

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

UKLÁDÁNÍ A PŘÍPRAVA DAT 2023/2024

Projektová dokumentácia

Michal VAŇO
xvanom00
Pavel KRATOCHVÍL
xkrato61

Zadanie:
UKLÁDÁNÍ
ROZSÁHLÝCH DAT V
NOSQL DATABÁZÍCH

21.10.2023



1 Úvod

Cieľom tejto časti projektu bolo preskúmať štyri rôzne orientované NoSQL databázy. Rozhodli sme sa vyskúšať ich nielen teoreticky, ale aj prakticky. Pre tieto účely sme spojizdnilí všetky štyri druhy databáz cez Docker, aby sme mohli lokálne skúšať ich schopnosti.

2 Dokumentová databáza – MongoDB

Vybraným dokumentovo orientovaným databázovým produktom pre tento projekt je MongoDB¹. Dáta ukladané na MongoDB server sú členené do databáz, kde databáza (database) uchováva kolekcie (collection) dokumentov (document). Dáta sú uložené vo formáte BSON² (binárna reprezentácia formátu JSON).

Dokumenty v kolekcii sú zložené z položiek–názvu a hodnoty, kde hodnota môže byť akéhokoľvek typu podporovaného vo formáte BSON (napr. dokument, pole, timestamp, reťazec, int, double, decimal). `_id` je rezervovaným názvom položky identifikujúci primárny kľúč. Hodnota položky `_id` je unikátna v kolekcii, nemenná a môže byť akéhokoľvek dátového typu okrem poľa. Východnou hodnotou je automaticky generovaná hodnota typu `ObjectId` veľkosti 12 B (obsahuje timestamp, náhodné číslo a hodnotu čítača pre unikátnosť v rámci kolekcie).

Medzi výhody dokumentovo orientovaných databáz možno zaradiť flexibilné dokumentové schémy. Schéma môže obsahovať akúkoľvek štruktúru dát ľubovoľnej veľkosti obmedzenej iba maximálnou veľkosťou BSON dokumentu (16 MB). Navyše, schémy dokumentov v kolekcii nemusia byť homogénne, čo umožňuje zmeny v schémach počas vývoja aplikácie alebo rastu datasetu.

Pre využitie predností poskytovaných databázou MongoDB sme z **Národného katalógu otvorených dát pre štatutárne mesto Brno** vybrali dátovú sadu **Pohřbené významné osobnosti**.

¹<https://www.mongodb.com/docs/>

²<https://bsonspec.org/>

2.1 Popis vybranej dátovej sady a vhodnosti použitia MongoDB

Vybraná dátová sada je dostupná vo formáte CSV, ShapeFile, KML a mnohých ďalších, no my sme zvolili formát **GeoJSON**³ pre jeho prehľadnosť a jednoduchosť použitia v programovacom jazyku Python (s využitím štandardnej knižnice `geojson`⁴), v ktorom sme zostavili skript na parsovanie datasetu, ošetrovanie chybných položiek v dátach počiatočné naplnenie kolekcie. Jednou z praktických funkcií MongoDB je natívna podpora objektového typu GeoJSON umožňujúca pokročilé **priestorové dotazy** (geospatial queries) nad dátovou sadou. Príklady takýchto dotazov sme uviedli v sekcii 2.5.

Dataset obsahuje informácie o zhruba 800 brnenských osobnostiach, ich významnosti a taktiež obsahuje informácie o mieste ich posledného odpočinku (súradnice hrobového miesta). Tieto dáta môžu byť využité pre rôzne historické, kultúrne a genealogické výskumy. Podrobná štruktúra dát a dátové typy položiek sú znázornené v Schéme 2.

Dokumentovo orientovaný typ DB (a konkrétne nami vybraný produkt MongoDB), je jediný, ktorý dokáže efektívne uspokojiť všetky naše požiadavky. Vďaka flexibilnej schéme a voľnosti dátových typov položiek (prípadne nestingu) je možné dátovú sadu ľahko rozšíriť o poznatky z iného zdroja informácií (návrh v Sekcii 2.2) čo je veľké pozitívum oproti stĺpcovej DB.

Taktiež vďaka výbornej dátovej lokalite, teda že všetky informácie o konkrétnej osobnosti sa nachádzajú na jednom mieste (v jednom dokumente), dokážeme informácie z DB získať rýchlo a efektívne. Túto vlastnosť by nebolo možné dosiahnuť v time-series DB, keďže tento typ sa sústreďuje na uchovávanie časovo zoradených záznamov (a často aj s homogénnou štruktúrou). Ďalšou výhodou oproti time-series DB je, že MongoDB nám umožňuje veľmi flexibilné dotazy a rozne typy indexov podľa toho, aké dotazy plánujeme v budúcnosti vykonávať.

Grafovo orientovaná databáza by bola pre vybranú dátovú sadu veľmi nevhodná, keďže sa v nej nenachádzajú žiadne komplexné vzťahy medzi entitami, ktoré by sme potrebovali uchovávať. Isté vzťahy by sme medzi osobnosťami našli (aj v externých zdrojoch, Sekcia 2.2), no toto je ľahšie dosiahnuť

³<https://datatracker.ietf.org/doc/html/rfc7946>

⁴<https://github.com/jazzband/geojson>

1	long int	ogc_fid	# jednoznacny identifikator zaznamu
2	string	title	# meno a priezvisko osobnosti
3	string	description	# povolanie vyznamnej osobnosti
4	string	url_1	# odkaz encyklopedie dejin Brna
5	date	create_date	# datum vytvorenia zaznamu
6	date	last_date	# datum poslednej editacie zaznamu
7	double int	numeric_1	# cislo prehliadkovej trasy
8	date	date_1	# datum narodenia
9	date	date_2	# datum umrtia
10	long int	list_1	# charakteristika povolania

Schéma 1: Štruktúra, dátové typy a názvy položiek vybranej dátovej sady Pohrbené významné osobnosti

odkazom cez atribút dokumentu ObjectID, ktorý by slúžil ako odkaz na iný dokument.

2.2 Napojenie na externé dáta

Dáta môžu byť prepojené na základe geografických súradníc s inými geografickými dátami, čo umožňuje ďalšie analýzy a zobrazenia.

Dátovú sadu je jednoduché automatizovane rozšíriť minimálne o informácie z Internetovej encyklopédie dejín Brna⁵, keďže ku takmer každej uvedenej osobnosti v dátovej sade je uvedený link na profil v encyklopédii. Preto sme tento fakt využili, a dáta nahrávané do MongoDB sme obohatili o vybrané informácie z encyklopédie.

