UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu

MongoDB

Martin, Rampouch, 2. ročník

V Hradci Králové
dne 21.5.2025

Obsah

Úvod

Tato semestrální práce se věnuje návrhu a implementaci distribuovaného systému založeného na dokumentově orientované databázi MongoDB. Cílem je vytvořit plně funkční databázovou infrastrukturu, která demonstruje klíčové vlastnosti moderních NoSQL databází, jako jsou horizontální škálování, vysoká dostupnost dat, replikace a shardování.

MongoDB bylo zvoleno pro své široké praktické využití, přehlednou práci s JSON dokumenty a schopnost snadno modelovat i komplexnější struktury dat. Projekt pracuje s verzí MongoDB 8.0.8**,** nasazenou pomocí Dockeru v rámci více kontejnerového prostředí, které umožňuje simulovat reálné nasazení shardovaného clusteru.

Práce propojuje teoretické poznatky z oblasti NoSQL databází s praktickými zkušenostmi s konfigurací a správou clusterového řešení. Kromě toho se zaměřuje i na integraci reálných dat, jejich validaci a analýzu pomocí skriptů v jazyce Python.

Cílem této práce není detailní optimalizace výkonu nebo pokročilé zabezpečení pro produkční prostředí. Stejně tak nejsou řešeny cloudové varianty nasazení (např. MongoDB Atlas, AWS). Hlavní přínos spočívá ve studijním a demonstračním charakteru práce, která má sloužit jako praktický návod pro studenty nebo vývojáře hledající úvod do distribuovaného světa MongoDB.

1 Architektura

1.1 Schéma a popis architektury

Celé řešení je postaveno jako distribuovaný MongoDB cluster, rozdělený do několika typů kontejnerů, které spolu tvoří plnohodnotnou škálovatelnou databázovou infrastrukturu. Architektura byla navržena s důrazem na splnění požadavků zadání (sharding, replikace, minimálně 3 uzly) a zároveň tak, aby bylo možné celý systém jednoduše spustit pomocí docker compose a inicializačního scriptu *manual-cluster-init.sh*.

1.2 Specifika konfigurace

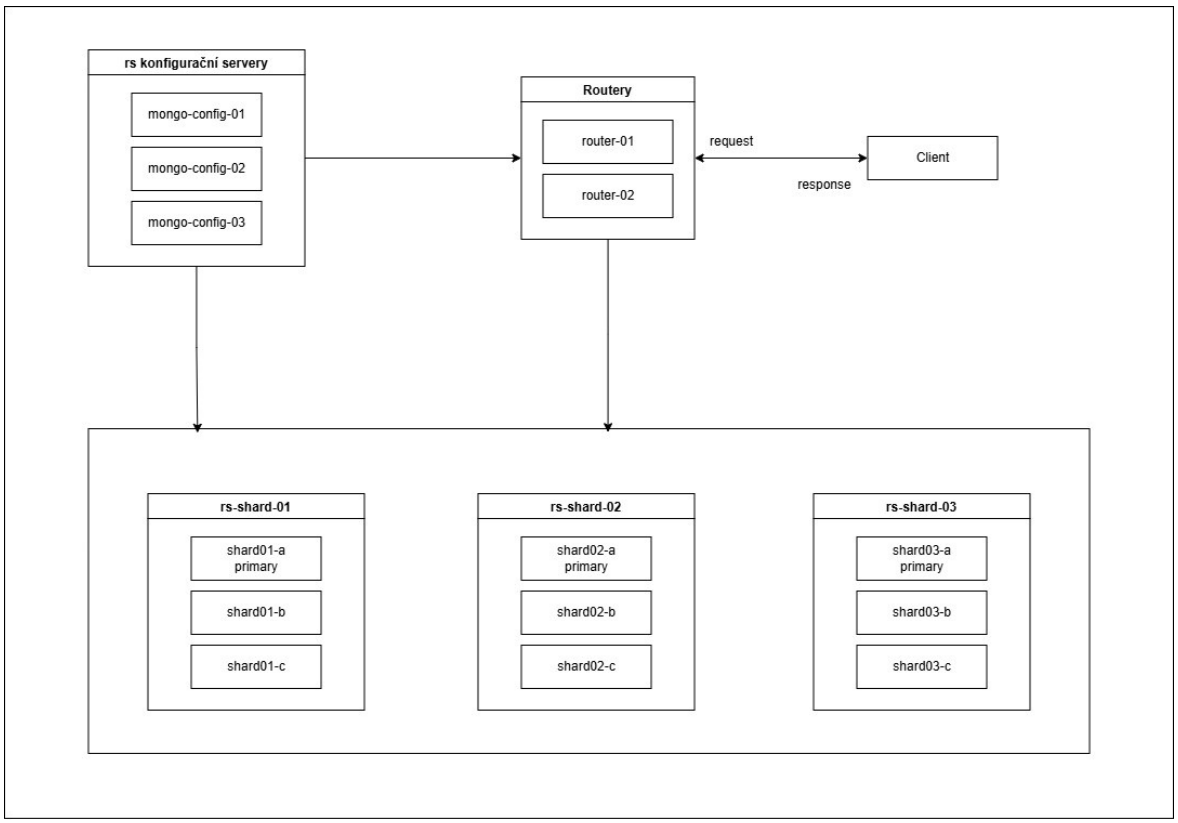
1.2.1 CAP teorém

V rámci práce je MongoDB provozováno jako distribuovaný systém využívající sharding i replikaci. Tím automaticky spadá do modelu, kde systém garantuje Partition tolerance (P). Zbývající dvě vlastnosti – Availability (A) a Consistency (C) – jsou do určité míry volitelné a vzájemně se ovlivňují dle konfigurace.

- **Partition tolerance** je vždy aktivní díky architektuře s vícenásobnými replikami a shardovanými kolekcemi. Každý shard je nezávislý replica set, který pokračuje v provozu i při síťových výpadcích mezi uzly.
- **Availability** je v této konfiguraci prioritní. MongoDB umožňuje automatický failover na sekundární uzly, takže systém zůstává dostupný i při výpadku primárních instancí.
- **Consistency** může být ovlivněna konfigurací. Ve výchozím nastavení je nastavena silná konzistence (čtení/zápis z primárního uzlu), ale může být upravena pomocí readPreference a writeConcern.

1.2.2 Cluster

MongoDB cluster v této práci je navržen jako škálovatelná a vysoce dostupná databázová infrastruktura, která kombinuje shardování a replikaci. Celý systém je nasazen pomocí docker compose a inicializačního scriptu.



1.2.3 Komponenty clusteru:

- **Config servery** (3)
  Slouží k udržování metadat o shardu a chunku. Každý běží jako samostatný kontejner a společně tvoří config replica set.
  Názvy kontejnerů: mongo-config-01, mongo-config-02, mongo-config-03

- **Shard servery** (3 shardy × 3 repliky)
  Každý shard je samostatný replica set složený ze 3 uzlů
  - shard 1: shard01-a, shard01-b, shard01-c
  - shard 2: shard02-a, shard02-b, shard02-c
  - shard 3: shard03-a, shard03-b, shard03-c

- **Routery**
  Pro přístup k databázovému clusteru jsou nakonfigurovány dvě mongos instance (router01 a router02). Router01 slouží pro inicializaci clusteru, zatímco router02 demonstruje, že databáze může být obsluhována více nezávislými vstupy.

1.2.4 Uzly

Každý shard v rámci tohoto MongoDB clusteru je implementován jako samostatný replica set složený ze tří uzlů. Replikace slouží ke zvýšení dostupnosti a odolnosti systému – v případě výpadku primárního uzlu dojde k automatickému přepnutí (failoveru) na sekundární uzel, čímž je zajištěno zachování provozuschopnosti databáze.

V projektu jsou celkem tři replikované shardy, každý se třemi uzly:

- rs-shard-01 (uzly: shard01-a, shard01-b, shard01-c)
- rs-shard-02 (uzly: shard02-a, shard02-b, shard02-c)
- rs-shard-03 (uzly: shard03-a, shard03-b, shard03-c)

Každý z těchto replica setů je inicializován skriptem (*init-shardXX.js*), který definuje členy s odpovídající prioritou. Primární uzel má nejvyšší prioritu, sekundární uzly mají nižší, aby nedocházelo k častým přepnutím.

Kromě shardů obsahuje systém také replikovaný config server (rs-config-server) se třemi členy (configsvr01, configsvr02, configsvr03), který zajišťuje konzistentní metadata o struktuře clusteru a rozmístění dat.

Replikační mechanismus je důležitý jak pro vysokou dostupnost, tak pro podporu škálovatelných operací, jako je čtení ze sekundárních uzlů nebo geografická redundance v distribuovaných systémech.

1.2.5 Sharding

Sharding je klíčovým prvkem horizontálního škálování. Data jsou rozdělena mezi tři shardované replica sety, což zajišťuje efektivní rozložení zátěže a umožňuje práci s rozsáhlými datasety bez přetížení uzlů. Každý shard má vlastní tři členy.

Shardování je aktivováno pomocí skriptu *manual-cluster-init.sh*, který:

- přidá jednotlivé shardy do clusteru,
- zapne sharding na cílové databázi,
- nastaví shardovací klíč pro konkrétní kolekce

1.2.6 Replikace

Každý shard i config server je implementován jako replica set se třemi uzly, zajišťuje vysokou dostupnost, odolnost vůči výpadkům a podporu pro čtení ze sekundárních uzlů.

- replikované shardy (každý 3 uzly)
- 1 replikovaný config server (také 3 uzly).

Primární uzel každého replica setu má nejvyšší prioritu a ostatní jsou nastaveny jako sekundární.

Při výpadku až dvou uzlů v jednom shardu zůstává databáze funkční.

1.2.7 Perzistence dat

Pro perzistenci dat využívá každý MongoDB kontejner připojený *volume svazek*. Díky tomu jsou všechna data uchována i po restartu kontejneru nebo hostitele.

MongoDB používá výchozí storage engine WiredTiger, který:

- ukládá data na disk
- udržuje aktivní pracovní set v paměti RAM
- pravidelně vytváří journals pro zotavení

Import dat a následná analýza probíhá nad těmito kolekcemi. Při načítání jsou data automaticky obnovena z disku.

1.2.8 Distribuce dat

Distribuce dat probíhá na základě shard klíče zvoleného při aktivaci shardingového režimu. Každý dokument je při zápisu směrován na konkrétní shard podle jeho hodnoty. Kolekce byly shardovány podle jednoznačných identifikátorů, aby bylo dosaženo rovnoměrného rozložení.

Rozdělení dat lze ověřit pomocí příkazů jako:

- *sh.status()*
- *db.collection.stats()*
- *sh.getShardedDataDistribution()*

1.2.9 Zabezpečení

Zabezpečení systému je zajištěno následujícími mechanismy:

- Autentizace uživatele: vytvořen uživatel ***martin*** s heslem ***rampouch*** a rolí **root** na databázi admin.
- Autorizace: řízený přístup pomocí rolí a přihlášení.
- Keyfile: každý MongoDB uzel používá sdílený keyfile (*mongo-keyfile*) pro ověření identity při komunikaci mezi uzly.

Toto zajišťuje, že přístup do clusteru mají pouze autorizované entity a komunikace mezi uzly je důvěryhodná.

2 Funkční řešení

2.1 Struktura

```
.
├── .gitattributes                          # Git metadata
├── .gitignore
├── Data                                    # Vstupní data a analýza dat
│   ├── Cleaned                             # Očištěné CSV soubory
│   │   ├── car_advertisements_cleaned.csv
│   │   ├── car_prices_cleaned.csv
│   │   ├── photos_cleaned.csv
│   │   ├── sales_cleaned.csv
│   │   └── wolt_data_cleaned.csv
│   ├── Original                            # Originální stažené datasety
│   │   ├── Ad_table (extra).csv
│   │   ├── Image_table.csv
│   │   ├── Price_table.csv
│   │   ├── Sales_table.csv
│   │   └── Wolt.csv
│   ├── Statistika output                   # Výstup statistických py scriptů
│   │   ├── Ad_table (extra)
│   │   ├── Image_table
│   │   ├── Price_table
│   │   ├── Sales_table
│   │   └── Wolt
│   ├── analyze_ad_table.py                 # py scritp analýza ad_table
│   ├── analyze_image_table.py              # py scritp analýza image_table
│   ├── analyze_price_table.py              # py scritp analýza price_table
│   ├── analyze_sales_table.py              # py scritp analýza sales_table
│   ├── analyze_wolt_dataset.py             # py scritp analýza wolt_dataset
│   └── clean_valid_data.py           # py scritp - předzpracování a očištění
├── Dotazy                        # Složka s tematicky rozdělenými MongoDB dotazy
│   ├── 01 Práce s daty.txt
│   ├── 02 Agregační funkce.txt
│   ├── 03 Embadded dokumenty.txt
│   ├── 04 Indexy a výkon.txt
│   └── 05 Cluster a konfigurace.txt
├── Funkční řešení                      # Funkční řešení projektu - cluster
│   ├── docker-compose.yml              # Docker Compose řešení MongoDB clusteru
│   ├── mongodb-build
│   │   ├── Dockerfile                  # Definujeimage MongoDB se skripty
│   │   └── auth                        # Složka s keyfile
│   ├── readme.md                       # Info o projektu - spuštění atd…
│   └── scripts                   # Veškeré skripty pro inicializaci a import dat
│       ├── auth.js
│       ├── data                  # složka s kopií Cleaned dat pro import scriptem
│       ├── enable-sharding.js
│       ├── import_all.py
│       ├── import_cars.py
│       ├── import_errors
│       ├── import_images.py
│       ├── import_prices.py
│       ├── import_sales.py
│       ├── import_wolt.py
│       ├── init-configserver.js
│       ├── init-router.js
│       ├── init-shard01.js
│       ├── init-shard02.js
│       ├── init-shard03.js
│       ├── manual-cluster-init.sh      # Hlavní setup skript pro celý cluster
│       └── myImportLib.py              # Vlastní py script - knihovna s funkcí
```

2.2 Instalace

Pro spuštění MongoDB clusteru je nejprve potřeba přejít do adresáře „*Funkční řešení*“ a spustit Docker Compose pomocí příkazu:

```
docker compose up -d
```

Tento příkaz spustí všechny definované kontejnery na pozadí.

2.2.1 Windows: předpoklady pro WSL

V případě použití systému Windows je nutné mít funkční prostředí WSL (Windows Subsystem for Linux), ideálně s distribucí Ubuntu. V nastavení Docker Desktop musí být povolena integrace s touto distribucí:

```
Settings → Resources → WSL Integration → Enable integration with additional distros
```

2.2.2 Instalace Pythonu a potřebných knihoven

Ve WSL (nebo v případě Linuxového systému) musí být dostupný Python 3. Lze ho nainstalovat příkazem:

```
sudo apt install python3 python3-pip -y
```

Dále je nutné nainstalovat následující Python knihovny: *Pymongo, pandas, matplotlib, seaborn, bson, folium.*

K instalaci slouží:

```
pip3 install pymongo pandas matplotlib seaborn bson folium
```

2.2.3 Inicializace clusteru a import dat

Po spuštění Docker kontejnerů a instalaci všech závislostí je třeba spustit inicializační skript, který nastaví MongoDB cluster a provede import dat:

```
./scripts/manual-cluster-init.sh
```

Tento skript se nachází v adresáři „*Funkční řešení/scripts/*“ a doporučuje se spouštět script z kořenového adresáře projektu (*Funkční řešení*).

2.3 docker-compose.yml

```
services:

## Router
  router01:
    build:
      context: mongodb-build
    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
  router02:
    build:
      context: mongodb-build
    container_name: router-02
    command: mongos --port 27017 --configdb rs-config-
server/configsvr01:27017,configsvr02:27017,configsvr03:27017 --bind_ip_all --keyFile
/data/mongodb-keyfile
    volumes:
      - ./scripts:/scripts
      - mongodb_cluster_router02_db:/data/db
      - mongodb_cluster_router02_config:/data/configdb
    ports:
      - 27118:27017
    restart: always
    links:
      - router01

## Config Servers
  configsvr01:
    build:
      context: mongodb-build
    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
  configsvr02:
    build:
      context: mongodb-build
    container_name: mongo-config-02
    command: mongod --port 27017 --configsvr --replSet rs-config-server --keyFile
/data/mongodb-keyfile
    volumes:
      - ./scripts:/scripts
```

```
      - mongodb_cluster_configsvr02_db:/data/db
      - mongodb_cluster_configsvr02_config:/data/configdb
    ports:
      - 27120:27017
    restart: always
  configsvr03:
    build:
      context: mongodb-build
    container_name: mongo-config-03
    command: mongod --port 27017 --configsvr --replSet rs-config-server --keyFile
/data/mongodb-keyfile
    volumes:
      - ./scripts:/scripts
      - mongodb_cluster_configsvr03_db:/data/db
      - mongodb_cluster_configsvr03_config:/data/configdb
    ports:
      - 27121:27017
    restart: always

## Shards
  ## Shards 01
  shard01-a:
    build:
      context: mongodb-build
    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
  shard01-b:
    build:
      context: mongodb-build
    container_name: shard-01-node-b
    command: mongod --port 27017 --shardsvr --replSet rs-shard-01 --keyFile
/data/mongodb-keyfile
    volumes:
      - ./scripts:/scripts
      - mongodb_cluster_shard01_b_db:/data/db
      - mongodb_cluster_shard01_b_config:/data/configdb
    ports:
      - 27123:27017
    restart: always
  shard01-c:
    build:
      context: mongodb-build
    container_name: shard-01-node-c
    command: mongod --port 27017 --shardsvr --replSet rs-shard-01 --keyFile
/data/mongodb-keyfile
    volumes:
      - ./scripts:/scripts
      - mongodb_cluster_shard01_c_db:/data/db
      - mongodb_cluster_shard01_c_config:/data/configdb
    ports:
      - 27124:27017
    restart: always

  ## Shards 02
```

```
  shard02-a:
    build:
      context: mongodb-build
    container_name: shard-02-node-a
    command: mongod --port 27017 --shardsvr --replSet rs-shard-02 --keyFile
/data/mongodb-keyfile
    volumes:
      - ./scripts:/scripts
      - mongodb_cluster_shard02_a_db:/data/db
      - mongodb_cluster_shard02_a_config:/data/configdb
    ports:
      - 27125:27017
    restart: always
    links:
      - shard02-b
      - shard02-c
  shard02-b:
    build:
      context: mongodb-build
    container_name: shard-02-node-b
    command: mongod --port 27017 --shardsvr --replSet rs-shard-02 --keyFile
/data/mongodb-keyfile
    volumes:
      - ./scripts:/scripts
      - mongodb_cluster_shard02_b_db:/data/db
      - mongodb_cluster_shard02_b_config:/data/configdb
    ports:
      - 27126:27017
    restart: always
  shard02-c:
    build:
      context: mongodb-build
    container_name: shard-02-node-c
    command: mongod --port 27017 --shardsvr --replSet rs-shard-02 --keyFile
/data/mongodb-keyfile
    volumes:
      - ./scripts:/scripts
      - mongodb_cluster_shard02_c_db:/data/db
      - mongodb_cluster_shard02_c_config:/data/configdb
    ports:
      - 27127:27017
    restart: always
  ## Shards 03
  shard03-a:
    build:
      context: mongodb-build
    container_name: shard-03-node-a
    command: mongod --port 27017 --shardsvr --replSet rs-shard-03 --keyFile
/data/mongodb-keyfile
    volumes:
      - ./scripts:/scripts
      - mongodb_cluster_shard03_a_db:/data/db
      - mongodb_cluster_shard03_a_config:/data/configdb
    ports:
      - 27128:27017
    restart: always
    links:
      - shard03-b
      - shard03-c
  shard03-b:
    build:
      context: mongodb-build
    container_name: shard-03-node-b
    command: mongod --port 27017 --shardsvr --replSet rs-shard-03 --keyFile
/data/mongodb-keyfile
```

```
    volumes:
      - ./scripts:/scripts
      - mongodb_cluster_shard03_b_db:/data/db
      - mongodb_cluster_shard03_b_config:/data/configdb
    ports:
      - 27129:27017
    restart: always
  shard03-c:
    build:
      context: mongodb-build
    container_name: shard-03-node-c
    command: mongod --port 27017 --shardsvr --replSet rs-shard-03 --keyFile
/data/mongodb-keyfile
    volumes:
      - ./scripts:/scripts
      - mongodb_cluster_shard03_c_db:/data/db
      - mongodb_cluster_shard03_c_config:/data/configdb
    ports:
      - 27130:27017
    restart: always

volumes:
  mongodb_cluster_router01_db:
  mongodb_cluster_router01_config:

  mongodb_cluster_router02_db:
  mongodb_cluster_router02_config:

  mongodb_cluster_configsvr01_db:
  mongodb_cluster_configsvr01_config:

  mongodb_cluster_configsvr02_db:
  mongodb_cluster_configsvr02_config:

  mongodb_cluster_configsvr03_db:
  mongodb_cluster_configsvr03_config:

  mongodb_cluster_shard01_a_db:
  mongodb_cluster_shard01_a_config:

  mongodb_cluster_shard01_b_db:
  mongodb_cluster_shard01_b_config:

  mongodb_cluster_shard01_c_db:
  mongodb_cluster_shard01_c_config:

  mongodb_cluster_shard02_a_db:
  mongodb_cluster_shard02_a_config:

  mongodb_cluster_shard02_b_db:
  mongodb_cluster_shard02_b_config:

  mongodb_cluster_shard02_c_db:
  mongodb_cluster_shard02_c_config:

  mongodb_cluster_shard03_a_db:
  mongodb_cluster_shard03_a_config:

  mongodb_cluster_shard03_b_db:
  mongodb_cluster_shard03_b_config:

  mongodb_cluster_shard03_c_db:
  mongodb_cluster_shard03_c_config:
```

