# UPA 2. projekt

Ahoj! Rádi bychom vám představili náš projekt, ve kterém srovnáváme různé typy databází a popisujeme jejich vhodnost pro různé účely. Zde je kompletní seznam databází, které jsme zahrnuli:

- sloupcová wide-column databáze (Apache Cassandra)
- dokumentová databáze (MongoDB)
- grafová databáze (Neo4J)
- databáze časových řad (InfluxDB)

## Apache Cassandra (Sloupcová databáze) – Doprava

### Název sady

Statistika průjezdu vozidel ze sledovaných křižovatek v roce 2024 odkaz

### Vhodná distribuce datové sady

CSV – obsahuje časové záznamy, křižovatky, data o intenzitě provozu.

### Zvolený druh databáze

Sloupcová databáze – Apache Cassandra

### Vysvětlení, proč je tento druh databáze nejlepší pro danou sadu

Apache Cassandra poskytuje možnost zpracovat velké objemy dat s nízkou latencí. Oblast dopravy zahrnuje velké objemy dat, generované v reálném čase. Cassandra je užitečná v případech, kdy je potřeba zapisovat a číst velká množství časově relevantních informací, což umožňuje rychlou analýzu a reakci v reálném čase. Díky replikaci dat, zajišťuje vysokou dostupnost a odolnost proti výpadkům.

- Data jsou dobře uspořádána do sloupců (např. Datum, Stanice, Počet), což je pro Cassandru přirozené a umožňuje efektivní dotazování na konkrétní sloupce.
- Škálovatelnost a distribuce: Cassandra je navržena jako distribuovaná a škálovatelná databáze, což zaručuje rychlé čtení a zápis dat i při velkém objemu a růstu a data lze replikovat pro zajištění vyšší dostupnosti.
- Rychlý zápis a retence: Je optimalizována pro rychlý zápis nových dat a snadnou správu retence díky funkci TTL, což je ideální pro pravidelné aktualizace záznamů.
- Optimalizace dotazů: Podpora sekundárních indexů a materiálizovaných pohledů umožňuje rychlé načítání dat podle specifických dotazů (např. podle stanice nebo časového rozmezí).

### Příkazy pro definici uložiště

```
CREATE KEYSPACE traffic WITH replication = {‘class’: ‘SimpleStrategy’,
‘replication_factor’ : 3};

CREATE TABLE traffic.traffic_data (
    stanice TEXT,
```

```
    datum TIMESTAMP,
    trida_objektu TEXT,
    pocet INT,
    PRIMARY KEY (stanice, datum)
) WITH CLUSTERING ORDER BY (datum DESC);
```

Tento příkaz vytvoří tabulku v databázi, která bude přijímat položky specifické pro daný dataset. Zároveň zajistí, že každý sloupec bude mít definovaný typ dat, což omezuje vkládání nesprávných nebo nekompatibilních dat do tabulky.

## Algoritmický popis importu dat

**Počáteční Naplnění** Databáze:

1. Připojit se k databázi.
2. Načíst data z CSV souboru.
3. Pro každý záznam:
    - Načíst hodnoty (datum, stanice, trida_objektu, pocet).
    - Vykonat `INSERT` dotaz.
4. Uzavřít připojení.

**Doplnění** Nových **či Změněných** Dat:

1. Připojit se k databázi.
2. Načíst data z CSV souboru.
3. Pro každý záznam:
    - Načíst hodnoty.
    - Vykonat `UPSERT` dotaz.
4. Uzavřít připojení.

**Příklad** skriptu

```
from cassandra.cluster import Cluster
from datetime import datetime
import csv

cluster = Cluster([API])
session = cluster.connect()

with open('data.csv', newline='', encoding='utf-8') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        stanice = row['Stanice']
        datum_str = row['Datum']
        trida_objektu = row['Třída objektu']
        pocet = int(row['Počet'])

        datum = datetime.strptime(datum_str, "%d.%m.%Y %H:%M")
```

```
        session.execute("""
        INSERT INTO traffic_data (stanice, datum, trida_objektu, pocet)
        VALUES (%s, %s, %s, %s)
        """, (stanice, datum, trida_objektu, pocet))

cluster.shutdown()
```

## Dotaz v jazyce databázového produktu

```
SELECT * FROM traffic.traffic_data
WHERE stanice = 'Novinářská x 28. října - od Fifejd'
ORDER BY datum DESC;
```

Tento dotaz načte všechna data pro stanice `Novinářská x 28. října - od Fifejd`, seřazená podle `datum` v sestupném pořadí.