2.3 Zmeny dát v čase, aktualizácie

Na stránke dátovej sady sme zistili, že sada má nepravidelnú periódu aktualizácie. V súčasnosti obsahuje dátová sada iba osobnosti pochované na v areáli Ústredného hřbitova, avšak sú plány na rozširovanie o ďalšie osobnosti. Môžeme teda predpokladať, že dôjde k niekoľkým väčším zmenám (pridanie známych osobností z ďalších cintorínov) ako aj malým, sporadickým a inkrementálnym zmenám, prípadne opravám existujúcich dát. Taktiež by zvolený databázový produkt umožňoval pridávanie ďalších položiek do dokumentov. Ak by napríklad niektorý cintorín poskytoval unikátne číslovanie hrobových

⁵<https://encyklopedie.brna.cz>

miest prípadne označenie častí cintorína, bolo by praktické mať túto informáciu priamo v dátovej sade.

Ako si môžeme všimnúť v Schéme 2, každá položka obsahuje záznam o čase poslednej úpravy (`last_date`), čo umožňuje sledovať zmeny v dátach a aktualizovať ich podľa potreby.

MongoDB umožňuje metódou `db.collection.update(query, update, options)` aktualizovať celý dokument ako aj iba niektoré položky. Po zistení zmeny v dátovej sade by sme zrejme vyhľadávali v databáze podľa `_id` (nemenná a unikátna vrámci kolekcie) použiteľ v `query` ako parameter v metóde `update()` a parameter `update` by obsahoval buď celý dokument alebo iba niekoľko zmenených záznamov.

2.4 Spôsob a priebeh vzniku dát

Podľa informácií uvedených na uvedenej stránke dátovej sady vytvorili dáta pracovníci Múzea mesta Brna a kolektívu autorov Internetovej encyklopédie dejín Brna. Môžeme predpokladať, že autori sú kvalifikovaný vo svojom obore a teda, že ide o veľmi kvalitný zdroj informácií s malou chybovosťou. Tomu nasvedčuje aj homogénnosť dát a veľmi nízka frekvencia absencií niektorých položiek. Napríklad, iba 9 z celkovo 810 obsiahnutých známych osobností nemá uvedený link na profil na spomínanú encyklopédiu. Mnoho ľudí nemá uvedený presný dátum narodenia (14) alebo úmrtia (39), čo však možno očakávať, keďže v mnohých prípadoch sa o nich takáto informácia nedochovala a je známy iba rok.

Aj keď MongoDB podporuje TTL index pre automatické vymazanie dokumentu (po určitej dobe alebo v špecifikovaný čas), v prípade vybranej dátovej sady to nie je žiadúce, keďže ide o historické záznamy určené na dlhodobé uchovanie.

Možná kompresia dát je minimálna, keďže každý dokument obsahuje unikátne hodnoty záznamov a ich počet a názvy sa môžu medzi dokumentmi líšiť. Z tohto dôvodu nemožno dáta ani rozumne agregovať. Jeden zo spôsobov ako dosiahnuť pamäťovú úsporu by mohlo byť skrátenie položky `url_1`, ktorá obsahuje vždy celú URL na encyklopédiu. Túto položku by bolo možné skrátiť iba na samotné identifikačné číslo profilu osobnosti, ktoré sa nachádza na konci URL ako tzv. query string. Keďže je ale počet záznamov relatívne (810) aj veľkosť celej kolekcie (296 kB bez indexov) nízky, rozhodli sme sa hodnotu záznamu neskracovať.

MongoDB neumožňuje doprednú definíciu dotazov, dáta môžu byť spot-

rebované iba prostredníctvom ad-hoc dotazov. Výnimkou sú serverless JavaScript funkcie v ak by sme využili funkcie ponúkané službou MongoDB Atlas, to je však nad rámec tohto zadanie.

Geografické dáta v spomínanom natívnom priestorovom (*geospatial*) formáte umožňujú rôzne dotazy s ohľadom na vzdialenosti jednotlivých bodov alebo území. Napríklad môžeme v našom prípade vykonať dotaz na všetky hrobové miesta na nejakom území alebo vo vzdialenosti od istého bodu (zadaného súradnicami). Frekvencia čítania dát závisí od konkrétneho využitia, ale dáta by mohli byť dôležité pre historický alebo genealogický výskum.

MongoDB ponúka možnosti indexácie, ktoré môžu urýchliť vyhľadávanie a dotazy nad dátami. Ukážkovým príkladom je vytvorenie indexu na priestorové (*geospatial*) položky v dokumente, ktoré je použité aj v priloženom skripte. Ďalej by mohol byť užitočný index na textové vyhľadávanie v prípade, ak by sme dáta obohatili o obsirnejší popis z Encyklopédie dejín Brna alebo z iného zdroja.

Pre umožnenie uchovávanie veľkého množstva dát a horizontálneho škálovania je MongoDB schopná *sharding-u*. Dáta sú potom rozdelené do menších častí, ktoré môžu byť efektívne umiestnené na viacerých serveroch. Tým sa zvyšuje dostupný výpočetný výkon ako aj dostupný úložný priestor.

2.5 Dotaz na MongoDB

```
1 start_date = datetime(1940, 1, 1)
2 end_date = datetime(1940, 12, 31)
3 location_point = [16.591516, 49.172261]
4 max_distance = 1000
5
6 results = col.find({
7     'date_1': {'$gte': start_date, '$lte': end_date},
8     'location': {
9         '$near': {
10             '$geometry': {
11                 'type': "Point",
12                 'coordinates': location_point
13             },
14             '$maxDistance': max_distance,
15         }
16     }
17 }, {
18     'title': 1,
19     'date_1': 1
20 })
```

1. Dotaz - klient zašle sformulovaný dotaz vo formáte BSON. Vďaka tomu môžu byť do dotazu vložené dátové typy, ktoré by vo formáte JSON byť nemohli.
2. Skenovanie indexov - ak existuje index na dotazované položky dokumentov, MongoDB ich použije na rýchle vyhľadanie.
3. Document Fetching - dokumenty vyhovujúce dotazu sú skopírované do operačnej pamäte.
4. Filtrovanie - Skopírované dokumenty sú následné filtrované podľa ďalších podmienok v dotaze. Toto v našom prípade zahŕňa kontrolu rozsahu položky `date_1` a kontrolu vzdialenosti od zadaného bodu v položke `Location`
5. Projekcia - určuje, ktoré položky zahrnúť vo výsledku. Napríklad, výsledok nášho dotazu bude obsahovať záznamy iba s `title` a `date_1`.
6. Vrátenie výsledku

3 Stĺpcová wide-column databáza – Apache Cassandra

Dôvodom obľúbenosti Cassandra DBMS⁶ (database management system) pre využitie v globálne rozšírených produktoch ako je Netflix, Spotify, je škálovateľnosť a distribuovanosť celého systému medzi viacerými fyzickými uzlami, aj keď pre užívateľa sa stále javí ako jediný celok.