2.3.1 Popis docker-compose.yml

Tento soubor definuje konfiguraci MongoDB clusteru se shardingem a replikací. Každá komponenta běží jako samostatný docker kontejner a je seřazena podle své role. Inicializace clusteru a import dat probíhá manuálně spuštěním skriptu *manual-cluster-init.sh* po startu kontejnerů.

2.3.1.1 Routery (router01, router02)

- Služby typu mongos, které slouží jako brány do MongoDB clusteru.
- Připojují se ke konfiguračnímu replikačnímu setu rs-config-server.
- Naslouchají na portech 27117 a 27118, aby bylo možné testovat více připojení.
- Používají keyFile pro zabezpečenou autentizaci s ostatními uzly clusteru.
- Využívají sdílený volume ./scripts:/scripts pro přístup ke spouštěcím skriptům.

2.3.1.2 Konfigurační servery (configsvr01, configsvr02, configsvr03)

- Tvoří samostatný replikační set rs-config-server.
- Uchovávají metadata o shardingu.
- Každý běží na vlastním portu (např. 27119) a ukládá data do persistentních volumes.

2.3.1.3 Shardy (shard01–03)

- Každý shard je tvořen vlastním replikačním setem, například rs-shard-01, který obsahuje 3 kontejnery: shard01-a, shard01-b, shard01-c.
- Slouží k uložení dat.
- Každý uzel má vlastní persistentní volume.

2.3.1.4 Další technické poznámky:

- **build:** všechny Mongo kontejnery se staví z vlastního Dockerfile, který obsahuje cestu ke keyfile pro replikaci a autentizaci.
- **volumes:** data a konfigurační informace jsou ukládány do pojmenovaných docker volumes, takže přežijí restart nebo znovuspuštění kontejneru.

3 Případy užití a případové studie

3.1 Vhodné případy užití MongoDB

MongoDB je univerzální dokumentově orientovaná NoSQL databáze, která se uplatňuje v široké škále scénářů moderních aplikací. Díky flexibilnímu schématu a dobré škálovatelnosti nachází využití v situacích, kdy je potřeba pracovat s velkými objemy semi-strukturovaných dat, provádět rychlé dotazy a analýzy či pružně měnit datový model. Mezi hlavní oblasti, kde se nasazení MongoDB osvědčuje, patří například:

- **Business intelligence a operativní analýza dat**: MongoDB dokáže zpracovávat obrovské objemy dat v reálném čase a umožňuje firmám získávat rychlé analytické přehledy. Např. MetLife sjednotila data z desítek systémů do MongoDB, čímž získala rychlý přehled o zákaznících a zefektivnila jejich obsluhu
- **E-commerce a katalogy produktů**: Dokumentová databáze je vhodná pro ukládání katalogových informací o produktech, zákaznických profilů či transakcí, kde oceníme flexibilitu přidávání nových atributů. Např. Walmart a eBay využívají MongoDB pro správu produktových katalogů a zákaznických interakcí díky možnosti snadno měnit strukturu uložených dat.
- **Systémy pro správu obsahu (CMS)**: Díky ukládání dat ve formátu JSON (BSON) je MongoDB přirozeně vhodné pro systémy spravující různorodý obsah texty, metadata, obrázky apod. Např. The New York Times používá MongoDB k ukládání archivu článků a médií, což zajišťuje rychlé vyhledávání v nestrukturovaném obsahu.
- **Internet věcí (IoT):** MongoDB bývá nasazeno v IoT platformách ke sběru a analýze dat ze sensorů a zařízení. Díky schopnosti ukládat různorodá datová schémata a horizontálně je škálovat dokáže pojmout obrovské množství událostí generovaných chytrými zařízeními. Např. Bosch používá MongoDB pro sběr a správu velkých objemů dat z IoT senzorů v reálném čase.
- **Mobilní a sociální aplikace**: U aplikací typu sociálních sítí či mobilních her je potřeba ukládat strukturované i nestrukturované informace o uživatelích, příspěvcích, zprávách a další obsah. MongoDB nabízí potřebnou flexibilitu a výkon pro tyto případy užití. Např. Tinder ukládá profily a zprávy uživatelů v MongoDB Atlas, což zajišťuje dostupnost a synchronizaci.

3.2 Využití MongoDB v rámci projektu

V tomto projektu je MongoDB využita k návrhu distribuovaného clusteru zpracovávajícího data z oblastí automotive a doručovacích služeb (Wolt). Díky práci s JSON dokumenty a flexibilnímu schématu bylo možné snadno importovat různorodá data a dále s nimi pracovat bez složitých transformací. Součástí řešení je také sharding a replikace pro škálování výkonu i zajištění dostupnosti v případě výpadku. MongoDB běží v prostředí docker compose a verze 8.0.8 poskytuje aktuální možnosti a bezpečnostní standardy.

MongoDB bylo zvoleno místo Redis a Apache Cassandra, protože lépe vyhovuje požadavkům na analýzu semi-strukturovaných dat. Redis je vhodný spíše pro cache a real-time operace v paměti, Cassandra pro škálovatelný zápis, ale ne pro práci s JSON a flexibilní dotazy. MongoDB naproti tomu kombinuje dokumentový model, agregace, sharding, replikaci a jednoduché nasazení v dockeru. Jde navíc o moderní a široce rozšířenou technologii s dobrou podporou, což z ní dělá ideální volbu pro tento projekt.

3.3 Porovnání s alternativními NoSQL databázemi

Redis je key-value databáze, která pracuje primárně v paměti a vyniká velmi nízkou latencí. Je ideální jako cache nebo pro real-time úlohy (např. fronty zpráv, chaty), ale není navržená pro trvalé uložení velkých dat nebo komplexní analýzy. I když podporuje snapshoty na disk a určitý stupeň škálování pomocí clusteru, nehodí se pro práci s JSON dokumenty a složitější dotazování. V tomto případě, kdy pracujeme s trvale uloženými daty a provádíme agregace nad velkým množstvím záznamů, by Redis představoval výrazná omezení.

Apache Cassandra je sloupcově orientovaná databáze optimalizovaná pro masivní škálování a vysokou dostupnost. Díky architektuře bez master uzlu je vhodná pro systémy s extrémní zátěží zápisem a požadavkem 100% dostupnosti. Nevýhodou je však složitější nasazení, nutnost pečlivého návrhu schématu podle přístupových vzorů a omezená flexibilita dotazování. Efektivní je jen pro dotazy podle primárních klíčů. Pro tento projekt, kde je třeba ad-hoc filtrovat a agregovat data napříč různými poli JSON dokumentů, by Cassandra nebyla vhodná. MongoDB nabízí bohatší dotazovací jazyk, lepší podporu dokumentového modelu a jednodušší vývoj.

Další databáze, jako CouchDB, DynamoDB nebo Neo4j, mají svá specifika, která mohou být výhodná v určitých případech. CouchDB nabízí podobný dokumentový model jako MongoDB, ale disponuje menší komunitou a omezenějším ekosystémem nástrojů. DynamoDB je výkonná a plně spravovaná služba, ale je úzce provázána s AWS prostředím. Grafové databáze jako Neo4j jsou vhodné pro modelování a analýzu silně propojených dat, nikoliv však pro obecné dokumentově orientované scénáře. MongoDB v tomto kontextu představuje univerzálnější volbu kombinující flexibilitu, dostupnost nástrojů a možnosti nasazení napříč prostředími.

3.4 Případové studie z praxe

3.4.1 Toyota Connected – telematická platforma s vysokou dostupností

Kontext: Toyota Connected North America je digitální divize automobilky Toyota, která zajišťuje služby pro připojená vozidla. Vyvíjí například systém Safety Connect, který detekuje dopravní nehody a propojuje vozidla s operátory záchranné služby.

Problém: Původní systém běžel na klasické on-premise databázi, která trpěla výpadky a nebyla schopná efektivně škálovat. To bylo kritické výpadky systému znamenaly riziko selhání při život ohrožujících situacích. Firma chtěla dosáhnout 99,99% dostupnosti a možnost okamžitého zpracování událostí z milionů vozidel.

Řešení: Toyota přešla na MongoDB Atlas plně spravovanou cloudovou databázovou službu běžící na AWS. Nasadili mikroservisní architekturu, kde MongoDB zajišťuje backend každé části systému. Klíčovou roli sehrála podpora multiregionální replikace, která zajistila vysokou dostupnost a toleranci výpadků. MongoDB slouží jako centrální úložiště telemetrických dat od jejich příjmu až po zobrazení operátorovi.

Výsledek: Dostupnost systému vzrostla na 99,99 %, databázová infrastruktura zvládá vysoké zatížení a tým je schopen provádět údržbu bez výpadků služby. Díky MongoDB se podařilo výrazně zrychlit zpracování nouzových hlášení (typicky 3 sekundy od události po zobrazení informací). Firma navíc snížila náklady natolik, že mohla začít službu Safety Connect nabízet zdarma. MongoDB zde prokázalo svou robustnost a vhodnost pro mission-critical aplikace v automobilovém průmyslu.

3.4.2 Shutterfly – škálovatelné úložiště metadat o fotografiích

Kontext: Shutterfly je populární online platforma pro sdílení a tisk fotografií, která obsluhuje miliony uživatelů a ukládá miliardy snímků. Původně firma používala Oracle databázi pro správu metadat o fotografiích.

Problém: Oracle se při růstu objemu dat stal bottleneckem a výkon klesal, údržba byla složitá a přidávání nových atributů vyžadovalo úpravy schématu. Firma potřebovala databázi, která zvládne obrovský objem transakcí (cca 10 000 operací za sekundu), umožní rychlý vývoj a snadné škálování.

Řešení: Po rozsáhlém testování NoSQL řešení Shutterfly přešlo na MongoDB. Dokumentový model umožnil ukládat metadata k obrázkům jako JSON dokumenty, což vývojářům výrazně usnadnilo úpravy a vývoj nových funkcí. MongoDB běželo nejprve v on-premise clusterech a později bylo přeneseno na MongoDB Atlas. Důležitá byla možnost horizontálního škálování bez výpadků.

Výsledek: Firma zaznamenala výrazné zlepšení výkonu a odezvy aplikace. Vývojáři získali flexibilitu při úpravách datové struktury bez nutnosti složitých SQL migrací. MongoDB umožnilo Shutterfly zvládat nárůst dat i uživatelů bez nutnosti zásadní přestavby architektury. Kritické systémy zůstaly v relačních databázích, čímž vznikla hybridní architektura MongoDB se specializuje na big data a metadata.

3.4.3 MetLife – sjednocení zákaznických dat napříč systémy

Kontext: MetLife je globální pojišťovna, která spravuje služby pro 90 milionů klientů. V minulosti používala více než 70 různých interních systémů, které uchovávaly zákaznická data. Tato roztříštěnost komplikovala obsluhu klientů a znemožňovala komplexní pohled na zákazníka.

Problém: Firma chtěla vytvořit jednotný 360° profil zákazníka zahrnující všechny smlouvy, události, kontakty i platby dostupný na jednom místě a v reálném čase. Tradiční přístupy (např. datový sklad) selhaly kvůli složitosti a zdlouhavé realizaci. Firma potřebovala flexibilní databázi, která umožní rychlou integraci dat z desítek systémů.

Řešení: MetLife postavilo aplikaci „The Wall“ na MongoDB. Každý zákazník má svůj vlastní dokument, do kterého se agregují informace ze všech systémů. MongoDB umožnilo iterativní vývoj, snadné rozšiřování dokumentů a rychlé indexování. Cluster běží ve dvou datových centrech, je replikovaný a umožňuje vysokou dostupnost.

Výsledek: Projekt byl spuštěn během tří měsíců (předchozí pokusy trvaly roky). Aplikace zjednodušila obsluhu klientů a zvýšila produktivitu call center. MongoDB umožnilo spojit technická i obchodní data do jednoho systému a otevřelo cestu k pokročilé analytice. Projekt The Wall se stal ukázkovým případem digitální transformace firmy a MongoDB klíčovým prvkem její datové strategie.

4 Výhody a nevýhody

4.1 Výhody a nevýhody MongoDB a konkrétního řešení v projektu

4.1.1 Obecné výhody MongoDB

MongoDB je dokumentově orientovaná NoSQL databáze, která umožňuje flexibilní ukládání dat bez pevně daného schématu. Ukládá data ve formátu BSON (JSON), což usnadňuje práci s různorodými strukturami a umožňuje jejich snadné rozšiřování. Díky tomu je vhodná pro rychlý vývoj a časté změny datové struktury.

Mezi hlavní výhody patří:

- Rychlé dotazování a agregace: MongoDB podporuje bohaté dotazy, indexy a silný agregační framework.
- Škálovatelnost: horizontální škálování pomocí shardingu umožňuje rozdělit data mezi více uzlů a zvýšit výkon.
- Vysoká dostupnost: díky replikaci je možné automaticky přepnout na záložní uzel v případě výpadku.
- Snadná integrace: široká podpora jazyků, dostupné nástroje a aktivní komunita.

4.1.2 Obecné nevýhody MongoDB

- Slabší podpora transakcí: komplexní ACID transakce přes více dokumentů a kolekcí jsou omezené.
- Chybějící referenční integrita: databáze nehlídá cizí klíče, kontrola konzistence je plně na aplikaci.
- Náročnější správa datové kvality: absence pevného schématu může vést k nejednotným strukturám.
- Větší režie u malých datasetů: overhead spojený s pokročilými funkcemi (např. sharding) není vždy efektivní.
- Odlišný dotazovací jazyk: vyžaduje jiný přístup než SQL, což může zpomalit adopci ve firmě.

4.2 Výhody řešení v projektu

V projektu je MongoDB nasazeno jako distribuovaný cluster s podporou shardingu a replikace.

Klíčové výhody řešení:

- Připravenost na velká data: sharding umožňuje škálování v případě nárůstu objemu dat.
- Odolnost vůči výpadkům: replikace zajišťuje vysokou dostupnost.
- Snadné nasazení díky docker compose: celý cluster lze jednoduše spustit lokálně i znovu vytvořit.
- Efektivní analýza dat: agregace probíhají přímo v databázi bez nutnosti exportu dat do jiných nástrojů.

4.3 Nevýhody řešení v projektu

- Zbytečná komplexita u malého datasetu: použití shardingu zvyšuje režii (router, více instancí) bez přímého přínosu.
- Správa clusteru je náročnější: vyžaduje znalosti správy více MongoDB služeb, konfiguraci replikace a správu portů.
- Rizika nevhodné konfigurace: výběr špatného shard klíče nebo nesprávná správa docker volume může vést k problémům.
- Sdílení zdrojů v rámci kontejnerů: všechny MongoDB instance běží na jednom stroji, což snižuje skutečnou odolnost vůči výpadku.

4.4 Shrnutí využití MongoDB v projektu

MongoDB přináší v rámci projektu vysokou flexibilitu, připravenost na budoucí škálování a umožňuje přirozenou práci s JSON daty. Využití shardingu a replikace demonstruje schopnosti moderní distribuované databáze, i když v menším projektu s omezeným objemem dat může být tato architektura zbytečně náročná na správu. Výsledné řešení však přineslo cennou zkušenost s konfigurací pokročilého databázového prostředí a odpovídá současným trendům v práci s daty.

5 Další specifika

Původním cílem bylo plně automatizovat inicializaci clusteru i import dat pomocí *docker-compose.yml*. I přes několik pokusů (např. init-container nebo entrypoint) se však tuto automatizaci nepodařilo spolehlivě rozběhnout. Nakonec byl zvolen přístup: hlavní část clusteru je spuštěna přes docker compose, zatímco inicializace probíhá jednorázově ručně skriptem *manual-cluster-init.sh*. Ten nastaví replikace, povolí sharding a provede import dat.

Kvůli demonstraci funkce shardingu bylo zmenšeno minimální alokační nastavení MongoDB na 1 MB místo původních 64MB.