## Popis zpracování dotazu

- Vyhledání **Uzlů** s Daty:
    - Cassandra používá konzistentní hašování k určení, které uzly obsahují data pro daný partiční klíč `stanice = 'Novinářská x 28. října - od Fifejd'`.
    - Databáze identifikuje uzly, kde jsou tato data uložena a to podle své **replikační** strategie (v tomto případě `SimpleStrategy` s replikačním faktorem 3), což zajišťuje dostupnost dat na více uzlech pro zvýšení spolehlivosti.
- **Načtení** Dat:
    - Jakmile jsou příslušné uzly identifikovány, Cassandra načte data ze specifikovaného partičního oddílu.
    - Uvnitř tohoto oddílu jsou data již seřazena podle `datum`, díky čemuž není nutné další řazení.
- Vrácení **Výsledků** Klientovi:
    - Cassandra spojí a připraví výsledky a doručí je klientovi ve vyžádaném pořadí (od nejnovějších záznamů).
    - Klientská aplikace (například program v Pythonu nebo jiná služba) poté přijme data připravená k dalšímu zpracování nebo zobrazení.

# MongoDB (Dokumentová databáze) – Stížnosti a petice

## Název sady

Stížnosti – počet podání v roce 2024 odkaz

## Vhodná distribuce datové sady

JSON – obsahuje údaje o roce podání, měsíci podání a počtu podání

## Zvolený druh databáze

Dokumentová databáze – MongoDB

## Vysvětlení, proč je tento druh databáze nejlepší pro danou sadu

1. Flexibilita schématu: MongoDB je schema-less, což znamená, že struktura dokumentů může být snadno měněna a přizpůsobena různým typům dat bez potřeby migrace databáze.
2. Snadné vkládání a aktualizace dat: Pomocí operace `upsert` lze jednoduše vkládat nové a aktualizovat existující záznamy, což zajišťuje efektivní správu dat i při častých změnách.
3. Vysoký výkon: MongoDB je optimalizováno pro rychlé čtení a zápisy, což je důležité pro aplikace s vysokým objemem dat a potřebou rychlého přístupu.
4. Škálovatelnost: MongoDB snadno škáluje horizontálně přidáním více uzlů do sharded clusteru, což zajišťuje, že databáze může růst společně s objemem dat a potřebami aplikace.

## Příkazy pro definici uložiště

```
db.createCollection("LogEntry", {
    validator: {
        $jsonSchema: {
            bsonType: "object",
            required: ["Rok", "Měsíc_název", "Měsíc_číslo", "Počet"],
            properties: {
                Rok: {
                    bsonType: "int",
                    description: "musí být celé číslo a je vyžadováno"
                },
                Měsíc_název: {
                    bsonType: "string",
                    description: "musí být řetězec a je vyžadováno"
                },
                Měsíc_číslo: {
                    bsonType: "int",
                    description: "musí být celé číslo a je vyžadováno"
                },
                Počet: {
                    bsonType: "int",
                    description: "musí být celé číslo a je vyžadováno"
                }
            }
        }
    },
    validationLevel: "strict"
})
```

Tato metoda založí strukturu databáze.

## Algoritmický popis importu dat

Python Skript pro Import Dat

```
from pymongo import MongoClient
import csv
```

```
def import_data(file_path):
    client = MongoClient('mongodb://api/')
    db = client['moje_databaze']
    collection = db['LogEntry']

    with open(file_path, 'r') as f:
        reader = csv.DictReader(f)
        for row in reader:
            query = {"Rok": row['Rok podání'], "Měsíc_číslo": row['Měsíc podání -
číslo']}
            update = {
                "$set": {
                    "Měsíc_název": row['Měsíc podání - název'],
                    "Počet": int(row['Počet podání'])
                }
            }
            collection.update_one(query, update, upsert=True)

    client.close()
```

Tento skript:

- Připojí se k MongoDB databázi.
- Načte data z CSV souboru.
- Provádí `updateOne` operaci s `upsert=True`, což zajišťuje, že pokud záznam neexistuje, bude vytvořen, a pokud existuje, bude aktualizován.
- Uzavře připojení k databázi.

