Datová sada: Statistika průjezdu vozidel ze sledovaných křižovatek v roce 2024

url:
https://opendata.ostrava.cz/statistika-prujezdu-vozidel-ze-sledovanych-krizovatek-v-roce-2024/

Distribuce: csv

druh databáze: Apache Cassandra

Vysvětlení, proč je nejlepší k uložení a dotazování:

Cassandra má flexibilní schéma, což umožňuje dynamické ukládání různých sloupců bez nutnosti předem definovaného schématu. Široko-sloupcová struktura usnadňuje ukládání a agregaci dat (např. počty projetí pro různé stanice a časy)
Cassandra podporuje textové, timestamp a integer typy, což je ideální pro uložení údajů o stanici, třídě objektu, datu a počtu projetí.
Cassandra je navržena pro efektivní ukládání a správu velkých objemů dat jako jsou průjezdy aut křižovatkami.
Cassandra používá primární klíč složený z více sloupců pro určení unikátnosti, čehož jednoduše dosáhneme složením data a stanice.
Zdrojem dat jsou v tomto případě senzory na křižovatkách města. Pro tento typ databází jsou ideální data, které se generují v reálném čase a jsou pravidelně odesílány do systému. V našem případě se průjezdy křižovatkou agregují po dobu jedné hodiny než jsou odeslány.
Cassandra není ideální pro aktualizaci, nebo mazání dat, což se v našem systému nestává skoro nikdy, pouze kdyby došlo k poruše senzorů.
Cassandra může automaticky ukládat a spravovat agregované záznamy dat jako počty projetí za delší časové úseky.
Cassandra je horizontalně škálovatelná, což znamená, že lze snadno přidávat nové uzly do clusteru a rozšířit kapacitu, což je vhodné pro sledování dopravy, kde se periodicky přidává množství dat.
Cassandra zajišťuje redundanci dat díky replikaci, kde je záznam uložen na více uzlech, což zajišťuje vysokou rychlost a dostupnost. V příkladu dole jsou nastavené tři repliky dat.
Cassandra umožňuje cachování čtení a podporuje předpočítání souhrnných hodnot. V našem případě bude efektivní využívat předpočítané agregace, což umožňuje rychlý přístup k výsledkům bez nutnosti opakovaného výpočtu.

Příkazy pro definici úložiště:

```
CREATE KEYSPACE doprava
WITH replication = {'class': 'SimpleStrategy', 'replication_factor': '3'};

USE doprava;

CREATE TABLE statistika_prujezdu (
    datum timestamp,
    stanice text,
    trida_objektu text,
    pocet int,
    PRIMARY KEY (stanice, datum)
);
```

Algoritmický popis importu dat:

```
import pandas as pd
from cassandra.cluster import Cluster
from datetime import datetime

cluster = Cluster([ip_address])
session = cluster.connect('doprava')

file_path = 'Statistika-poctu-prujezdu-2024.csv'
data = pd.read_csv(file_path)

for _, row in data.iterrows():
    datum = datetime.strptime(row['Datum'], '%d.%m.%Y %H:%M')
    stanice = row['Stanice']
    trida_objektu = row['Třída objektu']
    pocet = row['Počet']

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

cluster.shutdown()
```

Dotaz v jazyce daného databázového produktu:

```
#počet průjezdů vozidel pro všechny stanici v daném datovém rozmezí
SELECT stanice, SUM(pocet) AS celkovy_pocet_prujezdu
FROM statistika_prujezdu
AND datum >= '2024-01-01' AND datum < '2024-02-01';
GROUP BY stanice;
```

Jak nalezne uzly, kde jsou uložena požadovaná data:

Jeden uzel přijme dotaz a stane se koordinačním uzlem. Pro každý partition key je spočítán hash pomocí hashovací funkce, který určí, na kterých uzlech jsou data uložena a kde se mají posílat požadavky.