Projekt byl původně vyvíjen na operačním systému Kubuntu 25.04, ale kvůli snazší správě dokumentace v Microsoft Word byl později přenesen a upraven tak, aby běžel také na Windows 11 s využitím WSL (Ubuntu). Během tohoto přechodu bylo nutné opravit několik shell skriptů, které selhávaly kvůli rozdílné interpretaci (bash vs sh), a upravit některé příkazy docker compose. Po těchto úpravách je možné projekt bez problémů spustit jak na Linuxu, tak na Windows s wsl.s

6 Data

6.1 Dataset Ad_table (extra)

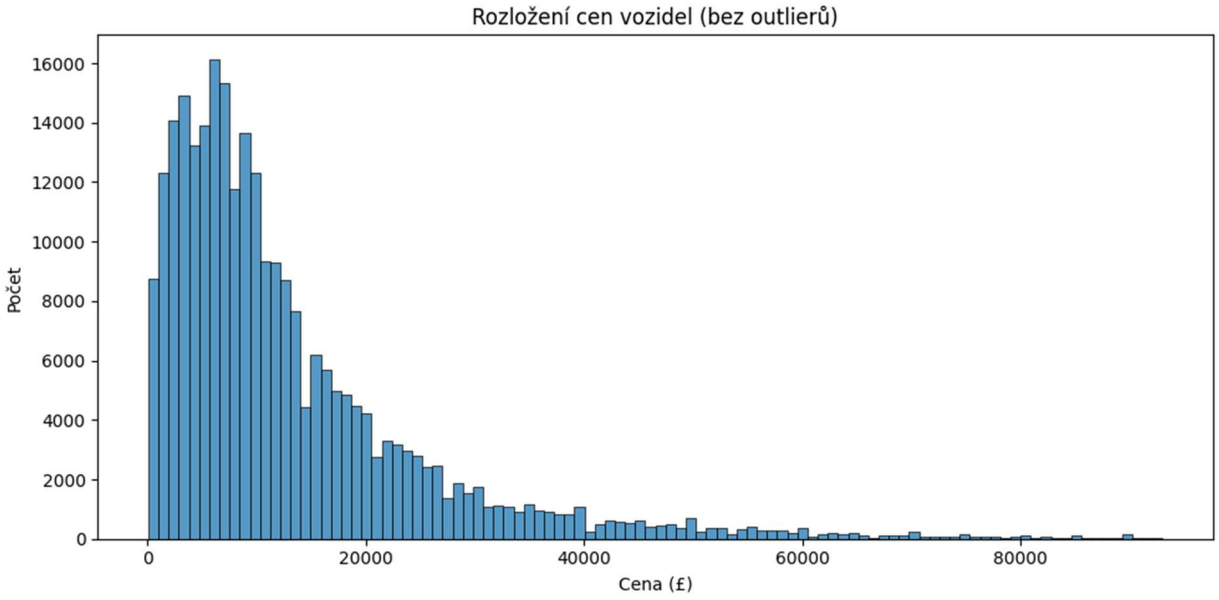
Tento dataset obsahuje 268 255 záznamů o inzerátech vozidel ve Velké Británii. Každý záznam reprezentuje jeden inzerát a zahrnuje technické atributy jako značka výrobce, model, cena, výkon motoru, karoserie, typ paliva nebo spotřeba. Dataset byl získán ze zdroje Deep Visual Marketing a použit v rámci kolekce cars v MongoDB.

6.1.1 Souhrnná statistika a přehled

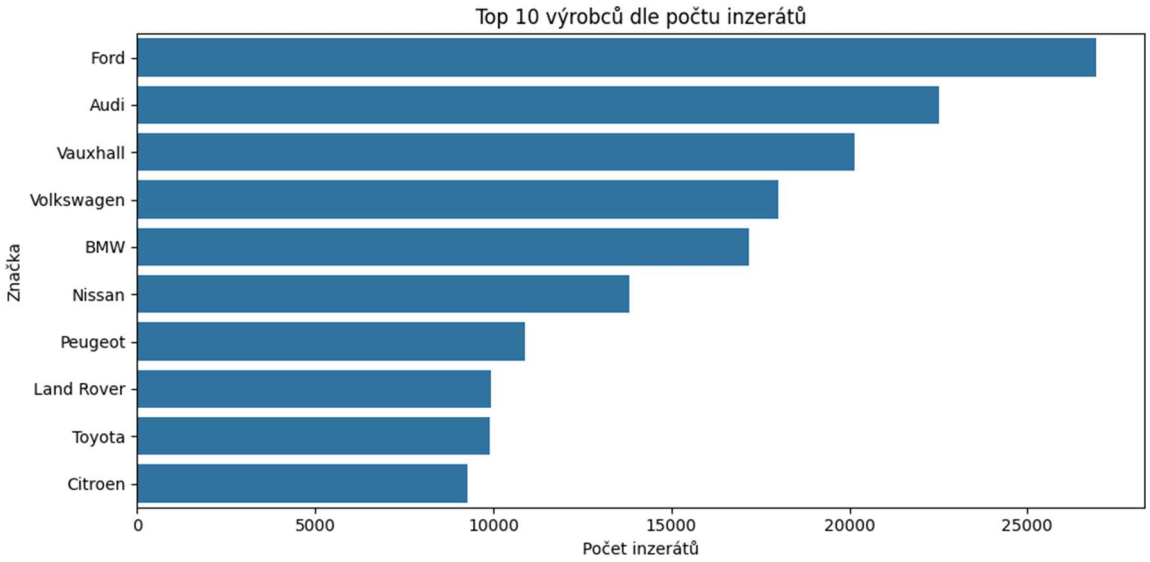
- Počet záznamů: 268 255
- Počet sloupců: 24
- Počet duplicitních Adv_ID: 0
- Počet unikátních modelů (Genmodel): 896
- Počet různých výrobců (Maker): 88
- Počet chybějících hodnot (celkem): 314 404

6.1.2 Vizualizace

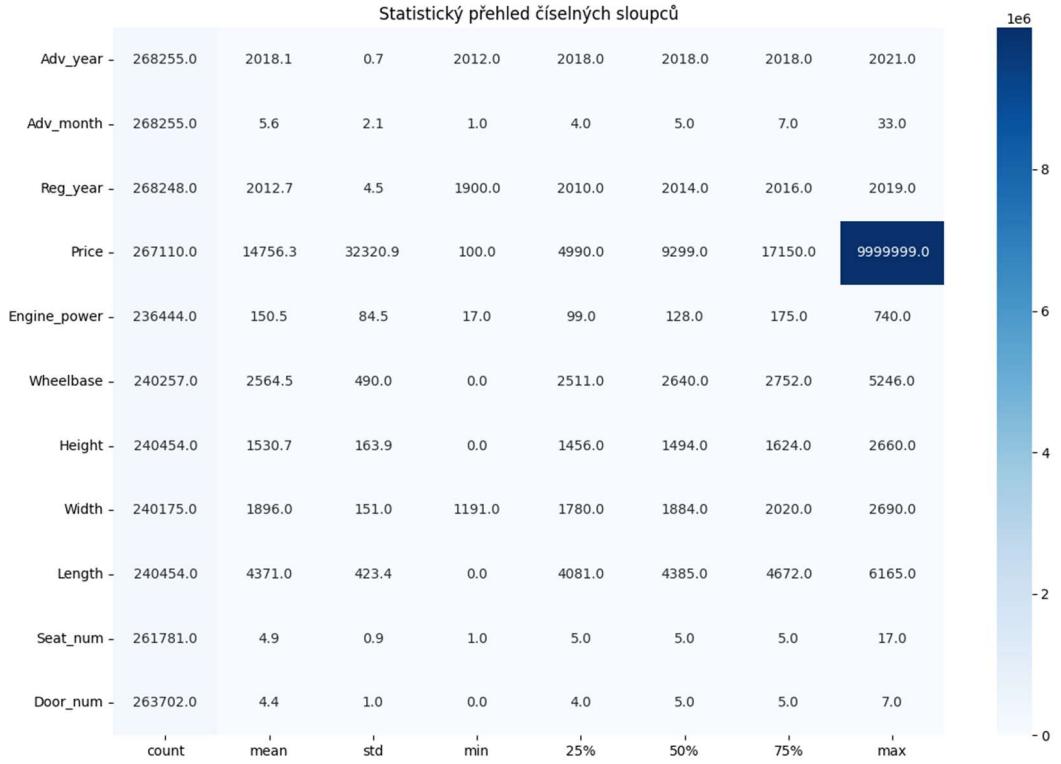
Histogram cen vozidel (po odfiltrování outlierů):



Top 10 značek dle počtu inzerátů:



Základní statistické hodnoty



6.1.3 Ukázka struktury v dokumentu:

```
_id: ObjectId('682b28b3cfcb19ff23f59dc7')
Maker : "Bentley"
Genmodel : "Arnage"
Genmodel_ID : "10_1"
Adv_ID : "10_1$$6"
Adv_year : 2017
Adv_month : 12
Color : "Blue"
Reg_year : 2002
Bodytype : "Saloon"
Runned_Miles : "55000"
Engin_size : "6.8L"
Gearbox : "Automatic"
Fuel_type : "Petrol"
Price : 24950
Engine_power : 450
Annual_Tax : " 315"
Wheelbase : 3116
Height : 1515
Width : 2125
Length : 5390
Average_mpg : "13.7 mpg"
Top_speed : "179 mph"
Seat_num : 5
Door_num : 4
```

```
[direct: mongos] RampaBase> db.cars.find({
...   Maker: { $regex: "skoda", $options: "i" },
...   Genmodel: { $regex: "superb", $options: "i" }
... }).limit(5)
...
[
  {
    _id: ObjectId('682b28b6cfcb19ff23f87623'),
    Maker: 'SKODA',
    Genmodel: 'Superb',
    Genmodel_ID: '80_9',
    Adv_ID: '80_9$$5',
    Adv_year: 2018,
    Adv_month: 3,
    Color: 'White',
    Reg_year: 2017,
    Bodytype: 'Hatchback',
    Runned_Miles: '100',
    Engin_size: '2.0L',
    Gearbox: 'Automatic',
    Fuel_type: 'Diesel',
    Price: 20995,
    Engine_power: 187,
    Annual_Tax: ' 140*',
    Wheelbase: 2841,
    Height: 1468,
    Width: 2031,
    Length: 4861,
    Average_mpg: '60.1 mpg',
    Top_speed: '146 mph',
    Seat_num: 5,
    Door_num: 5
  }
```

6.2 Dataset Image_table

Dataset Image_table obsahuje metadata k více než 1,4 milionu obrázků vozidel. Každý záznam zahrnuje identifikátor modelu (Genmodel_ID), identifikátor obrázku (Image_ID), název souboru obrázku, předpovězený úhel záběru (Predicted_viewpoint) a volitelnou informaci o kvalitě (Quality_check). Dataset je propojitelný s ostatními tabulkami prostřednictvím atributu Adv_ID.

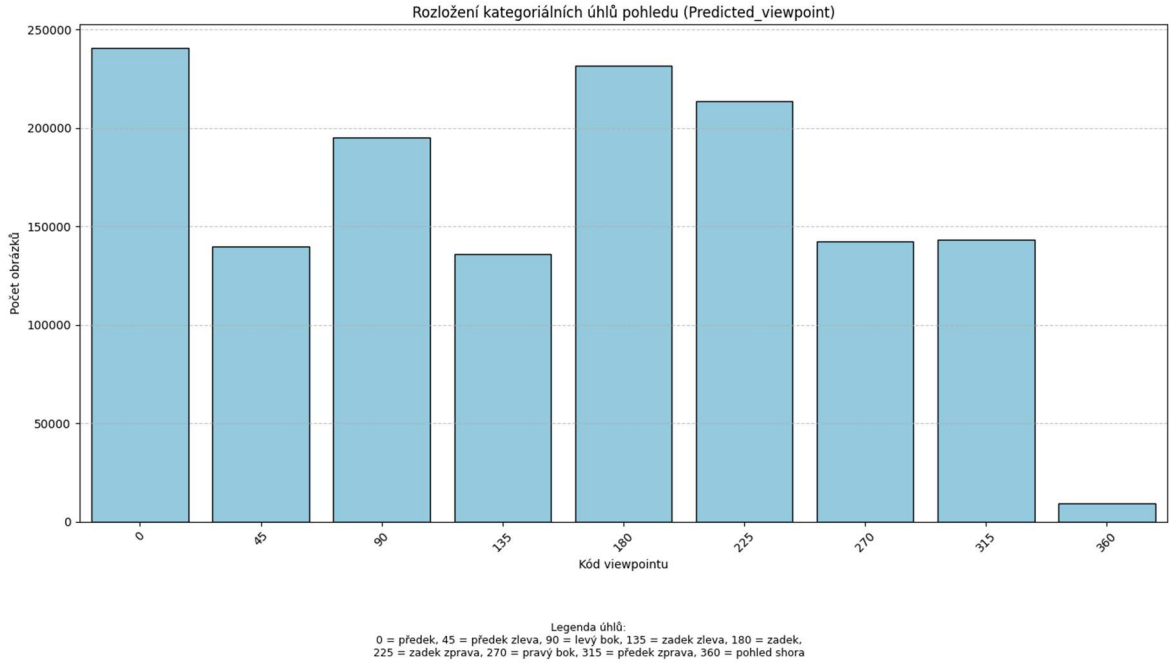
Predicted_viewpoint atribut reprezentuje kategoriální úhly pohledu na vozidlo (např. předek, bok, zadek). Hodnoty jsou reprezentovány čísly jako 0, 50, 100, atd.

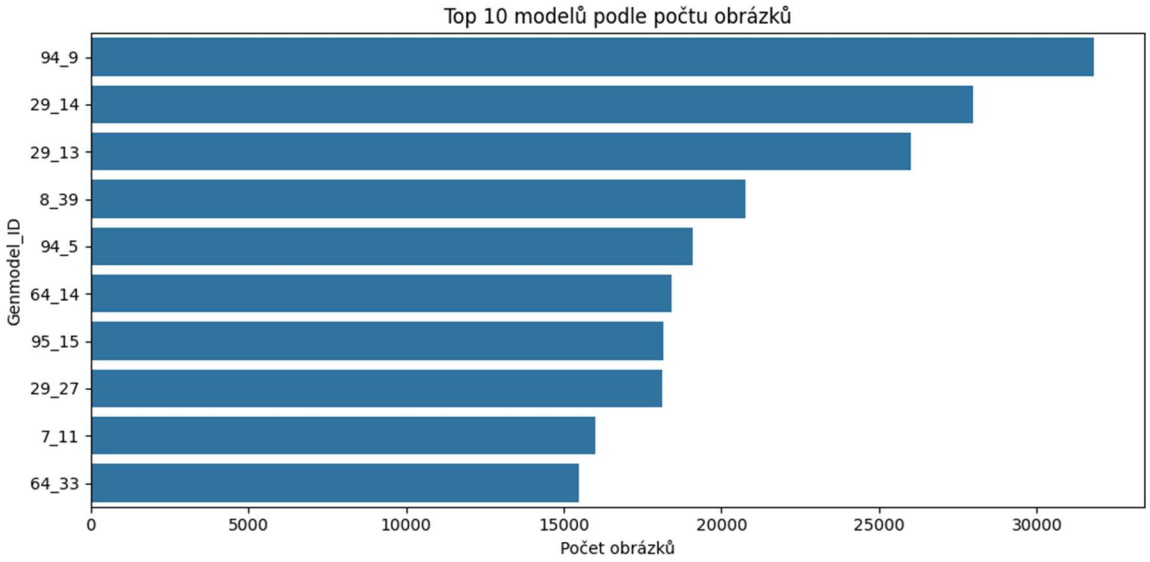
Přesné přiřazení čísel k pohledům (např. 0 = front, 180 = rear) není součástí datasetu, ale je popsáno v uživatelském manuálu DVM-CAR. Dataset včetně manuálu byl získán ze zdroje [Deep Visual Marketing].

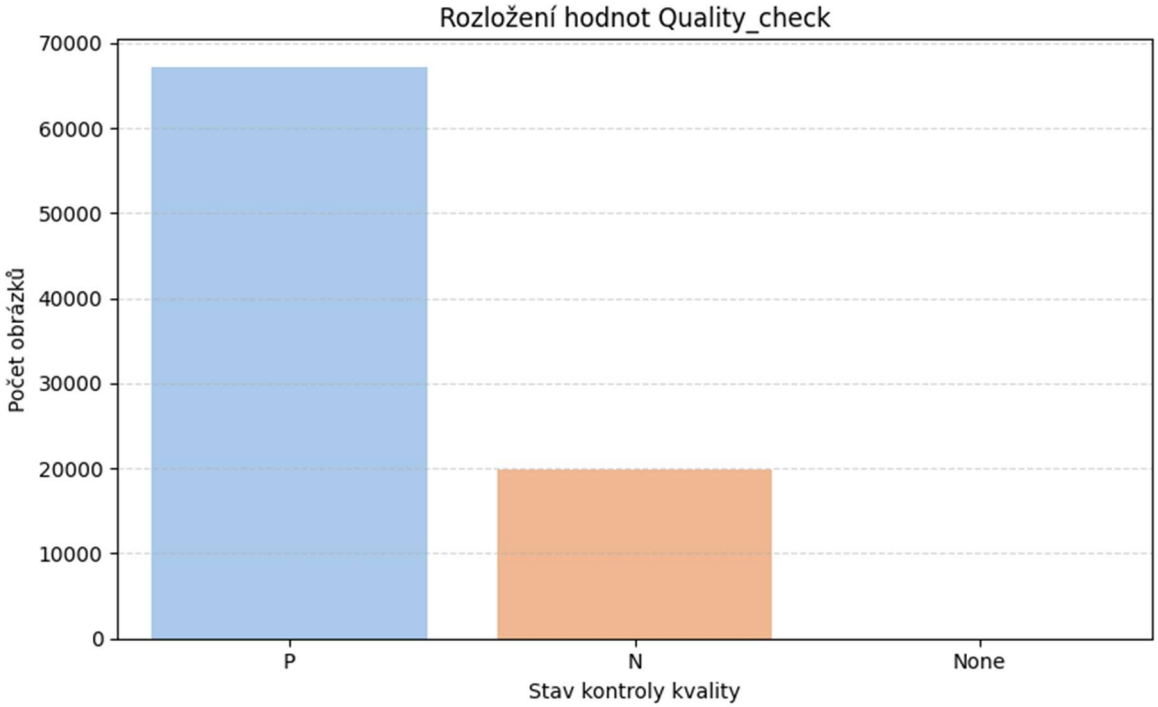
6.2.1 Souhrnná statistika a přehled

- Počet záznamů: 1 451 784
- Počet sloupců: 5
- Chybějící hodnoty celkem: 1 364 791
- Chybí pouze ve sloupci: Quality_check (94 % záznamů)

6.2.2 Vizualizace







P = Passed (prošlo ruční kontrolou), N = Not passed (neprošlo), NaN = nebylo kontrolováno

6.2.3 Ukázka struktury v dokumentu

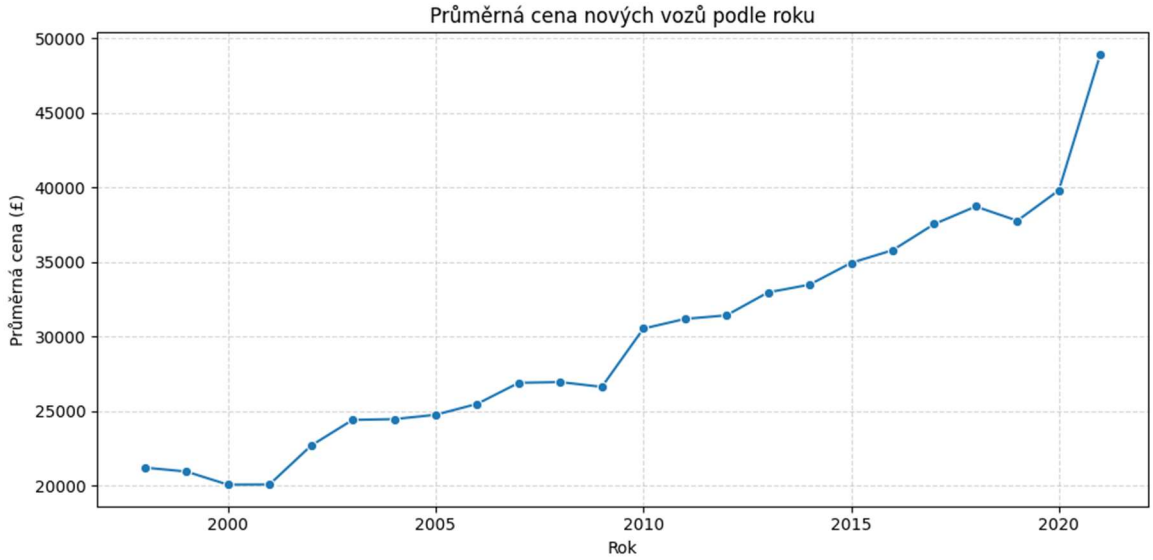
6.3 Dataset Price_table

Dataset Price_table obsahuje informace o vstupních cenách nových vozů při jejich uvedení na trh na britský trh. Data jsou uvedena pro jednotlivé modely (Genmodel_ID) v konkrétním roce. Ceny jsou v britských librách a reprezentují základní výbavovou verzi vozidla. Dataset je využit pro analýzu vývoje cen nových aut a srovnání cen mezi jednotlivými modely.