## Dotaz v jazyce databázového produktu

Příklad dotazu, který získá všechny záznamy pro konkrétní rok a určité časové období:

```
db.LogEntry.find({
    "Rok": 2024,
    "Měsíc_číslo": { "$gte": 1, "$lte": 6 }
})
```

- Vyhledání **dokumentů**:
    - MongoDB používá indexy pro rychlé vyhledávání dokumentů. Pokud dotaz zahrnuje pole `Rok` a `Měsíc_číslo`, je výhodné, aby byla tato pole indexována.
    - Pomocí indexování a metadat MongoDB najde servery (uzly), na kterých jsou uložena požadovaná data.
- Získání dat:
    - Dotazový engine MongoDB prohledá příslušné kolekce a najde všechny dokumenty, které odpovídají kritériím dotazu, například pro hodnoty `Rok` = 2024 a `Měsíc_číslo` v rozmezí 1–6.
- **Doručení výsledků** klientovi:
    - Koordinátor vrátí zpracované výsledky zpět klientovi.

- Výsledek je klientovi doručen ve formátu JSON, což usnadňuje další zpracování a použití výstupu v aplikacích.

## Grafové databáze – Neo4J

- využívají uzly k reprezentaci entit, hrany k zachycení vztahů mezi nimi a vlastnosti k přidání atributů, což umožňuje přímo modelovat propojená data
- jsou vhodné pro data s větším množstvím vzájemných vztahů, kde je nutné efektivně manipulovat s komplexními propojeními mezi jednotlivými entitami
- jsou navrženy tak, aby efektivně optimalizovaly dotazy zaměřené na vyhledávání vztahů a vzorců, přičemž zvládají složité operace, které by v relační databázi probíhaly pomalu
- poskytují snadné přizpůsobení pro případné změny ve struktuře dat, což je výhodné při práci se síťovými daty
- jsou ideální pro aplikace, které potřebují efektivně pracovat s komplexními vazbami mezi daty, jako jsou sociální sítě, doporučovací systémy, detekce podvodů či znalostní grafy

## 1) plný název zvolené datové sady vč. odkazu URL ve výše uvedeném katalogu

- Dopravní přestupky dle data a místa spáchání (sekce Doprava)
- https://opendata.ostrava.cz/dopravni-prestupky-dle-data-a-mista-spachani-v-roce-2024/

## 2) vhodná distribuce dané datové sady (např. CSV či GeoJSON) dle nabídky v katalogu

CSV je vhodný zejména pro tabulková data. Tento formát je ideální pro import do Neo4J, protože snadno transformuje tabulková data na grafové struktury. CSV se také hojně používá při analýze a zpracování dat v různých skriptech a nástrojích pro data mining, které umožňují snadný přístup k datům a jejich převod do grafových modelů.

## 3) zvolený druh NoSQL databáze a konkrétní databázový produkt/server

- grafová databáze – Neo4J

## 4) podrobné a logické vysvětlení, proč je nejlepší k uložení a dotazování zvolené datové sady použít právě daný druh NoSQL databáze

Tato datová sada obsahuje vzájemně propojené prvky jako přestupky, zákony, paragrafy, místa činu a způsoby vyřízení. Grafová databáze je ideální pro modelování vztahů mezi těmito entitami (například propojení přestupku, paragrafu a místa činu), protože umožňuje efektivní analýzu a dotazování na propojená data.

**Důvody** pro volbu Neo4J:

1) Dataset obsahuje složité propojení mezi časem, místem a událostmi. Neo4J vyniká v zachycení těchto propojení mezi entitami, což není snadné dosáhnout v databázích jako Apache Cassandra, MongoDB nebo InfluxDB. S grafovou databází Neo4J lze snadno prozkoumávat a vizualizovat vztahy mezi uzly, což usnadňuje analýzu propojených dat.
2) Neo4J nabízí silnou podporu pro dynamické přidávání nových vztahů mezi entitami (například nový přestupek spáchaný v určitém místě a čase). Zatímco Apache Cassandra a MongoDB

umožňují zápis dat, jejich možnosti pro správu složitých vztahů a vícestupňových dotazů jsou omezené. InfluxDB se zaměřuje na časové řady, ale není optimalizováno pro dotazování na vztahy mezi různými entitami.