Distribúcia dát medzi viacerými uzlami a rýchly zápis a čítanie je možné vďaka efektívnemu ukladaniu dát. Na najvyššej úrovni sú dáta rozdelené podľa **keyspace**, ktorý ohraničuje celky dát a každý **keyspace** tvorí nezávislú entitu. V definícii **keyspace** užívateľ definuje replikačný parameter (**replication_factor** alebo **datacenter_name** pre replikáciu cez niekoľko dátových centier) určujúci množstvo kópií dát v celom systéme. Pre náš účel je postačujúci jediný **keyspace** (pomenovaný **upa**), keďže budeme pracovať v rámci jedného celku dát. Replikačný faktor môžeme pre účel tohto projektu nastaviť na 1 (možné rozšírenie v opísané v Sekcii 3.3).

Ďalej sú dáta členené na tzv. **column families**, ktorých koncept je ekvivalentný s konceptom **table** v relačnej databáze. **Column family** obsahuje jeden alebo viac **rows** a každý **row** potom obsahuje niekoľko usporiadaných hodnôt (názov **column** je namapovaný na hodnotu v **row**).

Veľmi dôležitou súčasťou Cassandra DB sú tzv. **partition keys** slúžiace (po zahashovaní) ako look-up value a určujú rozdelenie dát medzi uzlami. Hodnoty, ktoré budú v **table** slúžiť ako **partition keys** sú definované už pri vytvorení daného **table**. Rows s rovnakým **partition key** budú teda uložené na jednom uzle. Príklad vytvorenia **table** je vidieť na Obrázku 3.2.1, kde parameter v **PRIMARY_KEY()** určuje **partition key** (môže byť aj n-tica). Následné položky určujú zoradenie záznamov (v tomto prípade budú stabilne zoradené podľa **code** a následne podľa **actualized**).

Keďže jeden dotaz na Cassandra DB by nemal mať presah cez viacero **partitions**, musí byť pre zachovanie efektivity spôsob organizácie dát a voľba **partition keys** prispôbena prevalentnému typu zamýšľaných dotazov.

⁶<https://cassandra.apache.org/doc/latest/>

3.1 Vlastnosti Datasetu

Aj keď Cassandra DB je systém bežne využívaný na aplikácie s miliónmi zázpismi a dotazmi za sekundu, v našom prípade demonštrujeme jej funkcionálnosť na menšej dátovej sade **Kvalita ovzdušia z Národného katalógu otvorených dát pre štatutárne mesto Brno**. Sada obsahuje periodické merania zo staníc (Obrázok 3.2.1) na území mesta Brna. Podľa popisu sa nejedná o verifikované dáta, no pochádzajú z kvalifikovaného zdroja - staníc Českého hydrometeorologického ústavu (ČHMÚ⁷) a Státního zdravotnického ústavu Ostrava⁸, takže dáta môžeme pre účely tohto projektu pokladať za spoľahlivé.

3.2 Popis vybranej dátovej sady a vhodnosti použitia Cassandra

Dátová sada pozostáva zo série záznamov zachytávajúcich určité metriky z konkrétného miesta v čase. Napríklad meranie zo stanice v Brne-Tuřany (kód: BBNYA, majiteľ stanice: ČHMÚ) zachytávajúce hodnoty škodlivých chemických zlúčenín a pevných častíc v ovzduší.

3.2.1 Organizácia uloženia v Cassandra DB

Prístup k dátam (SELECT, UPDATE, INSERT, DELETE) vyžaduje znalosť hodnoty (hodnôt) zadefinovaných ako primárny kľúč, pretože vyhľadávanie dát prebieha pomocou vyhľadávania zahešovaného primárneho kľúča (alebo n-tice).

Hypotetickým príkladom by mohlo byť vytváranie štatistiky zo všetkých meraní z konkrétnych staníc. Chceli by sme teda dáta ukladať a uchovávať ich v databáze ľahko prístupné na základe mena a kódu identifikujúceho konkrétnu stanicu a ďalej by mohli byť logicky zoradené podľa času merania (*actualized*). Keďže sa v hodnote položky **name** uvádza iba názov mestskej časti (Brno-Tuřany), nemusel by nám v budúcnosti jednoznačne identifikovať konkrétnu stanicu (aj keď teraz to platí, viď Obrázok 3.2.1 a Obrázok 3.2.1), preto by sme do primárneho kľúča zahrnuli aj kód stanice (zrejme unikátne). Zvyšné položky nepotrebujeme mať v primary key, keďže podľa

⁷<https://www.chmi.cz>

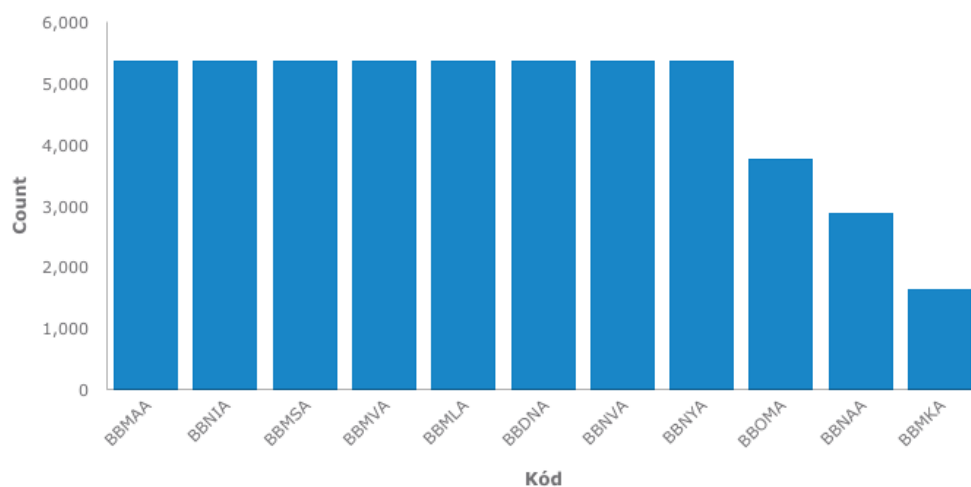
⁸<https://www.zuova.cz>

nich nebudeme vyhľadávať, ale naopak, očakávame ich ako výsledok vykonávaných dotazov. Výsledný príkaz na vytvorenie požadovaného `table` je uvedený na Obrázku [3.2.1](#).