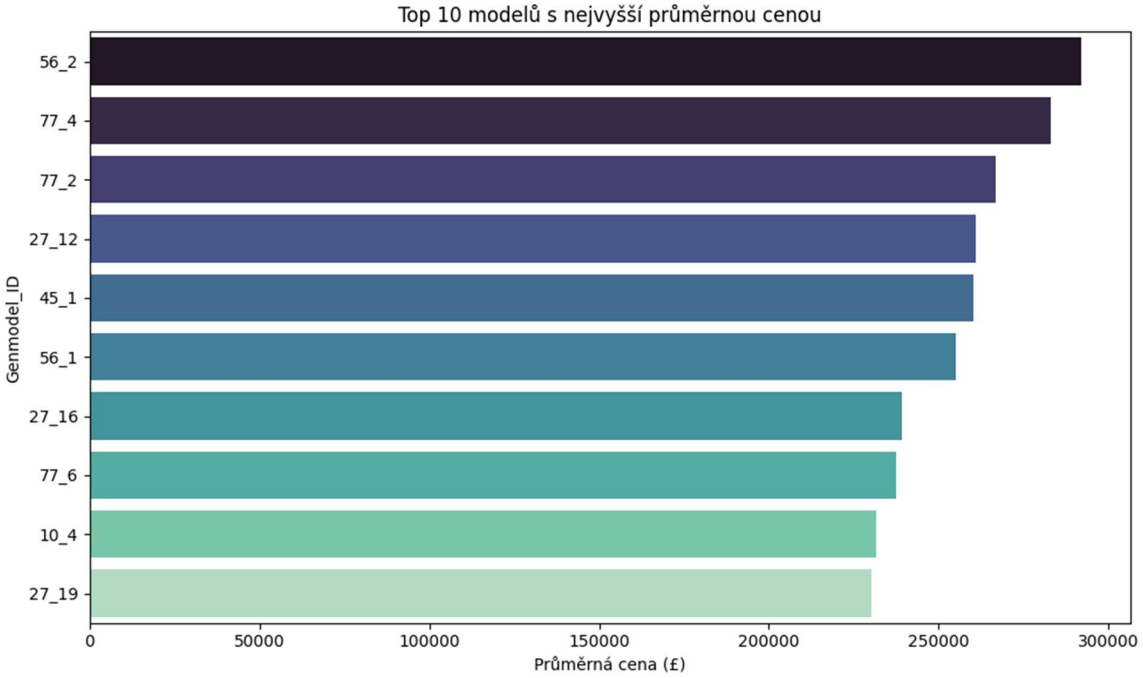
6.3.1 Souhrnná statistika a přehled

- Počet záznamů: 6 333
- Počet sloupců: 5
- Počet chybějících hodnot: 0
- Průměrná cena: 29 400 £
- Medián ceny: 17 815 £
- Nejdražší zaznamenaná cena: 320 120 £
- Nejlevnější zaznamenaná cena: 4 499 £

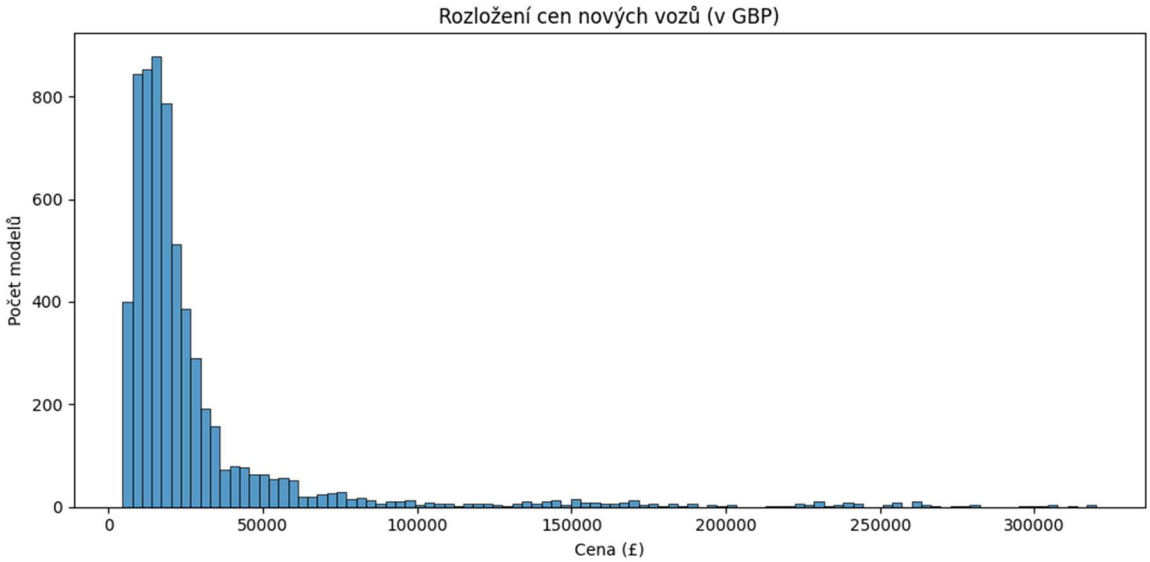
6.3.2 Vizualizace



Z grafu je patrný dlouhodobý trend růstu průměrných cen. Výrazný nárůst je pozorován v posledních letech, zejména po roce 2020.



Modely s nejvyššími průměrnými cenami přesahují hranici 250 000 £, mezi nimi jsou např. 56_2, 77_4, 77_2.



Vizualizace ukazuje, že většina modelů se pohybuje v cenové hladině do 50 000 £.

6.3.3 Ukázka struktury v dokumentu

```
_id: ObjectId('682b28bad7fa1703a725fb3e')
Maker : "Abarth"
Genmodel : "500"
Genmodel_ID : "2_2"
Year : 2009
Entry_price : 13400
```

```
_id: ObjectId('682b28bad7fa1703a725fb3f')
Maker : "Abarth"
Genmodel : "500"
Genmodel_ID : "2_2"
Year : 2010
Entry_price : 13945
```

```
_id: ObjectId('682b28bad7fa1703a725fb40')
Maker : "Abarth"
Genmodel : "500"
Genmodel_ID : "2_2"
Year : 2011
Entry_price : 13945
```

6.4 Dataset Sales_table

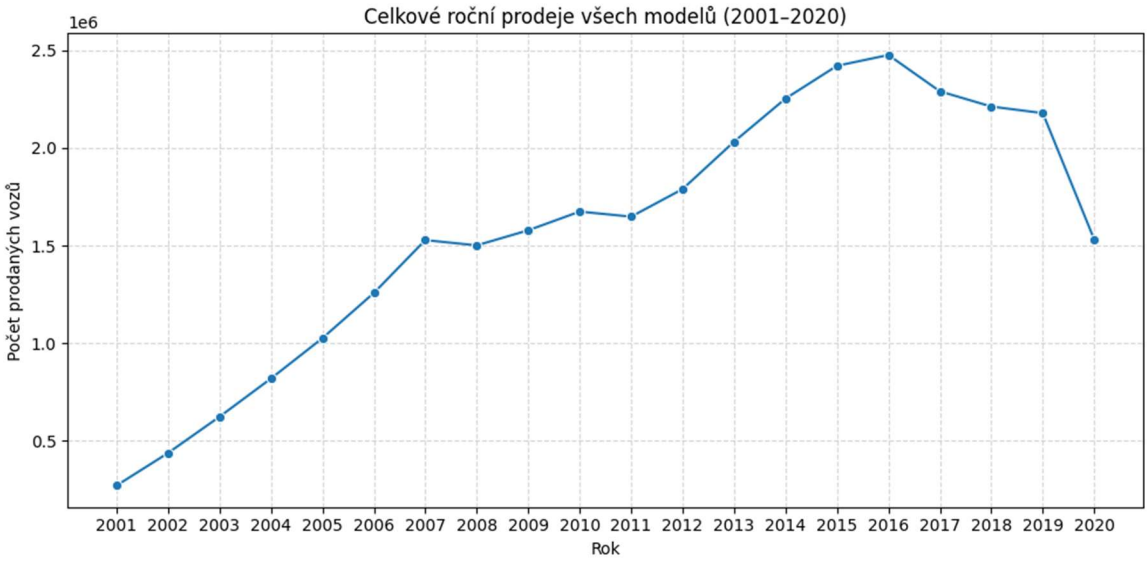
Dataset Sales_table obsahuje počty nově registrovaných vozidel v letech 2001–2020 na trhu ve Spojeném království. Přestože je tabulka označena jako Sales Table, nejde o reálné prodeje (transakce). Tak je zveřejňují vládní úřady (např. DVLA). Dataset byl získán ze zdroje Deep Visual Marketing.

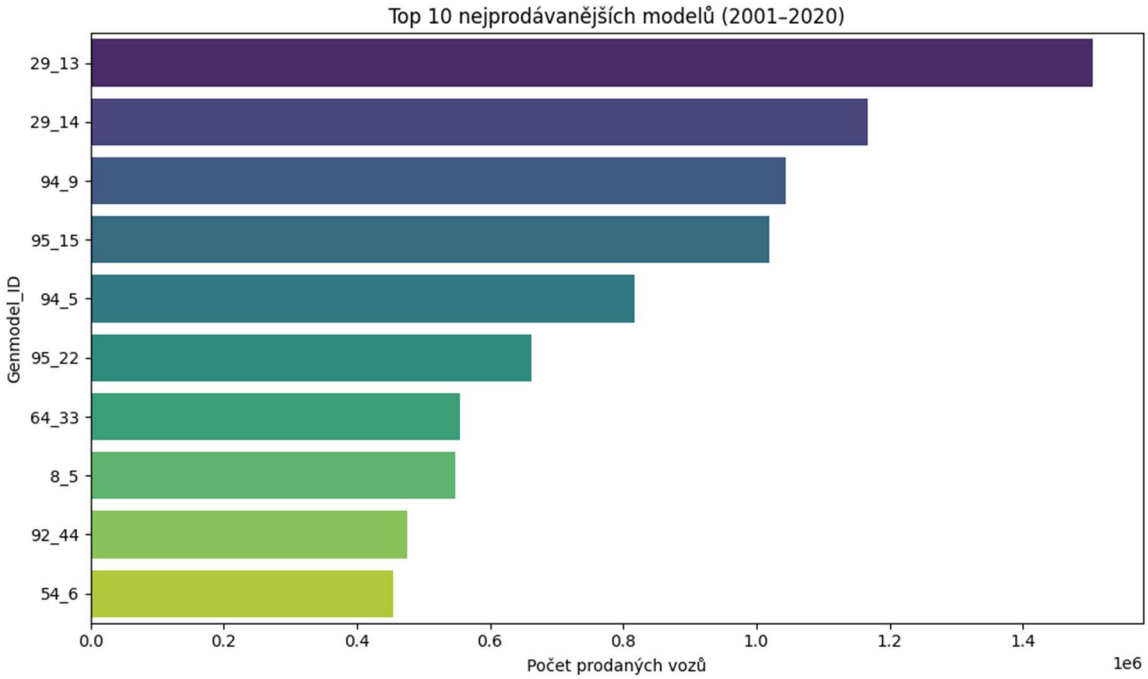
Každý záznam odpovídá jednomu modelu (Genmodel_ID) a zahrnuje název značky (Maker), modelu (Genmodel) a počty registrovaných kusů v jednotlivých letech. V MongoDB je tento dataset ukládán do kolekce *sales*, kde jsou jednotlivé roky reprezentovány jako klíče ve vnořeném dokumentu.

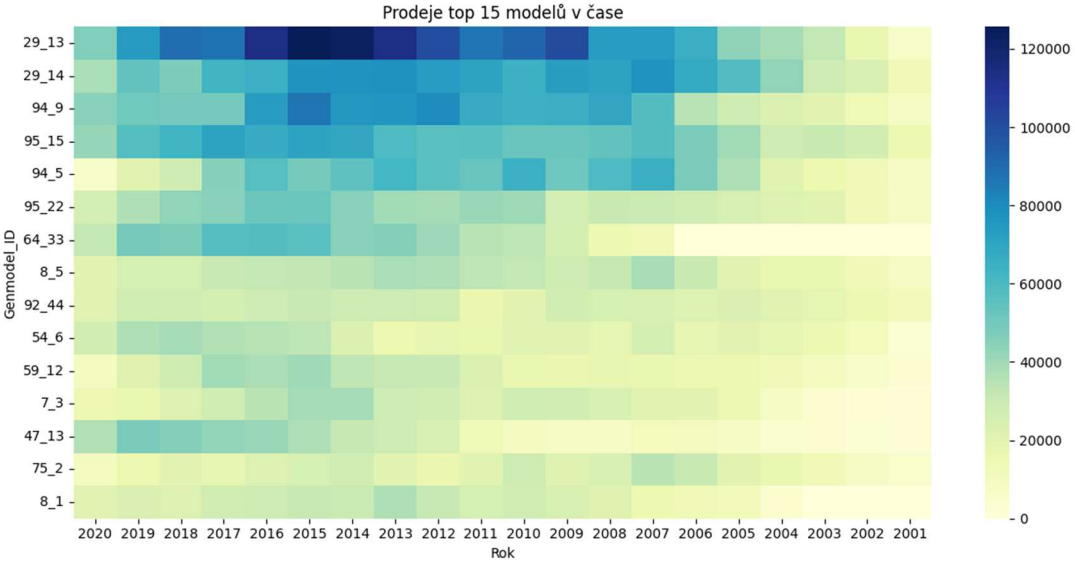
6.4.1 Souhrnná statistika a přehled

- Počet záznamů: 773
- Počet sloupců: 24
- Počet chybějících hodnot: 0
- Sledované období: 2001–2020
- Celkový počet registrovaných vozů: 33 412 900
- Rok s nejvyšším počtem registrací: 2016 (2 476 613 vozů)
- Rok s nejnižším počtem registrací: 2001 (269 887 vozů)
- Výrazný pokles: v roce 2020 (1 530 118 vozů), pravděpodobně vlivem COVID-19
- Top 3 nejregistrovanější modely celkem:
    - 29_13 – 1 505 740 vozů
    - 29_14 – 1 166 989 vozů
    - 94_9 – 1 043 713 vozů

6.4.2 Vizualizace




Top 10 nejprodávanějších modelů (2001–2020)
Genmodel_ID
29_13
29_14
94_9
95_15
94_5
95_22
64_33
8_5
92_44
54_6
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1e6
Počet prodaných vozů


Prodeje top 15 modelů v čase
Genmodel_ID
29_13
29_14
94_9
95_15
94_5
95_22
64_33
8_5
92_44
54_6
59_12
7_3
47_13
75_2
8_1
2020 2019 2018 2017 2016 2015 2014 2013 2012 2011 2010 2009 2008 2007 2006 2005 2004 2003 2002 2001
Rok
120000
100000
80000
60000
40000
20000
0

6.4.3 Ukázka struktury v dokumentu

6.5 Dataset Wolt

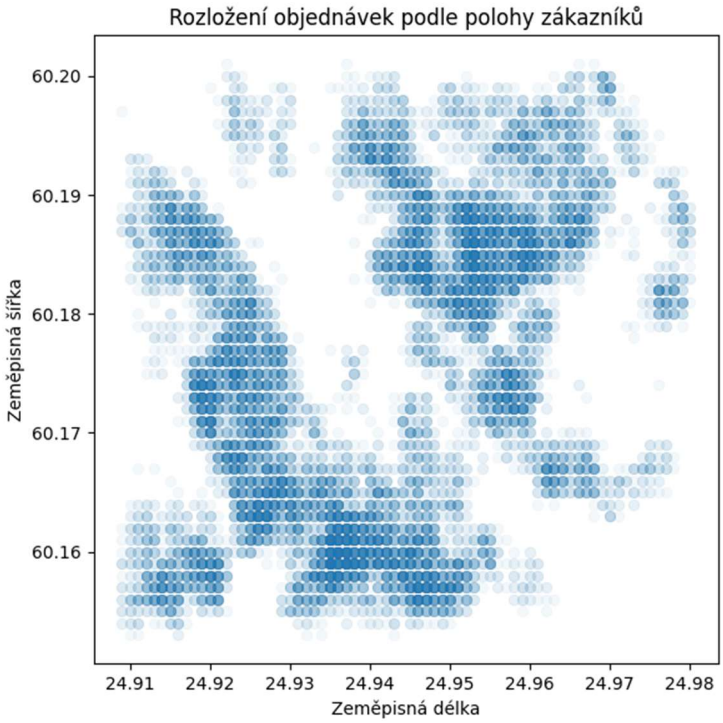
Dataset Wolt pochází ze zdroje na Kaggle a obsahuje záznamy o doručených objednávkách společnosti Wolt. Záznamy zachycují základní informace o objednávce, jako je čas doručení, počet položek, souřadnice zákazníka i restaurace a údaje o počasí.