3) Dotazy v grafové databázi Neo4J (Cypher) jsou navrženy tak, aby efektivně vyhledávaly vzorce a vztahy v datech. Například se můžete ptát: „Jaké přestupky se staly v daném městském obvodu v konkrétním měsíci?“. V Apache Cassandře, MongoDB nebo InfluxDB by to bylo mnohem složitější:

   - Cassandra by vyžadovala složité návrhy schématu
   - MongoDB by byl nucen používat složité agregace nebo „joins“
   - InfluxDB by se musel uchýlit k složitým agregacím nebo rozdělování dat do různých časových řad

## 5) syntakticky i sémanticky korektní příkazy pro definici úložiště v daném produktu/serveru NoSQL databáze pro danou datovou sadu

4) Definice **uzlů** a hran v Cypher

```
// Vytvoření uzlu pro přestupek
CREATE CONSTRAINT p FOR (p:Prestupek) REQUIRE p.id IS UNIQUE;
CREATE (p:Prestupek {
	id: '0857',
	datum_spachani: '2023-11-12',
	misto_cinu: 'v Havířově na ul. Boženy Němcové',
	zp_spraveni: 'pokuta'
})

// Vytvoření uzlu pro zákon
CREATE CONSTRAINT FOR (z:Zakon) REQUIRE z.cislo_zakona IS UNIQUE;
CREATE (z:Zakon {
	cislo_zakona: '168/1999',
	paragraf: '16',
	odstavec: 1,
	pismeno: ''
})

// Vytvoření uzlu pro místo činu
CREATE CONSTRAINT ON (m:Misto) REQUIRE m.nazev IS UNIQUE;
CREATE (m:Misto {
	nazev: 'v Havířově na ul. Boženy Němcové'
	})

// Vytvoření vztahů mezi uzly
MATCH (p:Prestupek {id: '0857'}), (z:Zakon {cislo_zakona: '168/1999'}), (m:Misto
{nazev: 'v Havířově na ul. Boženy Němcové'})
CREATE (p)-[:PORUSEN_ZAKON]->(z), (p)-[:STALO_SE_V]->(m)
```

2) Vložení více dat najednou pomocí Cypher

```
LOAD CSV WITH HEADERS FROM 'file:///dopravni_prestupky.csv' AS row
CREATE (p:Prestupek {
	id: row.id,
```

```
      datum_spachani: row.datum_spachani,
      misto_cinu: row.misto_cinu,
      zp_spraveni: row.zp_spraveni
      })

CREATE (z:Zakon {
      cislo_zakona: row.cislo_zakona,
      paragraf: row.paragraf
      })

CREATE (m:Misto {
      nazev: row.misto_cinu
      })
```

- v případě, že chceme předejít vkládání duplicitních uzlů, je vhodnější využít `MERGE`:

```
LOAD CSV WITH HEADERS FROM 'file:///dopravni_prestupky.csv' AS row
MERGE (p:Prestupek {id: row.id})
SET p.datum_spachani = row.datum_spachani,
      p.misto_cinu = row.misto_cinu,
      p.zp_spraveni = row.zp_spraveni

MERGE (z:Zakon {
      cislo_zakona: row.cislo_zakona,
      paragraf: row.paragraf
      })

MERGE (m:Misto {
      nazev: row.misto_cinu
})
```

- vztahy se definují úplně stejně jako v předešlém případu:

```
CREATE (p)-[:PORUSEN_ZAKON]->(z),(p)-[:STALO_SE_V]->(m)
```

## 6) algoritmický popis importu dat ze zvolené distribuce dané datové sady do připravené databáze a to jak počáteční naplnění prázdné databáze daty, tak pozdější doplnění nových či změněných dat

Pro import dat z formátu CSV do Neo4J pomocí operace `UPSERT` je důležité zvolit unikátní identifikátor, například ID přestupku. Tento klíč pomáhá určit, zda záznam již existuje a na základě toho se rozhodnout, zda vložit nový záznam, nebo aktualizovat ten stávající.