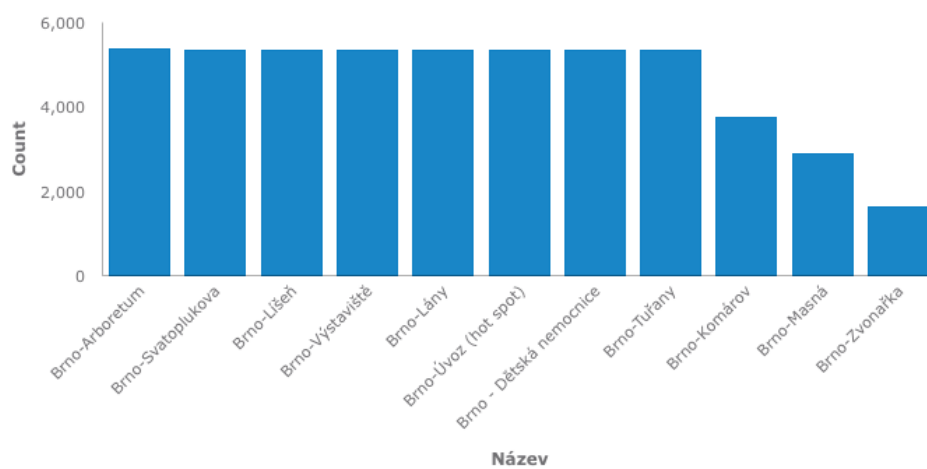
Jednou z výhod využitia Cassandra DB práve pre nami vybranú dátovú sadu je možnosť neuvádzať pre daný `row` hodnoty všetkých `columns`. Ako si môžeme všimnúť v dátovej sade, žiadne z meraní neobsahuje úplne všetky zbierané metriky (napr. `N02_1h`, `CO_8h`, atď.).

```
CREATE TABLE measurements (  
    objectid    int,  
    code        text,  
    name        text,  
    owner       text,  
    lat         float,  
    lon         float,  
    actualized  timestamp,  
    so2_1h      float,  
    no2_1h      float,  
    co_8h       float,  
    pm10_1h     float,  
    o3_1h       float,  
    pm10_24h    float,  
    pm2_5_1h    float,  
    globalid    text,  
    PRIMARY KEY ((name, code), actualized)  
)
```

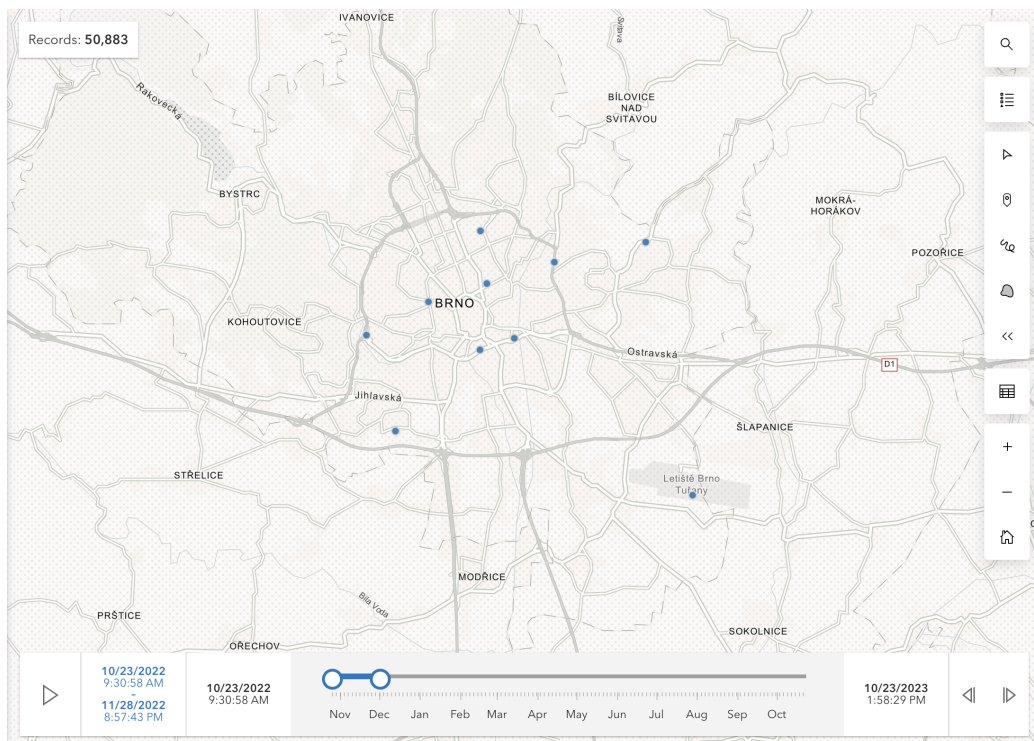
Obr. 1: CQL príkaz na vytvorenie table, ktorý bude uchovávať namerané hodnoty kvality ovzdušia zo staníc v Brne.



Obr. 2: Počty výskytu jednotlivých kódov staníc v dátovej sade Kvalita ovzdušia[1].



Obr. 3: Počty výskytu rôznych mien v záznamoch v dátovej sade Kvalita ovzdušia[1]



Obr. 4: Rozmiestnenie meracích staníc podľa súradníc uvedených v dátovej sade Kvalita ovzdušia[1].

```

1 string code
2 string name
3 string owner
4 double lon
5 double lat
6 date actualized
7
8 string N02_1h # kazdou hodinu
9 string CO_8h # kazdych 8h
10 string PM10_1h # kazdou hodinu
11 double O3_1h # kazdou hodinu
12 double PM10_24h # 24h prumer
13 string PM2_5_1h # kazdou hodinu

```

Schéma 2: Štruktúra, dátový typ jedného záznamu v dátovej sade Kvalita

ovzdušia. Pre hodnoty merania je v komentári uvedená perióda aktualizácie.