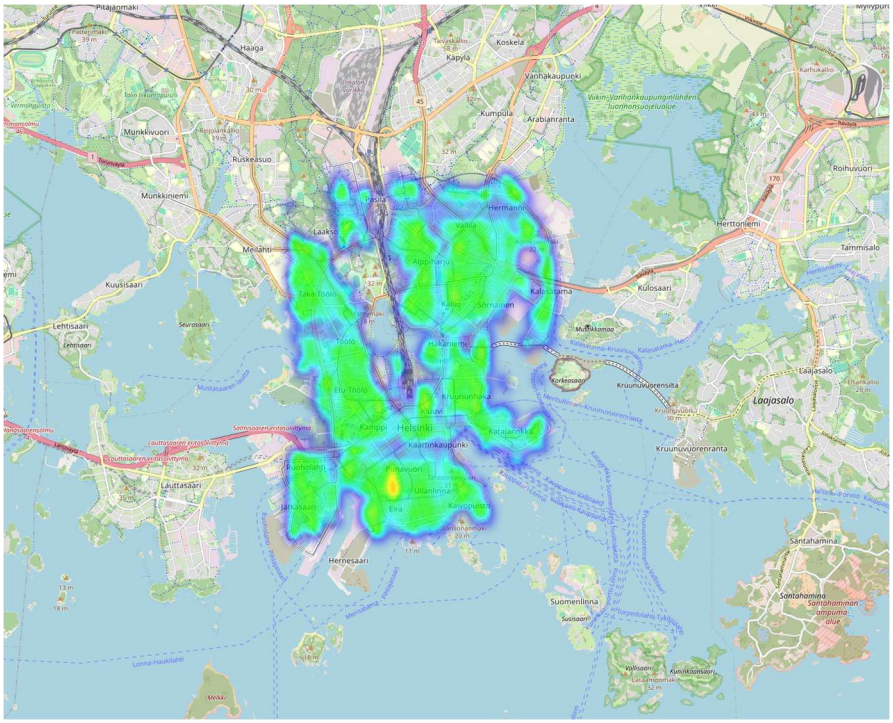
6.5.1 Souhrnná statistika a přehled

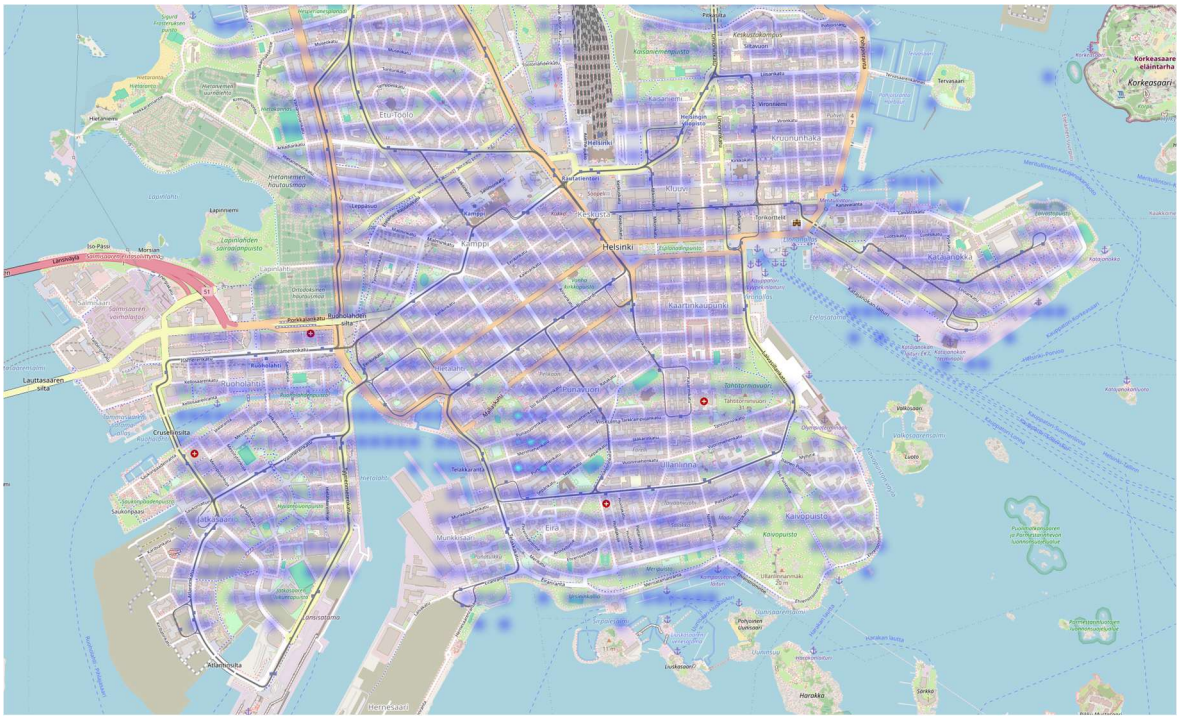
- Počet záznamů:18 706
- Počet sloupců: 16
- Počet chybějících hodnot: 831
- Průměrné zpoždění doručení: –1,2 min
- Mediánové zpoždění: –2,0 min
- Průměrná vzdálenost mezi restaurací a zákazníkem: 1,02 km
- Počet unikátních kombinací souřadnic zákazníků: 2 045 z celkových 18 706 záznamů

Záporné hodnoty zpoždění značí, že doručení proběhlo průměrně o 1–2 minuty dříve, než bylo odhadováno.

6.5.2 Vizualizace

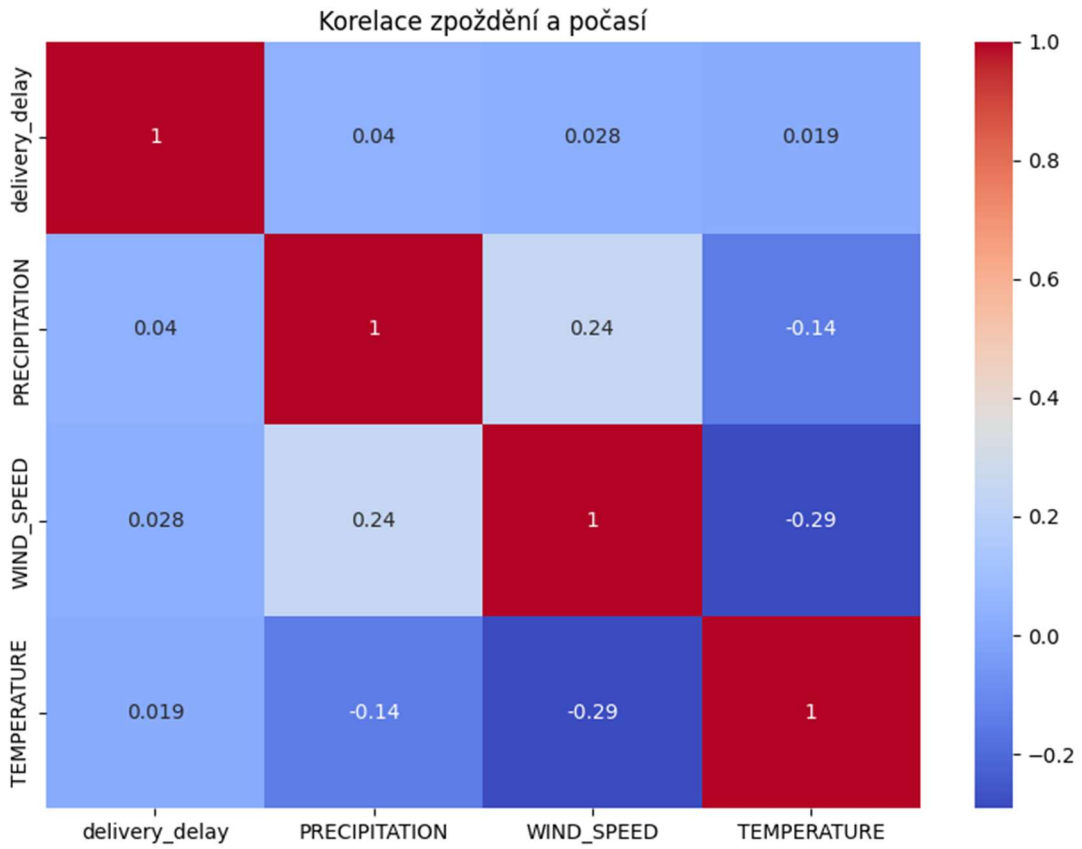




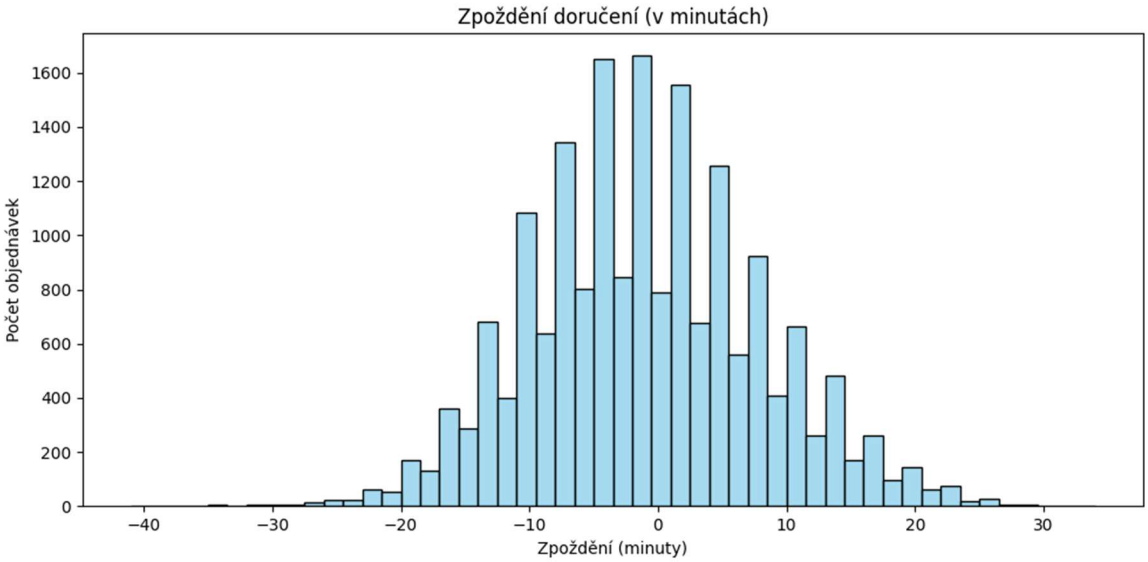


Wolt_user_location_heatmap.html

Rozložení objednávek na mapovém podkladu ukazuje koncentraci v centru Helsinek. Přestože každý bod odpovídá poloze jednoho zákazníka, mapa působí výrazně pravidelně a mřížkovitě. Důvodem je pravděpodobně zaokrouhlení nebo agregace GPS souřadnic z důvodu anonymizace. Tuto hypotézu podporuje i skutečnost, že dataset obsahuje pouze 2 045 unikátních souřadnic z celkových 18 706 záznamů.



Korelační analýza mezi zpožděním doručení a meteorologickými faktory ukazuje velmi slabé vazby. Vliv počasí na zpoždění je prakticky nulový. To naznačuje, že v tomto datasetu počasí nebylo významným faktorem ovlivňujícím doručovací časy.



Histogram ukazuje rozložení zpoždění mezi reálným a odhadovaným časem doručení. Většina objednávek byla doručena včas nebo o několik minut dříve, přičemž nejčastější zpoždění se pohybovalo kolem –2 minut. Rozdělení je relativně symetrické, což ukazuje na konzistentní plánování i plnění časových odhadů.

6.5.3 Ukázka struktury v dokumentu


```
_id: ObjectId('682b28bc961ae67c3326a5f0')
TIMESTAMP : "2020-08-01 06:07:00.000"
ACTUAL_DELIVERY_MINUTES - ESTIMATED_DELIVERY_MINUTES : -19
ITEM_COUNT : 1
USER_LAT : 60.158
USER_LONG : 24.946
VENUE_LAT : 60.16
VENUE_LONG : 24.946
ESTIMATED_DELIVERY_MINUTES : 29
ACTUAL_DELIVERY_MINUTES : 10
CLOUD_COVERAGE : 0
TEMPERATURE : 15
WIND_SPEED : 3.53644
PRECIPITATION : 0
```

```
_id: ObjectId('682b28bc961ae67c3326a5f8')
TIMESTAMP : "2020-08-01 07:59:00.000"
ACTUAL_DELIVERY_MINUTES - ESTIMATED_DELIVERY_MINUTES : 0
ITEM_COUNT : 1
USER_LAT : 60.182
USER_LONG : 24.955
VENUE_LAT : 60.178
VENUE_LONG : 24.949
ESTIMATED_DELIVERY_MINUTES : 24
ACTUAL_DELIVERY_MINUTES : 24
CLOUD_COVERAGE : 0
TEMPERATURE : 16.7
WIND_SPEED : 3.52267
PRECIPITATION : 0
```

6.6 Předzpracování dat

Data byla před importem do MongoDB očištěna pomocí Python skriptu *clean_valid_data.py*. Hlavní kroky zahrnovaly:

- odstranění mezer v názvech sloupců
- odstranění duplicitních záznamů podle klíčových identifikátorů
- odstranění zbytečných sloupců
- transformaci tabulky *Sales_table* z wide do long formátu (roky jsou uloženy jako klíče vnořeného dokumentu)
- uložení očištěných souborů do složky Data/Cleaned.

Následně byla data importována do MongoDB pomocí jednotlivých skriptů (*import_*.py*), které podporují dávkové vkládání záznamů, logování chyb a základní validaci typů. Import probíhá do kolekcí *cars, car_prices, sales, photos* a *wolt_data* v databázi *RampaBase*. Pro usnadnění opakovaného nasazení slouží i centrální spouštěcí skript *import_all.py*.

7 Dotazy

7.1 Práce s daty

7.1.1 InsertOne s validací a indexem

```
db.cars.insertOne({
  Adv_ID: "ADV_TEST_999",
  Genmodel_ID: "GEN_999",
  Maker: "TestBrand",
  Price: 19999,
  Body: "Hatchback"
})
```

Výklad:
Příkaz insertOne vkládá nový dokument. MongoDB ověřuje, že obsahuje požadovaná pole definovaná ve validačním schématu. V tomto případě se nová data týkají testovacího inzerátu/vozidla. Testuje integritu vložených dat dle validačního schéma.

Výstup:

```
{
  acknowledged: true,
  insertedId: ObjectId('682c5d66b59aa8271dd861e0')
}
```

7.1.2 UpdateMany s podmínkou

Změna značky z „BMW“ na „Bayerische Motoren Werke AG“ ve všech dokumentech kolekce cars.

```
db.cars.updateMany(
  { Maker: "BMW" },
  { $set: { Maker: "Bayerische Motoren Werke AG" } }
)
```

Výklad:
Hromadná aktualizace dokumentů. updateMany změní hodnoty všem odpovídajícím dokumentům.

Výstup:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 17194,
  modifiedCount: 17194,
  upsertedCount: 0
}
```

Výpis:

```
{
    _id: ObjectId('682c5718bfd84d0bb46931dc'),
    Maker: 'Bayerische Motoren Werke AG',
    Genmodel: 'M4',
    Genmodel_ID: '8_31',
    Adv_ID: '8_31$$26',
    Adv_year: 2018,
    Adv_month: 7,
    Color: 'White',
    Reg_year: 2018,
    Bodytype: 'Coupe',
    Runned_Miles: '3231',
    Engin_size: '3.0L',
    Gearbox: 'Automatic',
    Fuel_type: 'Petrol',
    Price: 69995,
    Engine_power: NaN,
    Annual_Tax: ' 140*',
    Wheelbase: NaN,
    Height: NaN,
    Width: NaN,
    Length: NaN,
    Average_mpg: NaN,
    Top_speed: NaN,
    Seat_num: 4,
    Door_num: 2
  }
```

7.1.3 DeleteMany s podmínkou

```
db.cars.deleteMany({ Adv_ID: /^TEST/ })
```

Výklad:

Použití regulárního výrazu pro identifikaci a smazání všech záznamů, které byly použity pro testování.

Výstup:

```
{ acknowledged: true, deletedCount: 4 }
```

7.1.4 ReplaceOne s kontrolou integrity

```
db.car_prices.replaceOne(
  {
    Genmodel_ID: "80_5",
    Year: 2009
  },
  {
    Genmodel_ID: "80_5",
    Year: 2009,
    Genmodel: "Octavia II Facelift",
    Entry_price: 17500
  }
)
```

Test validačního schéma → příkaz neprojde, jelikož porušil validaci Entry_price:

```
MongoServerError: Document failed validation
Additional information: {
  failingDocumentId: ObjectId('682c571f89238e8b153f5cf9'),
  details: {
    operatorName: '$jsonSchema',
    schemaRulesNotSatisfied: [
      {
        operatorName: 'properties',
        propertiesNotSatisfied: [
          {
            propertyName: 'Entry_price',
            details: [
              {
                operatorName: 'bsonType',
                specifiedAs: { bsonType: 'double' },
                reason: 'type did not match',
                consideredValue: 17500,
                consideredType: 'int'
              }
            ]
          }
        ]
      }
    ]
  }
}
```

Validní příkaz:

```
db.car_prices.replaceOne(
  {
    Genmodel_ID: "80_5",
    Year: 2009
  },
  {
    Genmodel_ID: "80_5",
    Year: 2009,
    Genmodel: "Octavia II Facelift",
    Entry_price: 17500.5
  }
)
```

Výstup:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

7.1.5 Merge pomocí $merge

```
db.cars.aggregate([
  {
    $group: {
      _id: "$Maker",
      total_ads: { $sum: 1 }
    }
  },
  {
    $merge: {
      into: "car_counts_by_maker",
      whenMatched: "merge",
      whenNotMatched: "insert"
    }
  }
])
```

Výklad:

- **$group:** seskupí dokumenty podle značky a spočítá počet inzerátů
- **$merge:** výsledek uloží do nové kolekce car_counts_by_maker, pokud značka existuje, sloučí záznamy, jinak je vloží.

Ukázka dat v nové kolekci:

```
{
  _id: 'Peugeot',
  total_ads: 10883
}
{
  _id: 'Opel',
  total_ads: 4
}
```

7.1.6 Rename pole pomocí $set + $unset

```
db.cars.updateMany(
  { Top_speed: { $exists: true } },
  [
    { $set: { TopSpeed: "$Top_speed" } },
    { $unset: "Top_speed" }
  ]
)
```

Přejmenování pole Top_Speed na TopSpeed.

Výklad:

- Pomocí $set překopíruje hodnotu původního parametru do nového pole TopSpeed
- $unset pak starý název odstraní

Výstup:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 268255,
  modifiedCount: 268255,
  upsertedCount: 0
}
```

7.2 Agregační funkce

7.2.1 Průměrná cena aut podle roku

```
db.car_prices.aggregate([
  { $match: { Entry_price: { $gt: 0 }, Year: { $exists: true } }},
  { $group: {
    _id: "$Year",
    avgPrice: { $avg: "$Entry_price" },
    count: { $sum: 1 }
  }},
  { $sort: { avgPrice: -1 }}
])
```

Zjišťuje, jak se v jednotlivých letech vyvíjely průměrné ceny aut při uvedení na trh. Agregace podle pole Year.