1) Import **počátečních** dat

```
Funkce import_dat(dataset):
    Pro každý záznam v dataset:
        Pokud existuje uzel Prestupek s ID záznamu:
            Aktualizuj uzel Prestupek s novými daty:
                - datum_spachani: datum_spachani záznamu
                - misto_cinu: misto_cinu záznamu
                - zp_spraveni: zp_spraveni záznamu
            Pokud existuje vztah k zákonu a místu:
                Aktualizuj vztah k zákonu a místu
        Jinak:
            Vytvoř nový uzel Prestupek:
```

```
                - id: ID přestupku
                - datum_spachani: datum_spachani
                - misto_cinu: misto_cinu
                - zp_spraveni: zp_spraveni
            Vytvoř vztah k zákonu a místu
        Konec pokud
    Konec pro
```

2) Aktualizace dat

```
Funkce aktualizuj_data(dataset):
    Pro každý nový nebo změněný záznam v dataset:
        Pokud existuje uzel Prestupek s ID záznamu:
            Aktualizuj informace o přestupku a vztahy
        Jinak:
            Vlož nový uzel a vztahy
        Konec pokud
    Konec pro
```

Cypher v Neo4J neobsahuje explicitní příkaz `UPSERT`, avšak tuto funkcionalitu lze napodobit kombinací příkazů `MATCH` a `MERGE`. Nejprve se pomocí `MATCH` hledají existující uzly a pokud se najdou, použije se `MERGE` k jejich aktualizaci, nebo v případě, že uzel neexistuje, vytvoří se nový.

## 7) alespoň jeden syntakticky i sémanticky korektní dotaz v jazyce daného databázového produktu nad databází s uloženými daty dané datové sady, který bude demonstrovat vhodnost zvoleného druhu NoSQL databáze pro daná data vč. popisu způsobu, jakým databázový server dotaz zodpoví

```
MATCH (p:Prestupek)-[:PORUSUJE]->(z:Zakon)
WHERE p.misto_cinu = 'v Havířově na ul. Boženy Němcové'
RETURN p.id AS PrestupekID, p.datum_spachani AS Datum, z.nazev AS Zakon
```

1) Vyhledání **uzlů** a **vztahů:**
    - `MATCH` vyhledává uzly typu `Prestupek` a vztahy typu `PORUSUJE` mezi uzly `Prestupek` a `Zakon`
    - `WHERE` filtruje přestupky podle místa činu, konkrétně na základě hodnoty `'v Havířově na ul. Boženy Němcové'`
2) Získání dat:
    - dotaz získá ID přestupku a datum jeho spáchání z uzlu typu `Prestupek` a zároveň název zákona z uzlu typu `Zakon`
3) **Doručení** a zpracování klientem:
    - po zpracování databázovým serverem se výsledky vrátí zpět klientovi, například webové aplikaci nebo API

## Časové řady (InfluxDB)

- jsou optimalizovány pro data, která obsahují časové značky, přičemž každý záznam je spojen s konkrétním časem nebo časovým obdobím

- jsou navrženy pro ukládání velkých objemů časově uspořádaných dat, s efektivní kompresí a indexováním
- jsou ideální pro aplikace, které pracují s velkým množstvím zápisů a potřebují časté aktualizace dat
- podporuje dotazy zaměřené na čas, jako agregace, vzorkování a pravidla pro uchovávání dat
- jsou vhodné pro monitorování serverů, sběr dat z IoT zařízení, analýzu senzorových dat nebo sledování finančního obchodování

## 1) plný název zvolené datové sady vč. odkazu URL ve výše uvedeném katalogu

- Počet evidovaných psů – časový vývoj (sekce Životní prostředí)
- https://opendata.ostrava.cz/pocet-evidovanych-psu/

## 2) vhodná distribuce dané datové sady (např. CSV či GeoJSON) dle nabídky v katalogu

Distribuce dat k dispozici je ve formátu CSV, který je ideální pro strukturovaná data uspořádaná v tabulkách. Tento formát lze snadno importovat do různých databázových systémů, včetně InfluxDB, kde je možné při importu jednoduše mapovat sloupce, jako je počet psů, městský obvod, rok a měsíc, na tagy a hodnoty. Tento přístup se hodí pro analýzu časových trendů, například pro sledování počtu psů v jednotlivých měsících nebo letech.