Ako nedostatok dokumentácie dátovej sady by sme označili nezdokumentované významy skratiek, označení a neuvedené jednotky jednotlivých metrik (NO₂, CO, atď.). Keďže ide o bežne merané metriky ovzdušia, nebol problém ich dohľadať, no zrejme by mali byť všetky tieto údaje prítomné v dokumentácii. Domnievame sa, že ide o hladiny nasledujúcich chemických zlúčenín alebo častíc:

- NO₂ - Oxid dusičitý
- CO - Oxid uhoľnatý
- PM₁₀ - pevné častice veľkosti menšej než 10 µm
- O₃ - Ozón
- PM_{2.5} - pevné častice veľkosti menšej než 2.5 µm

Iné typy NoSQL databáz by na zvolenú dátovú sadu neboli vhodné. Grafová je nevhodná, keďže tu neexistuje žiadny typ komplexného vzťahu medzi entitami, ktorý by bol cieľom modelovania a ukladania.

Dokumentová by bola o čosi použiteľnejšia, no dátová sada je pomerne riedka (nenachádzajú sa tu hodnoty pre každú metriku v každom zázname), takže by boli dáta zbytočne neefektívne uložené (napríklad vo formáte BSON). Ďalšou výhodou oproti dokumentovej databáze je škálovateľnosť Cassandra DB. Táto výhoda by sa prejavila obzvlášť pri zbere dát z viacerých miest z mnoho staníc, kedy by škálovateľnosť dokumentových databáz nemusela stačiť a kde by naopak Cassandra excelovala a zaručovala by konzistenciu a prístupnosť dát.

Databázový typ, ktorý sa jasne ponúka na danú dátovú sadu by mohla byť databáza na časovú radu (time series). Oproti takej databáze má ale wide-column jasnú výhodu vo flexibilitě a verzatilitě možných dotazov. Navyše, v našom prípade môže byť v každom meraní niekoľko hodnôt, ktoré chceme uchovať. To by mohol byť problém, ktorý by si vyžadoval rozdelenie každého nameraného agregovaného záznamu na viacero jednoduchších (name, code, actualized a jediná nameraná metrika), čo by viedlo na redundanciu. (name, code, actualized)

3.3 Napojenie na externé dáta

Dátová sada by mohla byť ľahke rozšírená o merania z ďalších staníc. Pekným projektom by bol zber dát vo viacerých mestách a ich distribuované uloženie v Cassandra DB uzloch pre každé jedno mesto. V takom prípade by bolo výhodné pridať pri vytvorení `table` do `PRIMARY_KEY` na prvé miesto `n-tice` položku obsahujúcu mesto a nastavenie replikačného faktoru na vyššiu hodnotu pre uchovanie viacero kópií (ak by došlo k zlyhaniu jedného alebo viacerých uzlov).

3.4 Zmeny dát v čase, aktualizácie

Dáta pribúdajú do dátovej sady pravidelne podľa najnižšej periódy zbierania metrík (1h). Nové položky je ľahké identifikovať podľa usporiadateľnej položky `actualized`. Pridávanie nových dát je efektívnejšie pomocou využitia `prepared statement`, ktorý je imlementovaný aj v priloženom skripte na naplnenie databázy. Cassandra DB navyše podporuje TTL pre každý `row`, takže je možné nastaviť životnosť záznamov na stanovený čas.

Pridanie dát do Cassandra DB by mohlo prebehnúť priamo z terminálu nástroja `cqlsh` jazykom CQL alebo pomocou iného programovacieho jazyka s kompatibilným driverom (napr. Python). Čítanie a dotazovanie je možné pomocou ad-hoc dotazov (aj keď je to považované za anti-pattern), no najrýchlejším spôsobom na opakované dotazovanie je vytvorenie `prepared statement`, ktorý umožňuje preskočenie fázy parsovania pri spracovaní dotazu[?].

Ako bolo spomenuté, dáta su podľa dokumentácie samotného zdroja neverifikované, teda sa nekontroluje ich správnosť. Aj keď motivácia zavádzať do primárneho kľúča unikátny identifikátor je malá, rozhodli sme sa do `PRIMARY_KEY` zahrnúť aj `objectId`, práve kvôli zachovaniu možnosti odstrániť alebo upraviť konkrétne `rows` (aj také, kde by bola zhoda na troch primárnych `columns`—`name`, `code`, `actualized`).

3.5 Dotaz na Cassandra DB

```
SELECT code, actualized, no2_1h, pm10_1h, pm2_5_1h
FROM upa.measurements
WHERE name='Brno-Svatoplukova' AND code = 'BBMSA'
ORDER BY actualized DESC LIMIT 15;
```

1. Parsovanie dotazu - Pred spracovaním prebehne kontrola syntaxe dotazu a overenie platnosti v kontexte jazyka CQL.
2. Partition Key Lookup - Uvedený SELECT dotaz špecifikuje `name='Brno-Svatoplukova'` a `code = 'BBMSA'`. Toto sú **partition keys**, z ktorých sa vygeneruje token, pomocou ktorého sa vyhľadáva uzol s hľadanými dátami. Každý uzol obsahuje hodnoty **tokens** také, aby bola každá možná hash hodnota namapovaná na práve jeden uzol.
3. **Clustering columns** - Cassandra DB vyhľadáva dotazované dáta na uzle na základe (v našom prípade je to hodnota položky **actualized**, podľa ktorej sú dáta zoradené).
4. Filtrovanie - V dotaze špecifikujeme stĺpce, ktoré chceme dostať vo výsledku.
5. Zoradenie - `ORDER BY actualized DESC` určí zoradenie v zostupnom poradí.
6. Obmedzenie - `LIMIT 15` obmedzí rozsah výsledku na prvých 15 položiek
7. Vrátenie výsledku

4 Grafová databáza – Neo4j

Na ukázanie fungovania grafovej databázy bola určená databáza Neo4j⁹. Na rozdiel od tradičných relačných databáz, Neo4j používa namiesto relačného modelu model grafový, teda s uzlami, vzťahmi medzi nimi, a s vlastnosťami daných uzlov a vzťahov. Neo4j sa vyznačuje ukladaním zoznamu susedov. To znamená, že každý uzol udržiava zoznam svojich susedných uzlov, vďaka čomu sú operácie prechodu (pri všetkých príkazoch, ktoré potrebujú pracovať so vzťahmi medzi uzlami) veľmi rýchle. Tento model umožňuje prístup ku vzťahom uzla (k susedom) v konštantnom čase, bez ohľadu na celkovú veľkosť grafu. Vzťahy sa v Neo4j ukladajú spôsobom, ktorý môže pripomínať spájaný obojsmerný zoznam s ukazateľmi na začiatkové a koncové uzly. Každý záznam vzťahu má pevnú veľkosť a obsahuje ukazovatele na jeho vlastnosti, začiatkové a koncové uzly a predchádzajúce a nasledujúce vzťahy pre oba uzly (v prípade, že dva uzly spája niekoľko vzťahov, nie len jeden).