Výstup:

```
{
  _id: 2021,
  avgPrice: 48947.36,
  count: 25
}
{
  _id: 2020,
  avgPrice: 39809.02197802198,
  count: 91
}
```

7.2.2 Top 5 modelů podle počtu inzerátů

```
db.cars.aggregate([
  { $group: {
    _id: "$Genmodel_ID",
    count: { $sum: 1 }
  }},
  { $sort: { count: -1 }},
  { $limit: 5 }
])
```

Najde nejčastěji inzerované modely aut v databázi. Seskupení podle Genmodel_ID, výstup ukazuje top modely podle četnosti.

Výstup:

```
[
  { _id: '94_9', count: 5569 },
  { _id: '29_14', count: 5179 },
  { _id: '29_13', count: 4761 },
  { _id: '64_14', count: 3427 },
  { _id: '8_39', count: 3372 }
]
```

7.2.3 Úhly pohledu u fotek

```
db.photos.aggregate([
  { $match: { Predicted_viewpoint: { $exists: true, $ne: null } }},
  { $group: {
    _id: "$Predicted_viewpoint",
    count: { $sum: 1 }
  }},
  { $sort: { count: -1 }}
])
```

Distribuce předních/bočních/zadních fotek pomocí group a sort.

Výstup:

```
[
  { _id: 0, count: 240344 },
  { _id: 180, count: 231717 },
  { _id: 225, count: 213611 },
  { _id: 90, count: 195121 },
  { _id: 315, count: 143035 },
  { _id: 270, count: 142216 },
  { _id: 45, count: 139732 },
  { _id: 135, count: 135909 },
  { _id: 360, count: 9247 }
]
```

7.2.4 Modely s nejvyšším počtem nových registrací

```
db.sales.aggregate([
  { $project: {
    Genmodel_ID: 1,
    total_sales: {
      $reduce: {
        input: { $objectToArray: "$sales" },
        initialValue: 0,
        in: { $add: ["$$value", "$$this.v"] }
      }
    }
  }},
  { $group: {
    _id: "$Genmodel_ID",
    total: { $sum: "$total_sales" }
  }},
  { $sort: { total: -1 }},
  { $limit: 5 }
])
```

Spočítá celkový počet registrací nových vozidel podle modelu a vrátí top 5 nejčastěji registrovaných.

Výstup:

```
[
  { _id: '29_13', total: 1505740 },
  { _id: '29_14', total: 1166989 },
  { _id: '94_9', total: 1043713 },
  { _id: '95_15', total: 1018866 },
  { _id: '94_5', total: 816382 }
]
```

7.2.5 Rozdělení aut podle cenových hladin

```
db.car_prices.aggregate([
  { $match: { Entry_price: { $gt: 0 } }},
  { $bucket: {
    groupBy: "$Entry_price",
    boundaries: [0, 10000, 20000, 40000, 60000, 100000],
    default: "100000+",
    output: {
      count: { $sum: 1 },
      avgPrice: { $avg: "$Entry_price" }
    }
  }}
])
```

Rozdělí vozidla do cenových kategorií podle vstupní ceny a spočítá jejich počet i průměrnou cenu v každé kategorii.

Výstup:

```
[
  { _id: 0, count: 1052, avgPrice: 8124.656844106464 },
  { _id: 10000, count: 2665, avgPrice: 15025.607879924954 },
  { _id: 20000, count: 1687, avgPrice: 26853.83461766449 },
  { _id: 40000, count: 396, avgPrice: 49501.81565656565 },
  { _id: 60000, count: 228, avgPrice: 76031.94736842105 },
  { _id: '100000+', count: 305, avgPrice: 181517.69508196722 }
]
```

7.2.6 Top značky podle počtu inzerátů + průměrná cena

```
db.cars.aggregate([
  { $match: { Maker: { $ne: null }, Price: { $gt: 0 } }},
  { $group: {
    _id: "$Maker",
    avgPrice: { $avg: "$Price" },
    count: { $sum: 1 }
  }},
  { $sort: { count: -1 }},
  { $limit: 5 }
])
```

Najde 5 nejčastěji inzerovaných značek vozidel a u každé spočítá průměrnou cenu z inzerátů.

Výstup:

```
[
  { _id: 'Ford', avgPrice: 8748.309054460407, count: 26937 },
  { _id: 'Audi', avgPrice: 18958.66164912748, count: 22521 },
  { _id: 'Vauxhall', avgPrice: 6646.687518603036, count: 20158 },
  { _id: 'Volkswagen', avgPrice: 11563.312684038003, count: 17999 },
  { _id: 'BMW', avgPrice: 21320.90915321827, count: 16049 }
]
```

7.2.7 Rozložení vozidel podle karoserie (Bodytype)

```
db.cars.aggregate([
  { $match: { Bodytype: { $exists: true, $ne: null } }},
  { $group: {
    _id: "$Bodytype",
    count: { $sum: 1 },
    avgPrice: { $avg: "$Price" }
  }},
  { $sort: { count: -1 } }
])
```

Výklad:

- Skupiny podle typu karoserie: např. Hatchback, SUV, Sedan, Coupe…
- Spočítá počet inzerátů daného typu
- Spočítá průměrnou cenu

Výstup:

```
[
  { _id: 'Hatchback', count: 103768, avgPrice: NaN },
  { _id: 'SUV', count: 64396, avgPrice: 20089.21990496304 },
  { _id: 'Saloon', count: 22583, avgPrice: NaN },
  { _id: 'MPV', count: 22507, avgPrice: NaN },
  { _id: 'Coupe', count: 17082, avgPrice: NaN },
  { _id: 'Estate', count: 16797, avgPrice: NaN },
  { _id: 'Convertible', count: 13206, avgPrice: NaN },
  { _id: 'Pickup', count: 5268, avgPrice: 18969.780561883068 },
  { _id: NaN, count: 954, avgPrice: NaN },
  { _id: 'Combi Van', count: 657, avgPrice: 20199.28310502283 },
  { _id: 'Panel Van', count: 477, avgPrice: 16562.989517819708 },
  { _id: 'Minibus', count: 229, avgPrice: 21955.152838427948 },
  { _id: 'Car Derived Van', count: 119, avgPrice: 11348.042016806723 },
  { _id: 'Limousine', count: 93, avgPrice: 58729.04301075269 },
  { _id: 'Window Van', count: 88, avgPrice: 26760.840909090908 },
  { _id: 'Camper', count: 24, avgPrice: 59321.5 },
  { _id: 'Manual', count: 5, avgPrice: 19121.4 },
  { _id: 'Chassis Cab', count: 1, avgPrice: 15000 },
  { _id: 'Tipper', count: 1, avgPrice: 2995 }
]
```

7.3 Embadded dokumenty

7.3.1 Převod embedded let do pole pomocí $objectToArray

```
db.sales.aggregate([
  { $match: { Maker: /AUDI/, Genmodel: /S6/ }},
  { $project: {
    Genmodel_ID: 1,
    yearsArray: { $objectToArray: "$sales" }
  }}
])
```

Výklad:

- $objectToArray převede embedded dokument na pole [{k: "2018", v: 200}]
- Tím lze s roky snadno dál pracovat: filtrovat, mapovat, sčítat.
- Je vhodné převádět objekt na pole, jelikož MongoDB neumí přímo iterovat přes klíče objektu.

Výstup:

```
[
  {
    _id: ObjectId('682c75413527aa116f8ff9ab'),
    Genmodel_ID: '7_31',
    yearsArray: [
      { k: '2018', v: 200 },
      { k: '2017', v: 633 },
      { k: '2016', v: 649 },
      { k: '2015', v: 622 },
      { k: '2014', v: 446 },
      { k: '2013', v: 126 },
      { k: '2010', v: 55 },
      { k: '2009', v: 200 },
      { k: '2020', v: 663 },
      { k: '2008', v: 218 },
      { k: '2005', v: 26 },
      { k: '2004', v: 132 },
      { k: '2003', v: 290 },
      { k: '2002', v: 38 }
    ]
  },
  {
    _id: ObjectId('682c75413527aa116f8ff9b2'),
    Genmodel_ID: '7_41',
    yearsArray: [
      { k: '2019', v: 117 }, { k: '2018', v: 151 },
      { k: '2017', v: 165 }, { k: '2016', v: 117 },
      { k: '2015', v: 96 },  { k: '2014', v: 79 },
      { k: '2013', v: 88 },  { k: '2012', v: 61 },
      { k: '2011', v: 1 },   { k: '2010', v: 28 },
      { k: '2009', v: 17 },  { k: '2020', v: 84 },
      { k: '2008', v: 41 },  { k: '2007', v: 92 },
      { k: '2006', v: 100 }, { k: '2004', v: 4 },
      { k: '2003', v: 19 },  { k: '2002', v: 31 },
      { k: '2001', v: 29 }
    ]
  }
]
```

7.3.2 Sečtení všech registrací přes všechny roky

```
db.sales.aggregate([
  { $project: {
    totalSales: {
      $reduce: {
        input: { $objectToArray: "$sales" },
        initialValue: 0,
        in: { $add: ["$$value", "$$this.v"] }
      }
    }
  }},
  { $group: {
    _id: null,
    totalRegistrations: { $sum: "$totalSales" }
  }}
])
```

Výklad:

- $project vytvoří pole totalSales pro každý dokument (model).
- $reduce projde pole a sečte všechny hodnoty v → získá celkový počet registrací pro daný model.
- $group sčítá totalSales ze všech dokumentů, výsledkem je celkový počet všech registrací v celé kolekci.

Výstup:

```
[ { _id: null, totalRegistrations: 31542071 } ]
```

7.3.3 Seznam registrací pro daný model jako jednotlivé řádky

```
db.sales.aggregate([
  { $match: { Genmodel_ID: "2_4" }},
  { $project: {
    Genmodel_ID: 1,
    sales: { $objectToArray: "$sales" }
  }},
  { $unwind: "$sales" },
  { $project: {
    Genmodel_ID: 1,
    year: "$sales.k",
    registrations: "$sales.v"
  }}
])
```

Dotaz zobrazí všechny registrace modelu 2_4 jako jednotlivé záznamy s uvedením roku a hodnoty. Výstup je vhodný pro další agregace nebo vizualizaci.

Výstup:

```
[
  {
    _id: ObjectId('682c75413527aa116f8ff970'),
    Genmodel_ID: '2_4',
    year: '2019',
    registrations: 2866
  },
  {
    _id: ObjectId('682c75413527aa116f8ff970'),
    Genmodel_ID: '2_4',
    year: '2018',
    registrations: 3907
  },
  {
    _id: ObjectId('682c75413527aa116f8ff970'),
    Genmodel_ID: '2_4',
    year: '2017',
    registrations: 3295
  },
  {
    _id: ObjectId('682c75413527aa116f8ff970'),
    Genmodel_ID: '2_4',
    year: '2016',
    registrations: 3132
  },
  {
    _id: ObjectId('682c75413527aa116f8ff970'),
    Genmodel_ID: '2_4',
    year: '2015',
    registrations: 1612
  },
  {
    _id: ObjectId('682c75413527aa116f8ff970'),
    Genmodel_ID: '2_4',
    year: '2014',
    registrations: 516
  },
  {
    _id: ObjectId('682c75413527aa116f8ff970'),
    Genmodel_ID: '2_4',
    year: '2013',
    registrations: 491
  },
  {
    _id: ObjectId('682c75413527aa116f8ff970'),
    Genmodel_ID: '2_4',
    year: '2012',
    registrations: 165
  },
  {
    _id: ObjectId('682c75413527aa116f8ff970'),
    Genmodel_ID: '2_4',
    year: '2020',
    registrations: 2144
  }
]
```

7.3.4 Průměrná registrace na model za posledních 5 let

```
db.sales.aggregate([
  { $project: {
    Genmodel_ID: 1,
    last5: {
      $filter: {
        input: { $objectToArray: "$sales" },
        as: "year",
        cond: { $gte: ["$$year.k", "2016"] }
      }
    }
  }},
  { $project: {
    Genmodel_ID: 1,
    avgLast5: {
      $avg: { $map: { input: "$last5", as: "y", in: "$$y.v" } }
    }
  }},
  { $sort: { avgLast5: -1 }},
  { $limit: 5 }
])
```

Cílem dotazu je zjistit průměrný počet registrací za posledních 5 let (od roku 2016) pro každý model a následně vybrat pět modelů s nejvyšší hodnotou.

Výstup:

```
[
  {
    _id: ObjectId('682c75413527aa116f8ffa55'),
    Genmodel_ID: '29_13',
    avgLast5: 82356
  },
  {
    _id: ObjectId('682c75413527aa116f8ffc4d'),
    Genmodel_ID: '95_15',
    avgLast5: 59943.6
  },
  {
    _id: ObjectId('682c75413527aa116f8ffc31'),
    Genmodel_ID: '94_9',
    avgLast5: 53859.2
  },
  {
    _id: ObjectId('682c75413527aa116f8ffa56'),
    Genmodel_ID: '29_14',
    avgLast5: 53433.2
  },
  {
    _id: ObjectId('682c75413527aa116f8ffb72'),
    Genmodel_ID: '64_33',
    avgLast5: 48925.4
  }
]
```

7.3.5 Vypsání modelů, které měly v posledních 5 letech pokles registrací

```
db.sales.aggregate([
  {
    $project: {
      Genmodel_ID: 1,
      Genmodel: 1,
      Maker: 1,
      last5: {
        $filter: {
          input: { $objectToArray: "$sales" },
          as: "item",
          cond: { $gte: ["$$item.k", "2016"] }
        }
      }
    }
  },
  {
    $project: {
      Genmodel_ID: 1,
      Genmodel: 1,
      Maker: 1,
      first: { $arrayElemAt: ["$last5", 0] },
      last: { $arrayElemAt: ["$last5", -1] }
    }
  },
  {
    $project: {
      Genmodel_ID: 1,
      Genmodel: 1,
      Maker: 1,
      drop: { $subtract: ["$last.v", "$first.v"] }
    }
  },
  {
    $match: { drop: { $lt: 0 } }
  }
])
```

Modely, které zaznamenaly v posledních 5 letech pokles registrací (rozdíl mezi nejstarším a nejnovějším rokem v daném období je záporný).

Výstup:

```
[
  {
    _id: ObjectId('682c75413527aa116f8ff970'),
    Maker: 'ABARTH',
    Genmodel: 'ABARTH 595',
    Genmodel_ID: '2_4',
    drop: -722
  },
  {
    _id: ObjectId('682c75413527aa116f8ff984'),
    Maker: 'ALFA ROMEO',
    Genmodel: 'ALFA ROMEO GIULIETTA',
    Genmodel_ID: '4_11',
    drop: -40
  },
  {
    _id: ObjectId('682c75413527aa116f8ff989'),
    Maker: 'ALFA ROMEO',
```

```
    Genmodel: 'ALFA ROMEO STELVIO',
    Genmodel_ID: '4_16',
    drop: -480
  },
  {
    _id: ObjectId('682c75413527aa116f8ff98a'),
    Maker: 'ALPINE',
    Genmodel: 'ALPINE A110',
    Genmodel_ID: '101_1',
    drop: -47
  },
  {
    _id: ObjectId('682c75413527aa116f8ff98c'),
    Maker: 'ASTON MARTIN',
    Genmodel: 'ASTON MARTIN DB11',
    Genmodel_ID: '6_2',
    drop: -176
  },
  {
    _id: ObjectId('682c75413527aa116f8ff994'),
    Maker: 'ASTON MARTIN',
    Genmodel: 'ASTON MARTIN VANTAGE',
    Genmodel_ID: '6_9',
    drop: -497
  },
  {
    _id: ObjectId('682c75413527aa116f8ff99c'),
    Maker: 'AUDI',
    Genmodel: 'AUDI A7',
    Genmodel_ID: '7_15',
    drop: -843
  },
  {
    _id: ObjectId('682c75413527aa116f8ff99d'),
    Maker: 'AUDI',
    Genmodel: 'AUDI A8',
    Genmodel_ID: '7_16',
    drop: -354
  },
  {
    _id: ObjectId('682c75413527aa116f8ff9aa'),
    Maker: 'AUDI',
    Genmodel: 'AUDI RS5',
    Genmodel_ID: '7_30',
    drop: -245
  },
  {
    _id: ObjectId('682c75413527aa116f8ff9af'),
    Maker: 'AUDI',
    Genmodel: 'AUDI S3',
    Genmodel_ID: '7_35',
    drop: -1301
  },
  {
    _id: ObjectId('682c75413527aa116f8ff9ba'),
    Maker: 'AUDI',
    Genmodel: 'AUDI TTS',
    Genmodel_ID: '7_48',
    drop: -1287
  },
  {
    _id: ObjectId('682c75413527aa116f8ff9c2'),
    Maker: 'BMW',
    Genmodel: 'BMW 1 SERIES',
    Genmodel_ID: '8_1',
    drop: -2860
  },
```

```
  {
    _id: ObjectId('682c75413527aa116f8ff9c4'),
    Maker: 'BMW',
    Genmodel: 'BMW 3 SERIES',
    Genmodel_ID: '8_5',
    drop: -5277
  },
  {
    _id: ObjectId('682c75413527aa116f8ff9c6'),
    Maker: 'BMW',
    Genmodel: 'BMW 5 SERIES',
    Genmodel_ID: '8_9',
    drop: -6988
  },
  {
    _id: ObjectId('682c75413527aa116f8ff9c7'),
    Maker: 'BMW',
    Genmodel: 'BMW 6 SERIES',
    Genmodel_ID: '8_11',
    drop: -808
  },
  {
    _id: ObjectId('682c75413527aa116f8ff9d4'),
    Maker: 'BMW',
    Genmodel: 'BMW X3',
    Genmodel_ID: '8_37',
    drop: -8335
  },
  {
    _id: ObjectId('682c75413527aa116f8ff9d6'),
    Maker: 'BMW',
    Genmodel: 'BMW X5',
    Genmodel_ID: '8_39',
    drop: -3710
  },
  {
    _id: ObjectId('682c75413527aa116f8ff9fd'),
    Maker: 'CITROEN',
    Genmodel: 'CITROEN C4',
    Genmodel_ID: '18_12',
    drop: -5290
  },
  {
    _id: ObjectId('682c75413527aa116f8ffa09'),
    Maker: 'CITROEN',
    Genmodel: 'CITROEN SPACETOURER',
    Genmodel_ID: '18_27',
    drop: -488
  },
  {
    _id: ObjectId('682c75413527aa116f8ffa25'),
    Maker: 'DS',
    Genmodel: 'DS DS3',
    Genmodel_ID: '21_1',
    drop: -430
  }
]
```

7.3.6 Rozbalení embedded dokumentu years na roky a součty

```
db.sales.aggregate([
  { $project: {
    Genmodel_ID: 1,
    sales: { $objectToArray: "$sales" }
  }},
  { $unwind: "$sales" },
  { $group: {
    _id: "$sales.k",
    total: { $sum: "$sales.v" }
  }},
  { $sort: { _id: 1 } }
])
```

Vývoj celkového počtu registrací automobilů v UK za celé období.