Jak data z uzlů získá a dále distribuovaným způsobem zpracuje:

Koordinační uzel pošle dotaz na dané uzly. Na těchto uzlech se provede filtrace podle časového rozmezí a agregace. Každý uzel provádí práci pouze na své části dat. Výsledky poté vrací koordinátorovi.

Jak výsledky doručí klientovi, který zadal dotaz, a jak je tento zkonzumuje:

Koordinátor shromáždí výsledky z uzlů, sečte je a vrátí je zpět klientovi.Ten je následně analyzuje, nebo vizualizuje.

datová sada: Návrh kapitálového rozpočtu na r. 2024

url:https://data.gov.cz/datov%C3%A1-sada?iri=https%3A%2F%2Fdata.gov.cz%2Fzdroj%2Fdatov%C3%A9-sady%2F00845451%2F0e36ed6abc2f2621c37de0564dc791c5

distribuce: csv
druh databáze: MongoDB

Proč MongoDB:

Dokumentová databáze MongoDB se dobře na ukládání a dotazování strukturovaných, ale zároveň flexibilních dat. Mongodb ukládá data jako JSON dokumenty. Každý záznam v datasetu může být reprezentován jako dokument, kde se každý řádek v csv přeloží na MongoDB dokument. Dataset má hodně různých atributů. Flexibilní schéma MongoDB umožňuje ukládat data bez předdefinovaných struktur, což umožní jakékoliv budoucí změny jako změna, smazání nebo přidání nových atributů, bez toho aby se musela přestrukturovat celá databáze. MongoDB umožňuje snadno přidávat pole jako *updated_at*, které usnadňují sledování historických změn v rozpočtových údajích (pole v kterém si uchováváme čas změny). MongoDB zvládá jak předdefinované, tak ad-hoc dotazy. Pro tuto sadu to zahrnuje např. filtrování projektů podle roku nebo místa. Agregační pipeline MongoDB je silným nástrojem pro analytické dotazy (např. součty nákladů pro různé kategorie), což je pro rozpočtovou analýzu ideální. MongoDB podporuje složené indexy, což urychluje časté dotazy na klíčová pole, jako jsou *COL_PROJECT_ID* nebo *COL_COST_NEED_o*.

Vytvoření databáze a definice dat:

Vytvoříme databázi. Protože v datasetu mnohým záznamům chybí i důležité atributy jako project_id, tak nedefinujeme žádné schéma ani validaci. MongoDB vytvoří pole "*_id*" pro každý dokument, které se bude používat jako klíč

.

use rozpocetDB;
db.createCollection('rozpocet2024')

Prvotní naplnění databáze:

Pomocí příkazu z MongoDB CLI tools naimportujeme data přímo z csv. Jako názvy polí/atributů se použijí názvy sloupců v csv.

mongoimport --db rozpocetDB --collection rozpocet2024 --type csv --file C:\Users\Bahno\Desktop\skola\7sem\UPA\proj1\UPA-Project1\Project2\datasets\rozpocet.csv --headerline

Při upsert používám *_id* jako klíč, pokud není _id není specifikovánou použiji jako klíč kombinaci *COL_NAME* a *COL_COVER_PART*

```
from pymongo import MongoClient
from bson import ObjectId

client = MongoClient('mongodb://localhost:27017/')
```

```
db = client['rozpocetDB']
collection = db['rozpocet2024']

def upsert_record(record):
    if '_id' in record:
        key = {'_id': record['_id']}
    else:
        key = {'COL_NAME': record['COL_NAME'], 'COL_COVER_PAR':
record['COL_COVER_PAR']}
    collection.update_one(key, {'$set': record}, upsert=True)


record = {
    # '_id': ObjectId("672f7c0aa04f10d8ed8c287e")
    'COL_NAME': "Testovaci insert1",
    'COL_COVER_PAR': 3000,
    'COL_COST_ALL': 6000,
}

upsert_record(record)
```