4.1 Zvolený dataset

Ako dataset, ktorý ukáže schopnosti a možnosti Neo4j, bol zvolený dataset **Finanční toky neziskovému sektoru v ČR**, a to vo formáte CSV. Tento dataset obsahuje informácie o rôznych organizáciách v Českej republike, ktoré pobrali príspevky od štátu v rôznych hodnotách. Obsahuje množstvo dát, ako napríklad názov a geografickú polohu danej organizácie, čiastku, ktorú táto organizácia dostala, resp. koľko peňazí jej bolo uvoľnených a koľko reálne prečerpala. Ďalej sa tu objavuje rok založenia organizácie, status (aktívny alebo neaktívny) tejto organizácie, a podobne. Tento dataset je zaujímavý práve preto, pretože jeho štruktúra postavená na vzťahoch medzi organizáciami dostávajúcimi príspevky a štátnymi inštitúciami, ktoré im tento príspevok dávajú, je ako stvorená na zaznamenanie v grafe. Tieto vzťahy sú kľúčové pre pochopenie toku peňazí a Neo4j, ako grafová databáza, je ideálna na ich efektívne zobrazenie a dotazovanie. Vybratý dataset sa aktualizuje každý rok.

Iné NoSQL databázy neboli vhodné pre tento typ datasetu. InfluxDB nie je vhodná databáza pre tento dataset, a hlavným dôvodom je to, že tento dataset sa zaoberá tokom peňazí medzi rôznymi subjektami, a aj keď sa tu nachádza časový údaj, nie je veľmi vhodné ho zobrazovať ako postupnosť časových údajov, keďže na seba nenadväzujú.

⁹<https://neo4j.com/docs/>

Cassandra, napriek tomu, že je výborná v škálovateľnosti (čo sa môže hodiť pri pridávaní rôznych vzťahov medzi subjektami), nie je navrhnutá na prirodzené spracovanie, resp. dotazovanie komplexných vzťahov. Wide-column model Cassandry nie je vhodný pre datasety, kde je hlavným zámerom zobrazenie vzťahov medzi subjektmi. Normalizácia by mohla viesť k potenciálnej duplikácii dát, čo nie je efektívne pre väčšie datasety.

Pokiaľ ide o MongoDB, hoci ponúka veľmi veľkú flexibilitu v oblasti schémy, zobrazenie zložitých vzťahov môže byť zložitý. Na získanie súvisiacich údajov by mohli byť potrebné viaceré dotazy alebo spojenia, čo môže byť neefektívne pre veľké datasety.

Na druhej strane, Neo4j, ako grafová databáza, je prirodzene navrhnutá na tvorenie, reprezentáciu a dotazovanie vzťahov. Uzly a hrany v Neo4j môžu mať atribúty, čo robí tento systém prirodzeným pre datasety s komplexnými vzájomnými prepojeniami. Neo4j poskytuje efektívne dotazovanie vzťahov bez potreby zložitých spojení, ktoré môžu byť v iných databázach náročné na výkon.

4.2 Spracovanie datasetu a vytvorenie databázy

Vybraný dataset obsahuje veľké množstvo stĺpcov, no len niektoré z nich boli vybraté. Ide o názov organizácie, mesto, okres, a kraj jej pôsobenia, čiastka uvoľnená a čiastka čerpaná, a nakoniec sa vybral stĺpec so štátnou inštitúciou, ktorá financie zverila danej organizácii. V datasete sa nachádzalo veľa nevybraných stĺpcov, ako napríklad geografická poloha danej organizácie, dátum odovzdania peňazí, oblasť operačného programu a podobne, no veľmi často sa niektoré riadky pre dané organizácie vôbec nevypĺňali, a preto boli vyhodnené a nepoužité. V datasete sa ale nachádzali prípady, v ktorých niektoré organizácie dostali peniaze od nešpecifikovaných inštitúcií. Túto skutočnosť sme v databáze zachovali, lebo v reálnom svete by takéto situácie mohli byť prešetrované (a dáta doplnené).

Na databázu bolo vytvorené spojenie cez Python skript, v ktorom sa spracoval pôvodný dataset cez knižnicu **Pandas**, odstránili sa nadbytočné stĺpce (a riadky, kde chýbali potrebné informácie), a cez jazyk Cypher boli odoslané dáta do databázy. Zdrojový kód je možné nájsť v súbore `neo4j_proj.py`. Priložený skript používa príkazy **MERGE**, ktoré umožňujú pridávať nové uzly do databázy bez toho, aby vznikli duplicity. Skript síce pri každom spustení maže celú databázu a nanovo ju vytvára, no to kvôli testovacím účelom – nie je nutné ju mazať pred každým pridaním. Retencia dát je daná nastavením

databázy, a je možné ju meniť. Jazykom Cypher je možné spätne meniť už existujúce dáta, keby to bolo nutné, a to použitím príkazov **SET** alebo **MERGE**. Ich funkcia je podobná, no **MERGE** vytvorí dané uzly a hrany, ak je to potrebné, pričom **SET** funguje len na existujúce dáta. Neo4j umožňuje distribuovať a škálovať dáta pre rýchlejšie získavanie dát, napríklad vo verzii Neo4j Enterprise Edition. Neo4j nerozlišuje medzi ad-hoc dotazmi a preddefinovanými dotazmi.

Bolo potrebné riešiť niekoľko problémov. Prvým bolo to, že pridávanie informácií do databázy po jednom riadku bolo veľmi pomalé, pretože transakčná réžia zaberala mnoho času. Pridať po jednom všetkých 90000 riadkov trvalo veľmi dlhú dobu. Riešením bolo optimalizovať pridávanie do tabuľky. Prvou optimalizáciou bolo použitie niekoľkých príkazov v jednom dotaze. Na začiatku sa pre jeden riadok v pôvodnom CSV súbore pridávala každá entita v jednom príkaze (každý uzol predstavujúci organizáciu, štátnu inštitúciu, ich spojenia, kraje, okresy, a mestá) - toto všetko vznikalo v jednom riadku pôvodného datasetu, a toto všetko sa vykonávalo po jednom, čo malo za dôsledok extrémne pomalé spracovávanie. Po tejto optimalizácii znamenal jeden riadok v CSV jeden dotaz v Cypher. No ani toto nestačilo.