Výstup:

```
[
  { _id: '2001', total: 269887 },
  { _id: '2002', total: 436447 },
  { _id: '2003', total: 622347 },
  { _id: '2004', total: 818247 },
  { _id: '2005', total: 1024944 },
  { _id: '2006', total: 1256941 },
  { _id: '2007', total: 1527848 },
  { _id: '2008', total: 1500885 },
  { _id: '2009', total: 1578106 },
  { _id: '2010', total: 1674012 },
  { _id: '2011', total: 1648030 },
  { _id: '2012', total: 1789436 },
  { _id: '2013', total: 2032197 },
  { _id: '2014', total: 2253307 },
  { _id: '2015', total: 2421219 },
  { _id: '2016', total: 2476613 },
  { _id: '2017', total: 2290444 },
  { _id: '2018', total: 2212385 },
  { _id: '2019', total: 2178658 },
  { _id: '2020', total: 1530118 }
]
```

7.4 Indexy a výkon

7.4.1 Vytvoření složeného indexu pro dotazy podle značky a ceny.

```
db.cars.createIndex({ Maker: 1, Price: -1 })
```

Výklad:

Složený index urychlí dotazy typu „auta určité značky seřazená podle ceny“. Umožňuje efektivní využití při filtrování i řazení.

7.4.2 Vyhodnocení dotazu pomocí explain()

```
db.car_prices.find({ Genmodel_ID: "77_2" }).explain("executionStats")
```

Výklad:

- *explain("executionStats")* zobrazí detailní informace o provedení dotazu, zda byl použit index, kolik dokumentů bylo prohledáno a kolik vráceno.
- Hodnota *totalKeysExamined* ukazuje, kolik indexovaných záznamů bylo prohlédnuto.
- Hodnota *totalDocsExamined* ukazuje, kolik dokumentů bylo skutečně načteno z databáze.
- V projektu slouží tento příkaz ke kontrole, zda jsou správně vytvořené indexy skutečně využívány (např. na poli *Genmodel_ID*).

Výstup:

```
{
  queryPlanner: {
    winningPlan: {
      stage: 'SINGLE_SHARD',
      shards: [
        {
          explainVersion: '1',
          shardName: 'rs-shard-03',
          connectionString: 'rs-shard-03/shard03-a:27017,shard03-b:27017,shard03-
c:27017',
          serverInfo: {
            host: 'cc03dd45b971',
            port: 27017,
            version: '8.0.8',
            gitVersion: '7f52660c14217ed2c8d3240f823a2291a4fe6abd'
          },
          namespace: 'RampaBase.car_prices',
          parsedQuery: { Genmodel_ID: { '$eq': '77_2' } },
          indexFilterSet: false,
          queryHash: '2B1ECF09',
          planCacheShapeHash: '2B1ECF09',
          planCacheKey: 'C6462F7B',
          optimizationTimeMillis: 0,
          maxIndexedOrSolutionsReached: false,
          maxIndexedAndSolutionsReached: false,
          maxScansToExplodeReached: false,
          prunedSimilarIndexes: false,
```

```
          winningPlan: {
            isCached: false,
            stage: 'FETCH',
            filter: { Genmodel_ID: { '$eq': '77_2' } },
            inputStage: {
              stage: 'IXSCAN',
              keyPattern: { Genmodel_ID: 'hashed' },
              indexName: 'Genmodel_ID_hashed',
              isMultiKey: false,
              isUnique: false,
              isSparse: false,
              isPartial: false,
              indexVersion: 2,
              direction: 'forward',
              indexBounds: {
                Genmodel_ID: [ '[-4283814741635797331, -4283814741635797331]' ]
              }
            }
          },
          rejectedPlans: []
        }
      ]
    }
  },
  executionStats: {
    nReturned: 4,
    executionTimeMillis: 1,
    totalKeysExamined: 4,
    totalDocsExamined: 4,
    executionStages: {
      stage: 'SINGLE_SHARD',
      nReturned: 4,
      executionTimeMillis: 1,
      totalKeysExamined: 4,
      totalDocsExamined: 4,
      totalChildMillis: Long('0'),
      shards: [
        {
          shardName: 'rs-shard-03',
          executionSuccess: true,
          nReturned: 4,
          executionTimeMillis: 0,
          totalKeysExamined: 4,
          totalDocsExamined: 4,
          executionStages: {
            isCached: false,
            stage: 'FETCH',
            filter: { Genmodel_ID: { '$eq': '77_2' } },
            nReturned: 4,
            executionTimeMillisEstimate: 0,
            works: 5,
            advanced: 4,
            needTime: 0,
            needYield: 0,
            saveState: 0,
            restoreState: 0,
            isEOF: 1,
            docsExamined: 4,
            alreadyHasObj: 0,
            inputStage: {
              stage: 'IXSCAN',
              nReturned: 4,
              executionTimeMillisEstimate: 0,
              works: 5,
              advanced: 4,
              needTime: 0,
              needYield: 0,
```

```
              saveState: 0,
              restoreState: 0,
              isEOF: 1,
              keyPattern: { Genmodel_ID: 'hashed' },
              indexName: 'Genmodel_ID_hashed',
              isMultiKey: false,
              isUnique: false,
              isSparse: false,
              isPartial: false,
              indexVersion: 2,
              direction: 'forward',
              indexBounds: {
                Genmodel_ID: [ '[-4283814741635797331, -4283814741635797331]' ]
              },
              keysExamined: 4,
              seeks: 1,
              dupsTested: 0,
              dupsDropped: 0
            }
          }
        }
      ]
    }
  },
  queryShapeHash:
'94949557510653B52BF2CE377B1DF2CE05BFAD4C3DAE8518761CA32DAE5CF7E3',
  serverInfo: {
    host: 'c9f222b5b2b5',
    port: 27017,
    version: '8.0.8',
    gitVersion: '7f52660c14217ed2c8d3240f823a2291a4fe6abd'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600,
    internalQueryFrameworkControl: 'trySbeRestricted',
    internalQueryPlannerIgnoreIndexWithCollationForRegex: 1
  },
  command: {
    find: 'car_prices',
    filter: { Genmodel_ID: '77_2' },
    lsid: { id: UUID('4abdf7d3-a3e4-4b8a-b959-01dd77d7b25b') },
    '$clusterTime': {
      clusterTime: Timestamp({ t: 1747750867, i: 1 }),
      signature: {
        hash: Binary.createFromBase64('pLzClrcfVvjJdOzM6eg32wvBpBc=', 0),
        keyId: Long('7506503454924210199')
      }
    },
    '$db': 'RampaBase'
  },
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1747750873, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('3+hnKq9gCUXCpiNC9SIxypRrykY=', 0),
      keyId: Long('7506503454924210199')
    }
  },
  operationTime: Timestamp({ t: 1747750873, i: 1 })
}
```

Výstup ukazuje následující:

- MongoDB použilo index Genmodel_ID_hashed, což potvrzuje sekce inputStage.stage: IXSCAN a indexName: Genmodel_ID_hashed.
- Dotaz byl zpracován pouze na jednom shardu rs-shard-03, jak je uvedeno v sekci shards[0].shardName.
- Výsledkem bylo nalezení 4 dokumentů (nReturned: 4) s minimální zátěží: totalKeysExamined: 4, totalDocsExamined: 4.
- Celkový čas zpracování dotazu byl 1 ms (executionTimeMillis: 1).
- Použitý plán (winningPlan) zahrnoval nejprve skenování indexu (IXSCAN), následované načtením dokumentů (FETCH).

7.4.3 Použití hint() pro vynucení indexu

```
db.car_prices.find({ Genmodel_ID: "77_2" }).hint("Genmodel_ID_hashed")
```

Výklad:

- hint() slouží k vynucení konkrétního indexu, i když by MongoDB jinak mohlo použít jiný plán vykonání.
- V tomto dotazu je explicitně určeno, že má MongoDB použít index vytvořený na poli Genmodel_ID_hashed.
- Tento příkaz je užitečný zejména při ladění výkonu dotazů nebo při testování, zda určitý index dává lepší výsledky než jiný.

Výstup:

```
[
  {
    _id: ObjectId('682c75406f6ffa3ba7764a53'),
    Maker: 'Rolls-Royce',
    Genmodel: 'Dawn',
    Genmodel_ID: '77_2',
    Year: 2015,
    Entry_price: 264000
  },
  {
    _id: ObjectId('682c75406f6ffa3ba7764a54'),
    Maker: 'Rolls-Royce',
    Genmodel: 'Dawn',
    Genmodel_ID: '77_2',
    Year: 2016,
    Entry_price: 264000
  },
  {
    _id: ObjectId('682c75406f6ffa3ba7764a55'),
    Maker: 'Rolls-Royce',
    Genmodel: 'Dawn',
    Genmodel_ID: '77_2',
    Year: 2017,
    Entry_price: 264000
  },
  {
    _id: ObjectId('682c75406f6ffa3ba7764a56'),
    Maker: 'Rolls-Royce',
    Genmodel: 'Dawn',
    Genmodel_ID: '77_2',
    Year: 2018,
    Entry_price: 275240
  }
]
```

7.4.4 Fulltextový index na všechny atributy

```
db.cars.createIndex({ "$**": "text" })
```

Výklad:

- Tento příkaz vytvoří tzv. wildcard textový index, indexuje všechna textová pole ve všech dokumentech v kolekci.
- Díky tomu je možné provádět $text dotazy bez nutnosti specifikovat konkrétní pole.
- Výhodné je to zejména při rychlém fulltextovém hledání napříč strukturou dokumentů (např. db.cars.find({ $text: { $search: "SKODA Octavia" } })).

7.4.5 TTL index pro dočasná data

```
db.wolt_data.createIndex(
  { TIMESTAMP_DATE: 1 },
  { expireAfterSeconds: 86400 }
)
```

Výklad:

- TTL indexy umožňují MongoDB automaticky mazat dokumenty po určitém čase.
- expireAfterSeconds: 86400 znamená, že dokument bude smazán 24 hodin po hodnotě v poli TIMESTAMP_DATE.
- Použitelné pouze u polí typu Date. Mongo pravidelně kontroluje dokumenty a maže ty, které expirovaly.

7.4.6 Analýza vhodnosti shard klíče

```
db.adminCommand({
  analyzeShardKey: "RampaBase.cars",
  key: { Adv_ID: "hashed" }
})
```

Výklad:

- analyzeShardKey je administrační příkaz, který analyzuje rozložení hodnot v daném poli a zjistí, zda je vhodné použít jej jako shard key.
- Pomáhá zjistit, jak dobře by se podle daného klíče dělily chunky.
- V tomto případě zkoumáme hashovaný Adv_ID, který je unikátní pro každý inzerát/vozidlo a vhodný pro rovnoměrné shardování.
- Výstup obsahuje např. počet potenciálních chunků, doporučení a metriky fragmentace.

Souhrn informací z výstupu:

- numDocsTotal: 268255
  - Kolekce cars obsahuje celkem 268 255 dokumentů.
- numDocsSampled: 268255
  - Všechny dokumenty byly použity při analýze.
- numDistinctValues: 268255
  - Každý Adv_ID je unikátní → ideální vlastnost pro shard klíč.
- mostCommonValues: všechny s frekvencí 1
  - Nebyly nalezeny žádné duplicitní hodnoty v Adv_ID.
- avgDocSizeBytes: 485
  - Průměrná velikost dokumentu
- readDistribution a writeDistribution: 0
  - V době analýzy nebyly v kolekci zaznamenány žádné čtecí ani zápisové operace.

7.5 Cluster a konfigurace

7.5.1 Stav shardingu v clusteru

```
sh.status()
```

Výklad:
Zobrazí aktuální konfiguraci shardování, jaké shardy jsou aktivní, které kolekce jsou shardované a jak jsou rozdělené chunky.

Výstup:

```
shardingVersion
{ _id: 1, clusterId: ObjectId('682c751fe2e48580dbb24ea8') }
---
shards
[
  {
    _id: 'rs-shard-01',
    host: 'rs-shard-01/shard01-a:27017,shard01-b:27017,shard01-c:27017',
    state: 1,
    topologyTime: Timestamp({ t: 1747744036, i: 10 }),
    replSetConfigVersion: Long('1')
  },
  {
    _id: 'rs-shard-02',
    host: 'rs-shard-02/shard02-a:27017,shard02-b:27017,shard02-c:27017',
    state: 1,
    topologyTime: Timestamp({ t: 1747744036, i: 28 }),
    replSetConfigVersion: Long('1')
  },
  {
    _id: 'rs-shard-03',
    host: 'rs-shard-03/shard03-a:27017,shard03-b:27017,shard03-c:27017',
    state: 1,
    topologyTime: Timestamp({ t: 1747744036, i: 51 }),
    replSetConfigVersion: Long('1')
  }
]
---
active mongoses
[ { '8.0.8': 2 } ]
---
autosplit
{ 'Currently enabled': 'yes' }
---
balancer
{
  'Currently enabled': 'yes',
  'Failed balancer rounds in last 5 attempts': 0,
  'Currently running': 'no',
  'Migration Results for the last 24 hours': 'No recent migrations'
}
---
shardedDataDistribution
[
  {
    ns: 'RampaBase.photos',
    shards: [
      {
        shardName: 'rs-shard-01',
        numOrphanedDocs: 0,
```

```
        numOwnedDocuments: 483563,
        ownedSizeBytes: 90909844,
        orphanedSizeBytes: 0
      },
      {
        shardName: 'rs-shard-02',
        numOrphanedDocs: 0,
        numOwnedDocuments: 483106,
        ownedSizeBytes: 90823928,
        orphanedSizeBytes: 0
      },
      {
        shardName: 'rs-shard-03',
        numOrphanedDocs: 0,
        numOwnedDocuments: 484263,
        ownedSizeBytes: 91041444,
        orphanedSizeBytes: 0
      }
    ]
  },
  {
    ns: 'RampaBase.wolt_data',
    shards: [
      {
        shardName: 'rs-shard-01',
        numOrphanedDocs: 0,
        numOwnedDocuments: 6291,
        ownedSizeBytes: 2264760,
        orphanedSizeBytes: 0
      },
      {
        shardName: 'rs-shard-02',
        numOrphanedDocs: 0,
        numOwnedDocuments: 6165,
        ownedSizeBytes: 2219400,
        orphanedSizeBytes: 0
      },
      {
        shardName: 'rs-shard-03',
        numOrphanedDocs: 0,
        numOwnedDocuments: 6245,
        ownedSizeBytes: 2248200,
        orphanedSizeBytes: 0
      }
    ]
  },
  {
    ns: 'config.system.sessions',
    shards: [
      {
        shardName: 'rs-shard-01',
        numOrphanedDocs: 0,
        numOwnedDocuments: 2,
        ownedSizeBytes: 270,
        orphanedSizeBytes: 0
      }
    ]
  },
  {
    ns: 'RampaBase.cars',
    shards: [
      {
        shardName: 'rs-shard-01',
        numOrphanedDocs: 0,
        numOwnedDocuments: 89430,
        ownedSizeBytes: 43373550,
        orphanedSizeBytes: 0
```

```
      },
      {
        shardName: 'rs-shard-02',
        numOrphanedDocs: 0,
        numOwnedDocuments: 89401,
        ownedSizeBytes: 43359485,
        orphanedSizeBytes: 0
      },
      {
        shardName: 'rs-shard-03',
        numOrphanedDocs: 0,
        numOwnedDocuments: 89424,
        ownedSizeBytes: 43370640,
        orphanedSizeBytes: 0
      }
    ]
  },
  {
    ns: 'RampaBase.car_prices',
    shards: [
      {
        shardName: 'rs-shard-01',
        numOrphanedDocs: 0,
        numOwnedDocuments: 2005,
        ownedSizeBytes: 228570,
        orphanedSizeBytes: 0
      },
      {
        shardName: 'rs-shard-02',
        numOrphanedDocs: 0,
        numOwnedDocuments: 2232,
        ownedSizeBytes: 254448,
        orphanedSizeBytes: 0
      },
      {
        shardName: 'rs-shard-03',
        numOrphanedDocs: 0,
        numOwnedDocuments: 2096,
        ownedSizeBytes: 238944,
        orphanedSizeBytes: 0
      }
    ]
  },
  {
    ns: 'RampaBase.sales',
    shards: [
      {
        shardName: 'rs-shard-01',
        numOrphanedDocs: 0,
        numOwnedDocuments: 238,
        ownedSizeBytes: 47838,
        orphanedSizeBytes: 0
      },
      {
        shardName: 'rs-shard-02',
        numOrphanedDocs: 0,
        numOwnedDocuments: 269,
        ownedSizeBytes: 52993,
        orphanedSizeBytes: 0
      },
      {
        shardName: 'rs-shard-03',
        numOrphanedDocs: 0,
        numOwnedDocuments: 266,
        ownedSizeBytes: 52668,
        orphanedSizeBytes: 0
      }
```

```
    ]
  }
]
---
databases
[
  {
    database: { _id: 'config', primary: 'config', partitioned: true },
    collections: {
      'config.system.sessions': {
        shardKey: { _id: 1 },
        unique: false,
        balancing: true,
        chunkMetadata: [ { shard: 'rs-shard-01', nChunks: 1 } ],
        chunks: [
          { min: { _id: MinKey() }, max: { _id: MaxKey() }, 'on shard': 'rs-
shard-01', 'last modified': Timestamp({ t: 1, i: 0 }) }
        ],
        tags: []
      }
    }
  },
  {
    database: {
      _id: 'RampaBase',
      primary: 'rs-shard-01',
      version: {
        uuid: UUID('3a5bcb9a-6695-4080-a58c-9e9057e6f8ca'),
        timestamp: Timestamp({ t: 1747744054, i: 3 }),
        lastMod: 1
      }
    },
    collections: {
      'RampaBase.car_prices': {
        shardKey: { Genmodel_ID: 'hashed' },
        unique: false,
        balancing: true,
        chunkMetadata: [
          { shard: 'rs-shard-01', nChunks: 1 },
          { shard: 'rs-shard-02', nChunks: 1 },
          { shard: 'rs-shard-03', nChunks: 1 }
        ],
        chunks: [
          { min: { Genmodel_ID: MinKey() }, max: { Genmodel_ID: Long('-
3074457345618258602') }, 'on shard': 'rs-shard-03', 'last modified': Timestamp({
t: 1, i: 0 }) },
          { min: { Genmodel_ID: Long('-3074457345618258602') }, max: { Genmo-
del_ID: Long('3074457345618258602') }, 'on shard': 'rs-shard-01', 'last modi-
fied': Timestamp({ t: 1, i: 1 }) },
          { min: { Genmodel_ID: Long('3074457345618258602') }, max: { Genmo-
del_ID: MaxKey() }, 'on shard': 'rs-shard-02', 'last modified': Timestamp({ t: 1,
i: 2 }) }
        ],
        tags: []
      },
      'RampaBase.cars': {
        shardKey: { Adv_ID: 'hashed' },
        unique: false,
        balancing: true,
        chunkMetadata: [
          { shard: 'rs-shard-01', nChunks: 1 },
          { shard: 'rs-shard-02', nChunks: 1 },
          { shard: 'rs-shard-03', nChunks: 1 }
        ],
        chunks: [
```

```
          { min: { Adv_ID: MinKey() }, max: { Adv_ID: Long('-
3074457345618258602') }, 'on shard': 'rs-shard-03', 'last modified': Timestamp({
t: 1, i: 0 }) },
          { min: { Adv_ID: Long('-3074457345618258602') }, max: { Adv_ID:
Long('3074457345618258602') }, 'on shard': 'rs-shard-01', 'last modified': Time-
stamp({ t: 1, i: 1 }) },
          { min: { Adv_ID: Long('3074457345618258602') }, max: { Adv_ID: MaxKey()
}, 'on shard': 'rs-shard-02', 'last modified': Timestamp({ t: 1, i: 2 }) }
        ],
        tags: []
      },
      'RampaBase.photos': {
        shardKey: { Image_ID: 'hashed' },
        unique: false,
        balancing: true,
        chunkMetadata: [
          { shard: 'rs-shard-01', nChunks: 1 },
          { shard: 'rs-shard-02', nChunks: 1 },
          { shard: 'rs-shard-03', nChunks: 1 }
        ],
        chunks: [
          { min: { Image_ID: MinKey() }, max: { Image_ID: Long('-
3074457345618258602') }, 'on shard': 'rs-shard-03', 'last modified': Timestamp({
t: 1, i: 0 }) },
          { min: { Image_ID: Long('-3074457345618258602') }, max: { Image_ID:
Long('3074457345618258602') }, 'on shard': 'rs-shard-01', 'last modified': Time-
stamp({ t: 1, i: 1 }) },
          { min: { Image_ID: Long('3074457345618258602') }, max: { Image_ID: Ma-
xKey() }, 'on shard': 'rs-shard-02', 'last modified': Timestamp({ t: 1, i: 2 }) }
        ],
        tags: []
      },
      'RampaBase.sales': {
        shardKey: { Genmodel_ID: 'hashed' },
        unique: false,
        balancing: true,
        chunkMetadata: [
          { shard: 'rs-shard-01', nChunks: 1 },
          { shard: 'rs-shard-02', nChunks: 1 },
          { shard: 'rs-shard-03', nChunks: 1 }
        ],
        chunks: [
          { min: { Genmodel_ID: MinKey() }, max: { Genmodel_ID: Long('-
3074457345618258602') }, 'on shard': 'rs-shard-03', 'last modified': Timestamp({
t: 1, i: 0 }) },
          { min: { Genmodel_ID: Long('-3074457345618258602') }, max: { Genmo-
del_ID: Long('3074457345618258602') }, 'on shard': 'rs-shard-01', 'last modi-
fied': Timestamp({ t: 1, i: 1 }) },
          { min: { Genmodel_ID: Long('3074457345618258602') }, max: { Genmo-
del_ID: MaxKey() }, 'on shard': 'rs-shard-02', 'last modified': Timestamp({ t: 1,
i: 2 }) }
        ],
        tags: []
      },
      'RampaBase.wolt_data': {
        shardKey: { _id: 'hashed' },
        unique: false,
        balancing: true,
        chunkMetadata: [
          { shard: 'rs-shard-01', nChunks: 1 },
          { shard: 'rs-shard-02', nChunks: 1 },
          { shard: 'rs-shard-03', nChunks: 1 }
        ],
        chunks: [
          { min: { _id: MinKey() }, max: { _id: Long('-3074457345618258602') },
'on shard': 'rs-shard-03', 'last modified': Timestamp({ t: 1, i: 0 }) },
```

```
          { min: { _id: Long('-3074457345618258602') }, max: { _id:
Long('3074457345618258602') }, 'on shard': 'rs-shard-01', 'last modified': Time-
stamp({ t: 1, i: 1 }) },
          { min: { _id: Long('3074457345618258602') }, max: { _id: MaxKey() },
'on shard': 'rs-shard-02', 'last modified': Timestamp({ t: 1, i: 2 }) }
        ],
        tags: []
      }
    }
  }
]
```