Query:

```
db.rozpocet2024.aggregate([
  {
    $match: {
     COL_COST_IN_EXP_ALL: { $gt: 1000000 } // Projekty, které mají předpokládanou výši
dotace větší než milion
    }
  },
  {
    $group: {
    _id: "$COL_SUBJ_OWNER", // Group podle subject owner
      total_cost: { $sum: "$COL_COST_ALL" } // Suma projektů pro každého vlastníka
    }
  },
  {
    $sort: { total_cost: -1 } // Seřadím od největší po nejmenší
  },
  {
    $limit: 5 // Ukážu pouze top 5 výsledků
  }
])
```

MongoDB používá shard klíč k distribuci date mezi shardy. Pokud je kolekce shardována MongoDB rozdělí data mezi shardy pomocí sharovacího klíče. MongoDB zjistí, který shard má data pro *COL_COST_IN_EXP_ALL*. MongoDB použije full scan na odpovídajícím shardu. Po zjištění, které shardy mají odpovídající dokumenty, MongoDB distribuuje dotaz na všechny relevantní shardy. Každý shard provede operaci filtrace (*$match*), aby našel projekty s předpokládanou výší dotace větší než milion. Po provedení filtrace na jednotlivých shardech, MongoDB použije operaci agregace $group, aby seskupil výsledky podle hodnoty *COL_SUBJ_OWNER* a spočítal celkový náklad (*COL_COST_ALL*) pro každého vlastníka projektu. Tato agregace probíhá lokálně na každém shardu. Po provedení agregace na každém shardu MongoDB **shromáždí výsledky** z těchto shardů a **sloučí je**. Výsledky seřadí podle *total_cost* a omezuje výstup na top 5 subjektů pomocí *$limit*. Po dokončení agregace a sloučení výsledků z různých shardů MongoDB vrátí zpracované výsledky ve formátu JSON. Tento formát obsahuje seznam subjektů (*COL_SUBJ_OWNER*) a jejich celkových nákladů (*total_cost*), které odpovídají kritériím dotazu.

Datová sada: Cyklostezky

url: https://data.gov.cz/datov%C3%A1-sada?iri=https%3A%2F%2Fdata.gov.cz%2Fzdroj%2Fdatov%C3%A9-sady%2F00845451%2F37085191

Druh databáze: Neo4J

Distribuce: GeoJSON (vhodné pro geodata a prostorové dotazy).

Proč Ne04J:

Neo4j umožňuje modelovat propojené entity jako uzly a vztahy mezi nimi. U datasetu cyklostezek jsou důležité například počáteční a koncové body tras a jejich propojení, které lze snadno reprezentovat jako uzly a hrany. **geometry.coordinates**: Souřadnice definují geometrii cyklostezky a umožňují modelovat uzly (start a konec úseku) a vztahy (trasa mezi těmito body). Neo4j obsahuje Spatial plugin, který umožňuje provádět prostorové dotazy (například vyhledání tras v blízkosti určitého bodu). To je klíčové pro cyklostezky, které mají přímé prostorové vazby. **LENGTH**: Délka trasy může být použita jako vlastnost vztahu mezi uzly, což je užitečné při vyhledávání nejkratších nebo nejdelších tras. Grafové databáze umožňují snadné přidávání nových vlastností bez nutnosti změny struktury databáze. To je výhodné pro případ, kdy by se do datasetu přidávaly nové atributy nebo vlastnosti. **POVRCH**: Povrch trasy je příkladem atributu, který by mohl být přidán jako vlastnost uzlu či vztahu a může být využit pro filtrování při dotazech. Neo4j podporuje komplexní analýzu propojených dat (například síťové analýzy, hledání sousedních uzlů, trasování cest mezi body). Propojení tras na základě jejich vzájemné návaznosti lze modelovat jako vztahy mezi uzly. To umožní efektivní dotazování, například na nejbližší sousední trasy nebo na propojené stezky v konkrétní oblasti.