Druhou, najväčšou optimalizáciou, sa podarilo čas spracovania celého datasetu skrátiť na približne 3 sekundy. Motiváciou bolo dať ešte viac riadkov do jedného dotazu. Optimalizácia spočívala v použití príkazu **UNWIND** jazyka *Cypher*. Jeho princíp je jednoduchý. Dopredu bol vybratý počet riadkov, ktoré chceme spracovať naraz, napríklad 5000. Z týchto 5000 riadkov sa vytvoril zoznam slovníkov, ktoré boli odovzdané funkcii **UNWIND**. Ako jej názov napovedá, táto funkcia rozbalila tieto dáta a aplikovala všetky príkazy pod ňou na každý riadok dát, ktoré dostala.

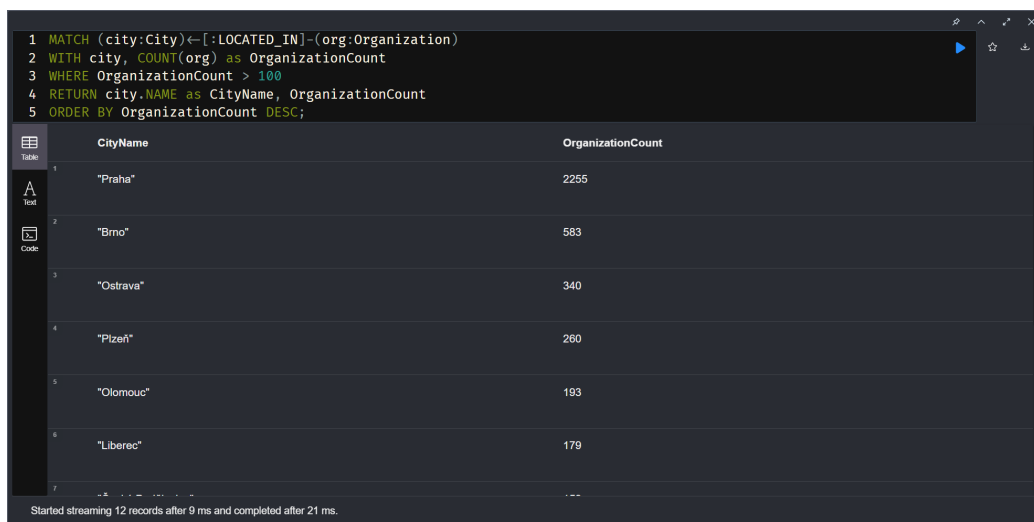
4.3 Vnútorne fungovanie dotazovania

Keď Neo4j prijme dotaz, najprv ho spracuje. To urobí tak, že rozdelí vstupný dotaz na tokeny, z ktorých potom zostaví abstraktný syntaktický strom. Ten sa potom optimalizuje (novšie aktualizácie Neo4j sa venujú najmä tejto optimalizácii). Potom si vytvorí vnútorný graf tohto dotazu, z ktorého vytvorí tzv. logický plán, ktorý je tvorený stromom rôznych operátorov, ktoré reprezentujú operácie, ktoré je potrebné vykonať, aby sa dotaz vykonal. Tu nastáva ďalšia optimalizácia, rozbaľovanie vnorených operácií a podobne. Tento logický plán je potom zmenený na tzv. execution plan, ktorý už predstavuje konkrétne kroky, ktoré bude Neo4j vykonávať. Tento plán (akási obdoba

skompilovaného programu, t.j. strojového kódu), je cachovaný a je možné ho použiť znova (a teda veľmi rýchlo). Z tohto plánu potom vykoná Neo4j dotaz a vráti výsledky. Toto celé je možné urýchliť vytvorením indexov, a to v tvare `CREATE INDEX FOR (var:Type) ON var.property`.

Dátové úložisko Neo4j je v podstate séria spájaných záznamov s pevnou veľkosťou. Vlastnosti sú uložené ako spájaný zoznam záznamov vlastností, pričom každý záznam obsahuje kľúč, hodnotu a ukazuje na ďalšiu vlastnosť (atribút). Uzly odkazujú na svoj prvý záznam atribútov a prvý vzťah vo svojom zozname vzťahov. Každý vzťah odkazuje na svoj počiatočný a koncový uzol a na predchádzajúci a nasledujúci záznam vzťahu pre oba uzly. Tieto rôzne veci sú uložené v rôznych súboroch, konkrétne napríklad `neostore.nodestore.db` pre uzly, `neostore.relationshipstore.db` pre vzťahy, a potom ďalšie takéto úložisko pre atribúty uzlov a vzťahov, prípadne ďalšie úložiská pre väčšie atribúty (reťazce alebo zoznamy dielčích typov). Kompresia dát nie je v Neo4j podporovaná.

4.4 Vybrané dotazy nad danou databázou



The screenshot shows a Cypher query interface with a query editor at the top and a results table below. The query is as follows:

```
1 MATCH (city:City)-[:LOCATED_IN]-(org:Organization)
2 WITH city, COUNT(org) as OrganizationCount
3 WHERE OrganizationCount > 100
4 RETURN city.NAME as CityName, OrganizationCount
5 ORDER BY OrganizationCount DESC;
```

The results table displays the following data:

	CityName	OrganizationCount
1	"Praha"	2255
2	"Brno"	583
3	"Ostrava"	340
4	"Plzeň"	260
5	"Olomouc"	193
6	"Liberec"	179
7

At the bottom of the interface, a status message reads: "Started streaming 12 records after 9 ms and completed after 21 ms."

Obr. 5: Dotaz ukazuje, koľko organizácií v každom kraji poberá príspevky.

```

1 MATCH (gov:Government)←[r:RECEIVED_FROM]-(org:Organization)
2 RETURN gov.NAME as GovernmentEntity, SUM(r.allocated) as TotalAllocatedFunds
3 ORDER BY TotalAllocatedFunds DESC
4 LIMIT 5;

```

	GovernmentEntity	TotalAllocatedFunds
1	"Ministerstvo školství, mládeže a tělovýchovy ČR"	22837393396.52999
2	"není k dispozici"	17302650512.850147
3	"Ministerstvo pro místní rozvoj"	4395588288.029996
4	"Ministerstvo práce a sociálních věcí ČR"	4270966436.830057
5	"Evropský sociální fond"	3900295581.5499463

Started streaming 5 records after 13 ms and completed after 131 ms.