7.5.2 Rozložení dat mezi shardy

```
sh.getShardedDataDistribution()
```

Výklad:
Vrací informace, kolik dokumentů se nachází na každém shardu pro jednotlivé kolekce. Umožňuje ověřit, že je rozložení dat rovnoměrné.

Výstup:

```
[
  {
    ns: 'config.system.sessions',
    shards: [
      {
        shardName: 'rs-shard-01',
        numOrphanedDocs: 0,
        numOwnedDocuments: 13,
        ownedSizeBytes: 1352,
        orphanedSizeBytes: 0
      }
    ]
  },
  {
    ns: 'RampaBase.photos',
    shards: [
      {
        shardName: 'rs-shard-01',
        numOrphanedDocs: 0,
        numOwnedDocuments: 483563,
        ownedSizeBytes: 90909844,
        orphanedSizeBytes: 0
      },
      {
        shardName: 'rs-shard-02',
        numOrphanedDocs: 0,
        numOwnedDocuments: 483106,
        ownedSizeBytes: 90823928,
        orphanedSizeBytes: 0
      },
      {
        shardName: 'rs-shard-03',
        numOrphanedDocs: 0,
        numOwnedDocuments: 484263,
```

```
        ownedSizeBytes: 91041444,
        orphanedSizeBytes: 0
      }
    ]
  },
  {
    ns: 'RampaBase.wolt_data',
    shards: [
      {
        shardName: 'rs-shard-01',
        numOrphanedDocs: 0,
        numOwnedDocuments: 6291,
        ownedSizeBytes: 2264760,
        orphanedSizeBytes: 0
      },
      {
        shardName: 'rs-shard-02',
        numOrphanedDocs: 0,
        numOwnedDocuments: 6165,
        ownedSizeBytes: 2219400,
        orphanedSizeBytes: 0
      },
      {
        shardName: 'rs-shard-03',
        numOrphanedDocs: 0,
        numOwnedDocuments: 6245,
        ownedSizeBytes: 2248200,
        orphanedSizeBytes: 0
      }
    ]
  },
  {
    ns: 'RampaBase.sales',
    shards: [
      {
        shardName: 'rs-shard-01',
        numOrphanedDocs: 0,
        numOwnedDocuments: 238,
        ownedSizeBytes: 47838,
        orphanedSizeBytes: 0
      },
      {
        shardName: 'rs-shard-02',
        numOrphanedDocs: 0,
        numOwnedDocuments: 269,
        ownedSizeBytes: 52993,
        orphanedSizeBytes: 0
      },
      {
        shardName: 'rs-shard-03',
        numOrphanedDocs: 0,
        numOwnedDocuments: 266,
        ownedSizeBytes: 52668,
        orphanedSizeBytes: 0
      }
    ]
  },
  {
    ns: 'RampaBase.car_prices',
    shards: [
      {
        shardName: 'rs-shard-01',
        numOrphanedDocs: 0,
        numOwnedDocuments: 2005,
        ownedSizeBytes: 228570,
        orphanedSizeBytes: 0
      },
```

```
      {
        shardName: 'rs-shard-02',
        numOrphanedDocs: 0,
        numOwnedDocuments: 2232,
        ownedSizeBytes: 254448,
        orphanedSizeBytes: 0
      },
      {
        shardName: 'rs-shard-03',
        numOrphanedDocs: 0,
        numOwnedDocuments: 2096,
        ownedSizeBytes: 238944,
        orphanedSizeBytes: 0
      }
    ]
  },
  {
    ns: 'RampaBase.cars',
    shards: [
      {
        shardName: 'rs-shard-01',
        numOrphanedDocs: 0,
        numOwnedDocuments: 89430,
        ownedSizeBytes: 43373550,
        orphanedSizeBytes: 0
      },
      {
        shardName: 'rs-shard-02',
        numOrphanedDocs: 0,
        numOwnedDocuments: 89401,
        ownedSizeBytes: 43359485,
        orphanedSizeBytes: 0
      },
      {
        shardName: 'rs-shard-03',
        numOrphanedDocs: 0,
        numOwnedDocuments: 89424,
        ownedSizeBytes: 43370640,
        orphanedSizeBytes: 0
      }
    ]
  }
]
```

7.5.3 Stav balanceru

```
sh.getBalancerState()
```

Výklad:
Vrací true nebo false, zda je aktivní balancer, který automaticky přesouvá chunky mezi shardy pro dosažení rovnováhy.

Výstup:

```
true
```

7.5.4 Informace o běžících routerech

```
db.adminCommand({ getCmdLineOpts: 1 })
```

Výklad:
Získá informace o konfiguraci běžícího mongos, včetně parametrů spuštění, konfiguračního souboru, verzí a cest.

Výstup:

```
{
  argv: [
    'mongos',
    '--port',
    '27017',
    '--configdb',
    'rs-config-server/configsvr01:27017,configsvr02:27017,configsvr03:27017',
    '--bind_ip_all',
    '--keyFile',
    '/data/mongodb-keyfile'
  ],
  parsed: {
    net: { bindIp: '*', port: 27017 },
    security: { keyFile: '/data/mongodb-keyfile' },
    sharding: {
      configDB: 'rs-config-
server/configsvr01:27017,configsvr02:27017,configsvr03:27017'
    }
  },
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1747768519, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('UWaBU4DbvuU8nM/PKjZ6f/buWjM=', 0),
      keyId: Long('7506503454924210199')
    }
  },
  operationTime: Timestamp({ t: 1747768519, i: 1 })
}
```

7.5.5 Ověření chunků a jejich rozložení

```
db.chunks.aggregate([
  { $group: { _id: "$shard", count: { $sum: 1 } } }
])
```

Výklad:

Pomocí agregace zjišťuje, kolik chunků připadá na každý shard. Užitečné pro kontrolu rovnováhy v clusteru. Spuštěno na config databázi.

Výstup:

```
[
  { _id: 'rs-shard-02', count: 5 },
  { _id: 'rs-shard-01', count: 6 },
  { _id: 'rs-shard-03', count: 5 }
]
```

7.5.6 Stav všech mongos routerů v clusteru

```
db.adminCommand({ listShards: 1 })
```

Výklad:
Vrací seznam všech shardů.

Výstup:

```
{
  shards: [
    {
      _id: 'rs-shard-01',
      host: 'rs-shard-01/shard01-a:27017,shard01-b:27017,shard01-c:27017',
      state: 1,
      topologyTime: Timestamp({ t: 1747744036, i: 10 }),
      replSetConfigVersion: Long('1')
    },
    {
      _id: 'rs-shard-02',
      host: 'rs-shard-02/shard02-a:27017,shard02-b:27017,shard02-c:27017',
      state: 1,
      topologyTime: Timestamp({ t: 1747744036, i: 28 }),
      replSetConfigVersion: Long('1')
    },
    {
      _id: 'rs-shard-03',
      host: 'rs-shard-03/shard03-a:27017,shard03-b:27017,shard03-c:27017',
      state: 1,
      topologyTime: Timestamp({ t: 1747744036, i: 51 }),
      replSetConfigVersion: Long('1')
    }
  ],
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1747769017, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('p1pRHfZh0kboEUTcHvVzx5792cw=', 0),
      keyId: Long('7506503454924210199')
    }
  },
  operationTime: Timestamp({ t: 1747769017, i: 1 })
}
```

Závěr

Zpracování tohoto projektu bylo rozsáhlé a technicky náročné. Vyžadovalo propojení několika technologií – od práce s dockerem, přes konfiguraci distribuovaného MongoDB clusteru se shardingem a replikací, až po analýzu a import dat pomocí Pythonu. Počáteční fázi komplikovaly problémy se správným nastavením shardingu a spuštěním služeb v souvislosti s automatizací inicializačního procesu.

Práce se zaměřila na návrh, realizaci a zprovoznění shardovaného MongoDB clusteru s replikací a zabezpečenou autentizací. Implementace probíhala pomocí docker compose a shell skriptů, které nastavují replikace, sharding, validaci a import reálných dat. Výsledný cluster je plně funkční, opakovaně spustitelný a přenositelný mezi Linuxem a Windows (s WSL).

Za přínos považuji hlubší pochopení principů NoSQL databází, praktickou práci se shardingem a schopnost řešit vzniklé problémy napříč platformami.

Jistým omezením je, že prostředí bylo testováno pouze lokálně. Pro budoucí vývoj by bylo vhodné zaměřit se na rozšíření o nástroje pro monitoring, zálohování nebo nasazení do cloudové infrastruktury.

Zdroje

BSON Types—MongoDB Manual. MongoDB, [cit. 2025-05-19]. Dostupné z: https://www.mongodb.com/docs/manual/reference/bson-types/

Built-In Roles—MongoDB Manual. MongoDB, [cit. 2025-05-19]. Dostupné z: https://www.mongodb.com/docs/manual/reference/built-in-roles/

CData. MongoDB Use Cases. [2024-08-26]. Dostupné z: https://www.cdata.com/blog/mongodb-use-cases

Choose a Shard Key—MongoDB Manual. MongoDB, [cit. 2025-05-19]. Dostupné z: https://www.mongodb.com/docs/manual/core/sharding-choose-a-shard-key/

Documents—MongoDB Manual. MongoDB, [cit. 2025-05-19]. Dostupné z: https://www.mongodb.com/docs/manual/core/document/

Edureka. Real World Use Cases of MongoDB. [2024-03-20]. Dostupné z: https://www.edureka.co/blog/real-world-use-cases-of-mongodb/

FAQ: MongoDB Storage — MongoDB Manual. MongoDB, [cit. 2025-05-19]. Dostupné z: https://www.mongodb.com/docs/manual/faq/storage/

I/O Digital TechHub. Redis vs MongoDB vs Cassandra. [2022-08-12]. Dostupné z: https://techhub.iodigital.com/articles/redis_vs_mongodb_vs_cassandra

Introduction to MongoDB — MongoDB Manual. MongoDB, [cit. 2025-05-19]. Dostupné z: https://www.mongodb.com/docs/manual/introduction/

Khandelwal, Shubham. Use Cases or Case Study of MongoDB. Medium [2021-05-12]. Dostupné z: https://shubhamkhandelwal523.medium.com/use-cases-or-case-study-of-mongodb-3413c330f6be

MINHHUNGIT. mongodb-cluster-docker-compose. GitHub, 2024 [cit. 2025-05-19]. Dostupné z: https://github.com/minhhungit/mongodb-cluster-docker-compose

MongoDB Documentation. MongoDB, [cit. 2025-05-19]. Dostupné z: https://www.mongodb.com/docs/

MongoDB Inc. Aggregation Framework Introduction. [2022-02-01]. Dostupné z: https://www.mongodb.com/developer/products/mongodb/introduction-aggregation-framework/

MongoDB Inc. Customer Case Study – MetLife. [2013-11-22]. Dostupné z: https://www.dataversity.net/metlifes-wall-powered-by-mongodb/

MongoDB Inc. Customer Case Study – Shutterfly. [cit. 2025-05-19]. Dostupné z: https://www.mongodb.com/solutions/customer-case-studies/shutterfly

MongoDB Inc. Customer Case Study – Toyota Connected. [cit. 2025-05-19]. Dostupné z: https://www.mongodb.com/solutions/customer-case-studies/toyota-connected

MongoDB Inc. Sharding Explained. [cit 2025-05-19]. Dostupné z: https://www.mongodb.com/resources/products/capabilities/database-sharding-explained

MongoDB Limits and Thresholds—MongoDB Manual. MongoDB, [cit. 2025-05-19]. Dostupné z: https://www.mongodb.com/docs/manual/reference/limits/

Replica Set Members—MongoDB Manual. MongoDB, [cit. 2025-05-19]. Dostupné z: https://www.mongodb.com/docs/manual/core/replica-set-members/

Replication—MongoDB Manual. MongoDB, [cit. 2025-05-19]. Dostupné z: https://www.mongodb.com/docs/manual/replication/

Role-Based Access Control—MongoDB Manual. MongoDB, [cit. 2025-05-19]. Dostupné z: https://www.mongodb.com/docs/manual/core/authorization/

Schema Validation—MongoDB Manual. MongoDB, [cit. 2025-05-19]. Dostupné z: https://www.mongodb.com/docs/manual/core/schema-validation/

Sharding—MongoDB Manual. MongoDB, [cit. 2025-05-19]. Dostupné z: https://www.mongodb.com/docs/manual/sharding/

Použité nástroje:

DIAGRAMS.NET. draw.io [online]. [cit. 2025-05-20]. Dostupné z: https://www.draw.io/

Docker Inc. Docker [software]. [cit. 2025-05-20]. Dostupné z: https://www.docker.com/

Docker Inc. (2). Docker Compose v2.34.0-desktop.1 [software]. [cit. 2025-05-20]. Dostupné z: https://docs.docker.com/compose/

Git SCM. Git [software]. [cit. 2025-05-20]. Dostupné z: https://git-scm.com/

GitHub Inc. Git Large File Storage (Git LFS) [software]. [cit. 2025-05-20]. Dostupné z: https://git-lfs.github.com/

Microsoft. Microsoft Word [software]. [cit. 2025-05-20]. Dostupné z: https://www.microsoft.com/cs-cz/microsoft-365/word

Microsoft. Visual Studio Code [software]. [cit. 2025-05-20]. Dostupné z: https://code.visualstudio.com/

Microsoft. MongoDB for VS Code Extension [software]. [cit. 2025-05-20]. Dostupné z: https://marketplace.visualstudio.com/items?itemName=mongodb.mongodb-vscode

MongoDB Inc. MongoDB Compass [software]. [cit. 2025-05-20]. Dostupné z: https://www.mongodb.com/products/tools/compass

Notepad++ Team. Notepad++ [software]. [cit. 2025-05-20]. Dostupné z: https://notepad-plus-plus.org/

The Document Foundation. LibreOffice Writer [software]. [cit. 2025-05-20]. Dostupné z: https://www.libreoffice.org/

Přílohy

Inicializace clusteru:

- docker-compose.yml
- manula-cluster-init.sh

Datasety:

- Ad_table (extra).csv
- Image_table.csv
- Price_table.csv
- Sales_table.csv
- Wolt.csv

Scripty:

- auth.js
- enable-sharding.js
- import_all.py
- import_cars.py
- import_images.py
- import_prices.py
- import_sales.py
- import_wolt.py
- init-configserver.js
- init-router.js
- init-shard01.js
- init-shard02.js
- init-shard03.js
- manual-cluster-init.sh
- myImportLib.py

Analýza:

- Ad_table (extra)
  - Ad_table (extra)_histogram_cen.png
  - Ad_table (extra)_statistika_heatmapa.png
  - Ad_table (extra)_top_znacky.png
  - Ad_table (extra)_vystup.txt
- Image_table
  - Image_table_top_models.png
  - Image_table_viewpoint_histogram_legend.png
  - Image_table_vystup.txt
  - Image_tablequality_check_barplot.png
- Price_table
  - Price_table_average_price_by_year.png
  - Price_table_histogram_entry_price.png
  - Price_table_histogram_price.png
  - Price_table_top_models_price.png
  - Price_table_vystup.txt
- Sales_table
  - Sales_table_heatmap_top_models.png
  - Sales_table_sales_by_year.png
  - Sales_table_top_models_total_sales.png
  - Sales_table_vystup.txt
- Wolt
  - Wolt_delay_histogram.png
  - Wolt_user_location_heatmap.html
  - Wolt_user_location_scatter.png
  - Wolt_vystup.txt
  - Wolt_weather_correlation.png

Ostatní:

- mongodbcluster.jpg
- mongodbcluster.drawio