Vytvoření databáze a definice dat:

V Neo4j není potřeba explicitně vytvářet databázové schéma předem (na rozdíl od relačních databází), protože Neo4j pracuje se schematicky volným modelem.

Vytvoření uzlů pro cyklostezky

```
CREATE (:Cyklostezka {
    id: 'TRASA-5',
    delka: 909.566927,
    typ: 4,
    povrch: 'nedefinovano',
    povrch_kod: 0,
    popis: 'cyklistický pruh na vozovce',
    pom: '5-4-nedefinovano'
});
```

Vložení geografických bodů

```
CREATE (:Bod {
    id: 'BOD-1',
    x: -478109.94800701784,
    y: -1095824.2460840154
});

CREATE (:Bod {
    id: 'BOD-2',
    x: -477829.88551876246,
    y: -1094975.2628272104
});
```

Vytvoření vztahů mezi cyklostezkou a body

```
MATCH (c:Cyklostezka {id: 'TRASA-5'}), (b1:Bod {id: 'BOD-1'}), (b2:Bod {id: 'BOD-2'})
CREATE (c)-[:ZAČÍNÁ]->(b1),
       (c)-[:KONČÍ]->(b2);
```

Prvotní naplnění databáze:

1. Načtení dat z GeoJSON souboru:
    a. Otevřete a načtěte GeoJSON soubor obsahující data o cyklostezkách a jejich souřadnicích.
    b. Pro každý prvek `Feature` extrahujte potřebné vlastnosti (např. `id`, `LENGTH`, `TRASA`, `CYKLO`, `POVRCH`, souřadnice).
2. Iterace nad daty a zajištění UPSERT operací:
3. Kontrola existence cyklostezky podle jejího id:
    a. Pokud cyklostezka existuje, aktualizujte její vlastnosti.
    b. Pokud cyklostezka neexistuje, vytvořte nový uzel a propojte ho s odpovídajícími geografickými body.

```python
import json
from neo4j import GraphDatabase

# Připojení k Neo4j databázi
uri = "neo4j://localhost:7687"
username = "neo4j"
password = "password"  # Upravte podle svého nastavení
driver = GraphDatabase.driver(uri, auth=(username, password))

def import_data(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        geojson_data = json.load(file)

    with driver.session() as session:
        for feature in geojson_data['features']:
            # Extrakce vlastností
            cyklostezka_id = feature['properties']['TRASA']
            length = feature['properties']['LENGTH']
            typ = feature['properties']['CYKLO']
            povrch = feature['properties']['POVRCH']
            popis = feature['properties']['POPIS']
            coordinates = feature['geometry']['coordinates']

            # UPSERT operace pro cyklostezku
            session.run("""
                MERGE (c:Cyklostezka {id: $cyklostezka_id})
                ON CREATE SET c.delka = $length, c.typ = $typ, c.povrch = $povrch, c.popis = $popis
                ON MATCH SET c.delka = $length, c.typ = $typ, c.povrch = $povrch, c.popis = $popis
            """, cyklostezka_id=cyklostezka_id, length=length, typ=typ, povrch=povrch, popis=popis)

            # Vložení nebo aktualizace geografických bodů a propojení
            for idx, (x, y) in enumerate(coordinates):
                point_id = f"{cyklostezka_id}_BOD_{idx}"
                session.run("""
                    MERGE (b:Bod {id: $point_id})
                    SET b.x = $x, b.y = $y
                    MERGE (c:Cyklostezka {id: $cyklostezka_id})
                    MERGE (c)-[:ZAČÍNÁ]->(b)  // Pro první bod
                    MERGE (c)-[:KONČÍ]->(b)  // Pro poslední bod (podmíněné spojení)
                """, point_id=point_id, x=x, y=y, cyklostezka_id=cyklostezka_id)

# Spuštění importu
import_data('cyklo_JTSK.geojson')
```

Dotazy:

Pro dotaz nad datasetem cyklostezek uloženým v Neo4j použijeme Cypher dotaz, který vyhledá všechny cyklostezky s konkrétním typem povrchu a vrátí jejich délku, popis a souřadnice jejich trasy.

```
MATCH (c:Cyklostezka)-[:ZAČÍNÁ|:KONČÍ*]->(b:Bod)

WHERE c.povrch = 'kvalitní'

RETURN c.id AS CyklostezkaID, c.delka AS Delka, c.popis AS Popis, collect([b.x, b.y]) AS Souradnice

ORDER BY c.delka DESC
```

Vyhledá všechny uzly označené jako **Cyklostezka**. Propojuje uzel cyklostezky s jejími geografickými body pomocí vztahů **ZAČÍNÁ** nebo **KONČÍ**. Hvězdička () umožňuje sledovat více vztahů najednou, čímž lze vyhledávat trasu celé cyklostezky. **WHERE c.povrch = 'kvalitní'**: Filtruje výsledky tak, aby byly vráceny pouze cyklostezky s konkrétním povrchem. Vrací **id**, **delka**, a **popis** pro každou cyklostezku. **collect([b.x, b.y])** seskupuje souřadnice všech bodů do seznamu, který odpovídá trase cyklostezky. Výsledky jsou seřazeny podle délky cyklostezky sestupně.

Databáze začne vyhledáváním uzlů označených jako **Cyklostezka** a pokračuje spojováním těchto uzlů s uzly **Bod** pomocí vztahů **ZAČÍNÁ** a **KONČÍ**. Pokud Neo4j běží v clustru, dotaz může být rozdělen a zpracován distribuovaným způsobem. Dotazovací engine paralelně zpracovává jednotlivé části grafu, aby byl dotaz rychlejší. Po vykonání dotazu na databázi se výsledky vrátí v podobě tabulky nebo seznamu záznamů, kde každý záznam obsahuje požadované hodnoty (například **CyklostezkaID**, **Delka**, **Popis** a **Souradnice**). Klient obdrží výsledky dotazu ve formě záznamů a zpracuje je podle potřeby.

Datová sada: Dopravní přestupky dle data a místa spáchání v roce 2024

url:
https://data.gov.cz/datov%C3%A1-sada?iri=https%3A%2F%2Fdata.gov.cz%2Fzdroj%2Fdatov%C3%A9-sady%2F00845451%2Fdec532858bf68c8b8846b79ebe8ea211

Druh databáze: InfluxDB

Distribuce: CSV

Proč InfluxDB:

Každý záznam v datasetu obsahuje pole **„Datum spáchání“**, což umožňuje snadno sledovat a analyzovat trend přestupků v čase, což je jeden z klíčových aspektů InfluxDB. Databáze InfluxDB je optimalizovaná pro časové řady a umožňuje efektivní ukládání a dotazování nad těmito daty. Mužeme tedy například vytvořit reporty, jako je přehled počtu přestupků nebo trendů bodových sankcí v čase. InfluxDB umožňuje výkonné agregace a výpočty na základě časových údajů, jako jsou průměry, sumy či maxima v určitém časovém rozsahu, což je velmi vhodné pro analýzu trendů přestupků podle dnů, měsíců, či roků. Sloupec **„Bod“** obsahuje bodové hodnocení přestupků, které lze pomocí InfluxDB efektivně agregovat a vypočítávat průměry, součty, maxima či minima v definovaných časových intervalech (např. měsíční nebo roční sumy). Sloupec **„Způsob vyřízení“** obsahuje typy vyřízení přestupků a umožňuje kategorizaci záznamů na základě hodnot, což usnadňuje analýzu, jakým způsobem byly jednotlivé přestupky řešeny v určitém čase. InfluxDB podporuje politiku retence, takže lze snadno nastavit dobu uchování historických dat a šetřit místo v databázi, aniž by byla ohrožena dostupnost relevantních dat pro nedávné dotazy. To se může hodit, pokud jsou historická data o přestupcích po určité době méně významná a lze je shrnout do agregovaných dat, případně smazat.