## 3) zvolený druh NoSQL databáze a konkrétní databázový produkt/server

- databáze časových řad – InfluxDB

## 4) podrobné a logické vysvětlení, proč je nejlepší k uložení a dotazování zvolené datové sady použít právě daný druh NoSQL databáze

Tato datová sada obsahuje časově označené údaje (rok, měsíc, počet psů), což je ideální pro analýzu časových řad. InfluxDB je navržena tak, aby efektivně zpracovávala tato data, to umožňuje rychlou agregaci, filtraci a analýzu trendů v čase. Tento formát je užitečný pro identifikaci vzorců, jako je sledování vývoje počtu psů v průběhu měsíců nebo let a nabízí efektivní zpracování dotazů zaměřených na časové změny.

**Důvody** pro volbu InfluxDB:

1) Dataset obsahuje časově označené údaje (rok, měsíc, počet psů), což je ideální pro práci s časovými řadami. Apache Cassandra je vhodná pro zpracování velkých objemů dat, avšak není optimalizována pro složité časové operace a agregace. MongoDB sice podporuje časové data, ale jeho výkon při analýze časových řad není tak efektivní jako u InfluxDB. Neo4J je určen pro grafová data a vztahy, nikoli pro časově orientované dotazy.
2) Rychlé zápisy a dotazy v reálném čase jsou esenciální pro analýzu trendů. Apache Cassandra je ideální pro rychlé zápisy, ale složité časové dotazy a agregace mohou být náročné na výkon, zejména u historických dat. Přestože MongoDB zvládá zápisy dobře, není optimalizován pro časové operace. Neo4J se zaměřuje na grafy a vztahy, časová orientace dat ho vůbec nezajímá.

3) InfluxDB podporuje složité dotazy s časovými operacemi, což výrazně zjednodušuje analýzu trendů v čase. Například dotaz na vývoj počtu psů v různých městských obvodech je v InfluxDB velmi snadný.
    - Cassandra by vyžadovala složitější návrh schématu, což by ztížilo správu a optimalizaci časových dotazů
    - MongoDB by muselo použít složité agregace, jako jsou `$group`, `$sort` a `$project`, což zvyšuje složitost dotazů a může zhoršit výkon při práci s časově orientovanými daty
    - Neo4J by mohla potřebovat složitější konstrukci grafů, což by mohlo být méně efektivní než v databázích, které jsou přímo navržené pro časová data

## 5) syntakticky i sémanticky korektní příkazy pro definici úložiště v daném produktu/serveru NoSQL databáze pro danou datovou sadu

- následující příkazy lze použít pouze tehdy, pokud máte InfluxDB nainstalovanou lokálně, nebo pokud používáte InfluxDB Cloud či jiný hostovaný server:

1) **Vytvoření** databáze:

```
# Vytvoření databáze 'psovod'
CREATE DATABASE psovod
```

- případně i tento příkaz, pokud chceme pracovat s touto specifickou databází:

```
USE psovod
```

2) Vložení dat:

- pro zápis dat o počtu psů do databáze použijeme tagy pro městský obvod a měsíc a pole pro evidenci počtu psů:

```
# Vložení dat pro městský obvod "Slezská Ostrava" v prosinci 2019
INSERT pocet_psu, obvod="Slezská Ostrava", rok=2019, mesic_cislo=12, mesic="Prosinec"
pocet=1888
```

3) Dotaz na data:

- k získání počtu psů pro konkrétní městský obvod a měsíc lze použít tento dotaz:

```
SELECT "pocet" FROM "pocet_psu" WHERE "obvod"='Slezská Ostrava' AND "rok"=2019 AND
"mesic"='Prosinec'
```

4) **Načtení** dat z CSV:

- pro zápis CSV dat do InfluxDB lze využít následující způsoby:
    - nahrát soubor nebo zadat data ručně přes uživatelské rozhraní
    - použít příkaz `influx write` v příkazovém řádku
    - využít Telegraf s podporou CSV pluginu
    - použít Flux pro pokročilé úpravy a manipulaci s daty
- v tomto případě se zaměříme na použití příkazu `influx write`:

```
influx write -b bucket_name \
  -f path/to/file.csv \
  --header "#constant measurement,pocet_psu" \
  --header "#datatype long,tag,dateTime:2006,tag,long,string"
```

- `#constant measurement,pocet_psu` znamená, že všechna importovaná data budou přiřazena k měření s názvem `pocet_psu`
- `#datatype long,tag,dateTime:2006,tag,long,string` specifikuje datové typy jednotlivých sloupců:
    - První sloupec (`Počet psů s čipem`) je typu `long` (číslo)
    - Druhý sloupec (`Nazev`) je `tag` (název obvodu)
    - Třetí sloupec (`Rok`) je časový údaj ve formátu `2006`
    - Čtvrtý sloupec (`Mesic cislo v roce`) je `long` (číslo)
    - Pátý sloupec (`Měsíc`) je `string` (název měsíce)

4a) Dotazování na data:

- vykonávání dotazů nad daty podle specifických kritérií, například podle městského obvodu nebo roku:

```
SELECT * FROM "pocet_psu" WHERE "obvod" = 'Slezská Ostrava' AND "rok" = 2019
```

4b) Agregace dat:

- pro zjištění celkového počtu psů v Slezské Ostravě za prosinec 2019:

```
SELECT SUM("pocet") FROM "pocet_psu" WHERE "obvod" = 'Slezská Ostrava' AND
"rok" = 2019 AND "mesic_cislo_v_roce" = 12
```

- pro výpočet průměrného počtu psů za prosinec 2019 použijeme následující příkaz:

```
SELECT MEAN("pocet") FROM "pocet_psu" WHERE "obvod" = 'Slezská Ostrava' AND
"rok" = 2019 AND "mesic_cislo_v_roce" = 12
```

4c) Dotaz s **časovým** intervalem:

- pro získání dat v určitém časovém období (například od ledna do prosince 2019), použijeme:

```
SELECT * FROM "pocet_psu" WHERE "obvod" = 'Slezská Ostrava' AND "rok" = 2019
AND time >= '2019-01-01T00:00:00Z' AND time <= '2019-12-31T23:59:59Z'
```

4d) **Pokročilé** dotazy:

- počet psů pro každý obvod v prosinci 2019 lze spočítat následujícím způsobem:

```
SELECT "obvod", SUM("pocet") FROM "pocet_psu" WHERE "rok" = 2019 AND
"mesic_cislo_v_roce" = 12 GROUP BY "obvod"
```

6) algoritmický popis importu dat ze zvolené distribuce dané datové sady do připravené databáze a to jak počáteční naplnění prázdné databáze daty, tak pozdější doplnění nových či změněných dat

