
    {
        $bucketAuto: {
            groupBy: "$totalPrice",
            buckets: 4,
            output: {
                 orders: { $push: { orderId: "$_id", total: "$totalPrice",
order_status: "$order_status" } },
                avgItems: { $avg: "$itemCnt" }
            }
        }
    },
    {
        $project: {
            _id: 0,
              rangeMin: "$_id.min",           // hranice intervalu, které
bucketAuto vyneslo do _id
            rangeMax: "$_id.max",
            avgItems: 1,
            orders: { $slice: ["$orders", 10] } // jen prvních 10 z každého
"kyblíku"
        }
    }
]);
```

7.5 Indexy – vytváření, statistiky a debug příkazy

Dotazy v této části ukazují vytváření a použití složených indexů, jejich mazání, diagnostiku pomocí explain() a statistiky využití indexů ($indexStats). Tyto techniky jsou klíčové pro ladění výkonu, zejména u složitějších dotazů.

```
// 25 – tvorba compound indexu na orders (customer + status)
use("ecommerce");
db.orders.createIndex({ customer_id: 1, order_status: 1 });

// 26 – dotaz, který nutí Mongo použít právě vytvořený index (hint)
use("ecommerce");
db.orders.find(
      {  customer_id:  "e50a30de3c32f9406a7185f40ce6874d",  order_status:
"delivered" }
).hint({ customer_id: 1, order_status: 1 });
```

```
// 27 - smazání nevyužívaného indexu na products.product_category_name
use("ecommerce");
db.products.dropIndex({ product_category_name: 1 });

// 28 – explain – přehled, zda se index použije a jaký node
use("ecommerce");
db.orders.find(
      {  customer_id:  "e50a30de3c32f9406a7185f40ce6874d",  order_status:
"created" }
).hint({ customer_id: 1, order_status: 1 }).explain("executionStats");

// 29 – výpis všech indexů kolekce customers
use("ecommerce");
db.customers.getIndexes();

// 30 – $indexStats – dlouhodobá statistika používání indexů na orders
use("ecommerce");
db.orders.aggregate([{ $indexStats: {} }]);
```

Závěr

V rámci této semestrální práce jsem navrhl a implementoval distribuovaný MongoDB cluster pomocí Docker Compose pro prostředí e-commerce aplikace, konkrétně na základě brazilského datasetu Olist.

Pochvalit se můžu za funkční řešení: podařilo se mi sestavit plně funkční sharded cluster se všemi důležitými komponentami (config servery, replikované shardy, routery), implementovat zabezpečení pomocí RBAC a keyfile, a zároveň připravit strukturovaný datový model s kombinací vnořených dokumentů a referencí. Díky použití reálného datasetu bylo možné simulovat běžné provozní scénáře a otestovat funkčnost a výkon systému pomocí desítek různých dotazů včetně agregací a složitějších operací nad embedded poli.

Kriticky je však třeba říci, že použité e-commerce data nejsou typickým příkladem pro plné využití výhod MongoDB – data jsou převážně strukturovaná, málo se mění, a mnoho případů použití by zvládla i relační databáze. V některých případech (např. reviews, detailnější produktové relace) by bylo vhodné datový model rozšířit o další typy zanoření nebo denormalizace.

Z výsledků plyne, že moje řešení je dobře použitelné pro analýzu a správu větších datových objemů, je škálovatelné a připravené k dalšímu rozšiřování – např. o API vrstvu nebo nasazení do cloudu. MongoDB zde dobře obstálo jako univerzální NoSQL nástroj, zejména díky flexibilitě modelování a snadnému škálování.

Děkuji autorovi Github repozitáře minhhungit/mongodb-cluster-docker-compose:, ze kterého jsem vycházel a současným AI nástrojům, zejména editor Cursor, za pomocí něhož jsem projekt konzultoval a vzdělával se o technických problémech a neznámých.

Celkově lze říci, že práce splnila stanovené cíle a prohloubila mé znalosti jak v oblasti NoSQL databází, tak i v oblasti návrhu distribuovaných systémů.

Zdroje

[1] „minhhungit/mongodb-cluster-docker-compose: docker-compose for mongodb cluster sharded with replication". Viděno: 19. květen 2025. [Online]. Dostupné z: https://github.com/minhhungit/mongodb-cluster-docker-compose
[2] „How eBay Solves the Database Scaling Problem with MongoDB", Hybrid Cloud Management and Automation | Morpheus. Viděno: 19. květen 2025. [Online]. Dostupné z: https://morpheusdata.com/resources/cloud-blog-how-ebay-solves-the-database-scaling-problem-with-mongodb/
[3] Mehtab, „How Discord Stores Trillions Of Messages", Medium. Viděno: 19. květen 2025. [Online]. Dostupné z: https://medium.com/@iBMehta/how-discord-stores-trillions-of-messages-31ed9195c3e8
[4] „L'Oréal & MongoDB: Improving Application Performance And Developer Productivity", MongoDB. Viděno: 19. květen 2025. [Online]. Dostupné z: https://www.mongodb.com/solutions/customer-case-studies/loreal
[5] „MongoDB: Flexible database for the agile age". Viděno: 18. květen 2025. [Online]. Dostupné z: https://www.oracle.com/apac/database/mongodb/
[6] „Brazilian E-Commerce Public Dataset by Olist". [Online]. Dostupné z: https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce

Přílohy

Github s repozitářem https://github.com/sirluky/nosql-semestralni-prace