Vytvoření databáze a definice dat:

InfluxDB nepoužívá klasické schéma jako relační databáze, lze ale vytvořit tagy jako takovou kategorizaci.

```
CREATE DATABASE dopravni_prestupky
USE dopravni_prestupky

CREATE MEASUREMENT prestupky
TAG Datum_spachani
TAG Misto_cinu
TAG Zpusob_vyrizeni
```

Prvotní naplnění databáze:

Data obvykle vkládají ve formátu Line Protocol. Tento formát je jednoduchý, textový formát, který zahrnuje měření, tagy, pole a časové razítko.

Použijte následující příkaz pro import do InfluxDB:

influx -import -path=prestupky_lineprotocol.txt -precision=s
-database=dopravni_prestupky

Nejprve je ale potřeba upravit data. Pro tento účel můžete použít například následující Python skript, který převede každý řádek CSV do formátu Line Protocol a uloží ho do textového souboru.

```python
import pandas as pd
from datetime import datetime

# Načtení CSV souboru
data = pd.read_csv('/path/to/dopravniprestupky.csv')

# Funkce pro převod na Line Protocol
def to_line_protocol(row):
    measurement = "prestupky"
    tags = f"Misto_cinu={row['Místo činu']},Zpusob_vyrizeni={row['Způsob vyřízení']}"
    fields = f"Paragraf=\"{row['Paragraf']}\",Odstavec=\"{row['Odstavec']}\",Pismeno=\"{row['Písmeno']}\",Bod={row['Bod']}"
    timestamp = int(datetime.strptime(row['Datum spáchání'], "%Y-%m-%d").timestamp())
    return f"{measurement},{tags} {fields} {timestamp}"

# Převedení celého souboru a uložení do textového souboru
with open('/path/to/prestupky_lineprotocol.txt', 'w') as f:
    for _, row in data.iterrows():
        f.write(to_line_protocol(row) + "\n")
```

Dotazy:

SELECT MEAN(Bod)

FROM prestupky

WHERE time > now() - 1y

GROUP BY time(1mo), Paragraf

Dotaz hledá průměr hodnot bodů pro každý měsíc posledního roku.InfluxDB při vykonání dotazu nejprve analyzuje, které části datového úložiště obsahují záznamy v časovém rozsahu **time > now() - 1y**. Díky časové optimalizaci a tomu, že InfluxDB indexuje časová razítka záznamů, rychle identifikuje, které úseky (nazývané shard) databáze obsahují požadovaná data z posledního roku. Vyhledá tag **Paragraf** a provede filtrování, aby pouze data s touto kategorií zahrnula do dotazu.

InfluxDB z každého identifikovaného shardu načte odpovídající datové body (např. hodnoty sloupce **Bod** pro každý záznam). Při distribuci může databáze využít paralelizaci, pokud je nasazena v distribuovaném prostředí (např. na vícero uzlech), což zrychluje načítání dat z více shardů současně. V případě distribuované architektury každý uzel, který obsahuje příslušné shardy, vyfiltruje data na své úrovni a posílá předzpracované výsledky. Jakmile

jsou data načtena, InfluxDB aplikuje agregační funkci **MEAN** pro hodnoty **Bod** na měsíční úrovni (**GROUP BY time(1mo)**). Tato agregace probíhá na úrovni jednotlivých uzlů, což minimalizuje přenos dat mezi uzly a centrální částí databáze.

Po provedení agregace na jednotlivých uzlech se InfluxDB postará o seskupení všech částečných výsledků a vytvoření kompletní odpovědi na dotaz. Tyto výsledky se poté doručí klientovi ve formě řady časových bodů se sloučenými daty (průměrnými hodnotami **Bod**) na měsíční bázi. Tyto výsledky lze následně snadno použít k vizualizaci trendů v čase, například zobrazit měsíční průměrné hodnoty bodových sankcí dle paragrafů, což je užitečné pro analýzu a přehled o dlouhodobých trendech.