1) Import **počátečních** dat a jejich aktualizace:

```
// Funkce pro připojení k databázi InfluxDB
Funkce connect_to_influx:
	- Vytvoř klienta pro připojení k databázi InfluxDB s parametry:
		host: 'localhost'
		port: 8086
		databáze: 'psovod'
	- Vrať klienta

// Funkce pro kontrolu existence záznamu
Funkce exists_in_database:
	Parametry:
		client: klient InfluxDB
		obvod: název obvodu
		rok: rok
		mesic: název měsíce
	Kroky:
		- Vytvoř dotaz SELECT pro hledání záznamu s podmínkami:
			obvod: '{obvod}'
			rok: '{rok}'
			mesic: '{mesic}'
		- Spusť dotaz na databázi pomocí klienta
		- Pokud existují výsledky:
			- Vrať True
		- Jinak:
			- Vrať False

// Funkce pro vložení nového záznamu
Funkce insert_into_database:
	Parametry:
		client: klient InfluxDB
		obvod: název obvodu
		rok: rok
		mesic: název měsíce
		mesic_cislo: číslo měsíce v roce
		pocet_psu: počet psů
	Kroky:
		- Vytvoř JSON objekt pro nový záznam s hodnotami:
			měření: 'pocet_psu'
			tagy:
				obvod: '{obvod}'
				rok: '{rok}'
				mesic: '{mesic}'
			pole:
				mesic_cislo: '{mesic_cislo}'
				pocet_psu: '{pocet_psu}'
		- Použij klienta pro zápis dat do databáze

// Funkce pro aktualizaci existujícího záznamu
Funkce update_in_database:
```

```
      Parametry:
            client: klient InfluxDB
            obvod: název obvodu
            rok: rok
            mesic: název měsíce
            mesic_cislo: číslo měsíce v roce
            pocet_psu: počet psů
      Kroky:
            - Vytvoř dotaz UPDATE pro aktualizaci záznamu s podmínkami:
                  obvod: '{obvod}'
                  rok: '{rok}'
                  mesic: '{mesic}'
                  nastav:
                        pocet_psu: '{pocet_psu}'
                        mesic_cislo: '{mesic_cislo}'
            - Spusť dotaz na databázi pomocí klienta

// Funkce pro import dat
Funkce import_data:
      Parametry:
            client: klient InfluxDB
            dataset: seznam záznamů
      Kroky:
            - Pro každý záznam v datasetu:
                  - Získej hodnoty obvod, rok, mesic, mesic_cislo, pocet_psu
                  - Zavolej exists_in_database pro kontrolu existence záznamu
                  - Pokud záznam existuje:
                        - Zavolej update_in_database pro aktualizaci záznamu s
novými hodnotami
                  - Jinak:
                        - Zavolej insert_into_database pro vložení nového záznamu
```

- Unikátní **klíč**: Kombinace tagů `obvod`, `rok` a `mesic` slouží k jednoznačné identifikaci každého záznamu. Tento přístup umožňuje efektivní vyhledávání a zajišťuje, že každý záznam bude přiřazen ke správnému místu a časovému rámci.
- `UPSERT` operace: Při importu dat se uplatňuje logika `UPSERT`, což znamená, že pokud již existuje záznam na základě unikátního klíče, dojde k jeho aktualizaci. Pokud záznam chybí, vytvoří se nový. Tento proces zajišťuje, že data jsou vždy aktuální a nevedou k duplikacím.

## 7) alespoň jeden syntakticky i sémanticky korektní dotaz v jazyce daného databázového produktu nad databází s uloženými daty dané datové sady, který bude demonstrovat vhodnost zvoleného druhu NoSQL databáze pro daná data vč. popisu způsobu, jakým databázový server dotaz zodpoví

```
SELECT mean("pocet_psu")
FROM "pocet_psu"
WHERE "obvod" = 'Slezská Ostrava'
 AND time >= '2019-12-01T00:00:00Z'
 AND time < '2020-01-01T00:00:00Z'
GROUP BY time(1mo)
```

1) Vyhledání odpovídajících **časových řad**:

- InfluxDB vyhledá všechny záznamy v databázi, které splňují podmínku `WHERE "obvod" = 'Slezská Ostrava'`. Tento filtr se zaměřuje výhradně na záznamy týkající se obvodu Slezská Ostrava a zároveň vybírá pouze ty, které mají časovou značku v prosinci 2019.

2) Filtrace podle **časového** období:

- Parametr `WHERE time >= '2019-12-01T00:00:00Z' AND time < '2020-01-01T00:00:00Z'` určuje časový rámec dotazu. Tento rozsah zahrnuje celý měsíc prosinec 2019, počínaje 1. prosincem 2019 a konče půlnocí 1. ledna 2020. Tím se zajišťuje, že dotaz se zaměří pouze na data v tomto konkrétním měsíci.

3) Agregace po **měsíčních** intervalech:

- Použití `GROUP BY time(1mo)` znamená, že InfluxDB shromáždí data za celý měsíc a provede jejich agregaci. V tomto případě se seskupí všechny záznamy za prosinec 2019, přičemž bude spočítán průměrný počet psů (`mean("pocet_psu")`) pro tento měsíc. Tato operace umožňuje získat souhrnné údaje pro celý měsíc namísto individuálních záznamů.

4) Vrácení **výsledků:**

- Výsledky budou odeslány zpět klientovi ve formátu, který bude obsahovat průměrný počet psů pro prosinec 2019. Příklad výstupu:

```
time                        mean
----                        ----
2019-12-01T00:00:00Z        1888
```