Obr. 6: Dotaz ukazuje, ktoré štátne inštitúcie v ČR dali najviac peňazí organizáciám.

```

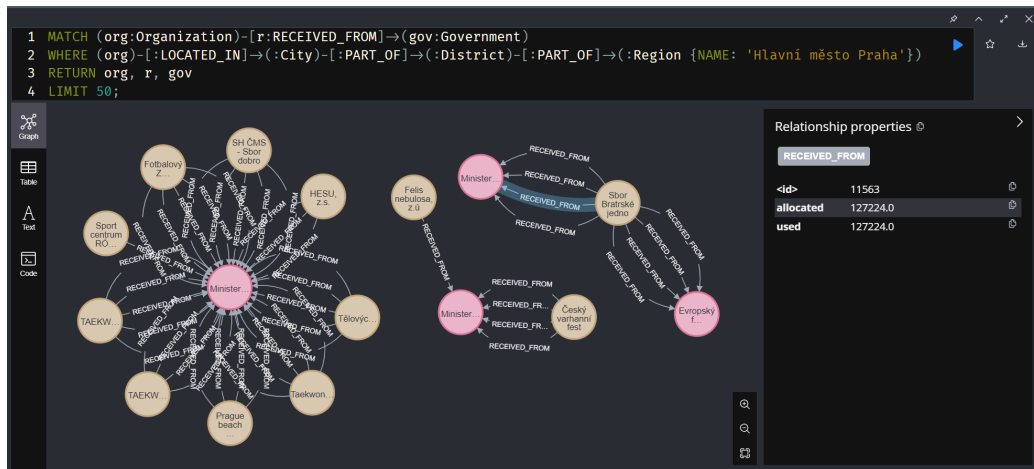
1 MATCH (org:Organization)←[r:RECEIVED_FROM]→(gov:Government)
2 WHERE (org)-[:LOCATED_IN]→(:City)-[:PART_OF]→(:District)-[:PART_OF]→(:Region {NAME: 'Hlavní město Praha'})
3 WITH org, SUM(r.allocated) as TotalAllocated
4 RETURN org.NAME as OrganizationName, TotalAllocated
5 ORDER BY TotalAllocated DESC;

```

	OrganizationName	TotalAllocated
1	"Fotbalová asociace České republiky"	2148270718.0
2	"Český svaz ledního hokeje z.s."	836280153.0
3	"Svaz měst a obcí České republiky"	748294225.64
4	"Český atletický svaz"	717303122.0
5	"Český olympijský výbor"	647506237.0
6	"Český tenisový svaz z.s."	598384869.0
7

Started streaming 2255 records after 16 ms and completed after 173 ms, displaying first 1000 rows.

Obr. 7: Tento dotaz ukazuje, ktoré organizácie v Pražskom kraji dostali najviac peňazí.



Obr. 8: Jeden z dotazov, ktoré umožňujú peknú vizualizáciu. Ukazuje štátne inštitúcie (v ružovom), a organizácie, ktoré od nich pobrali príspevky. Zaujímavosťou je všimnúť si, že v prípade, že organizácia dostala niekoľko dotácií od tej istej entity, vzniknú štyri vzťahy.

5 Databáza časových radov – InfluxDB

Na prezentáciu funkčnosti databázy časových radov bola vybraná databáza InfluxDB¹⁰, a to konkrétne verzia V2. Na rozdiel od tradičných relačných databáz, InfluxDB je databáza optimalizovaná pre rýchle zapisovanie a dotazovanie časovo orientovaných dát. Umožňuje efektívne ukladanie a spracovanie dát, ktoré sú pravidelne aktualizované a monitorované v čase. InfluxDB je špeciálne navrhnutý tak, aby zvládal veľké množstvo zápisov a dopytov za sekundu, čo je ideálne pre rôzne merania a monitorovania v reálnom čase. Dáta v InfluxDB sú organizované do skupín, čo zabezpečuje, že dotazy sú mimoriadne rýchle. InfluxDB totiž automaticky indexuje dáta podľa času, čo znižuje komplexitu a správu pri práci s týmito dátami. InfluxDB rozlišuje medzi tzv. **tag** a **field**. Kým **tag** je hodnota, ktorá pomáha odlišovať napríklad niekoľko rôznych meraní vrámci jedného času, **field** je nameraná hodnota. Ako príklad si môžeme uviesť dve meracie stanice teploty, jednu v Brne, druhú v Bratislave. Ak by merali v rovnaký časový okamih, ich **tag** by boli Brno, resp. Bratislava, a v ich **field** by boli ich merania. Indexovať sa dá aj pomocou týchto tagov, čo dokáže zrýchľovať dotazovanie nad určitou podmnožinou meraní. Tagov môže byť, samozrejme, viac.

5.1 Zvolený dataset

Na základe dostupných dátových sád pre mesto Brno musel byť vybraný dataset, ktorý je citlivý na čas a ktorý sa hodí na analýzu časových radov. Zvolený bol dataset **Dopravní nehody**. V priebehu niekoľkých rokov tento dataset zaznamenával pre každý deň nehody vozidiel, ktoré sa v Brne a blízkom okolí diali. Každý riadok predstavuje jedného ľudského účastníka vrámci nehody, ktorá je daná unikátnym identifikačným číslom. Všetky riadky s účastníkmi pre jednu nehodu obsahujú tie isté informácie, ktoré má nehoda spoločné pre všetky autá, čiže viditeľnosť, počasie, čas, počty zranených, mŕtvych, škoda na vozidlách, a podobne. Riadky sa odlišovali v minimálnom počte informácií, najmä išlo o úlohu účastníka (vodič, spolusediaci, atď). Údaje sa aktualizujú každý rok. Ich granularita, resp rozlíšenie, je na minúty.

InfluxDB bola zvolená pre tento dataset z niekoľkých dôvodov. Prvou je, prirodzene, to, že bola navrhnutá špeciálne pre časové rady. To znamená, že sme potrebovali databázu, ktorá je optimalizovaná na ukladanie, dota-

¹⁰<https://docs.influxdata.com/influxdb/v2/>

zovanie a analýzu časovo orientovaných dát. InfluxDB má zabudované silné agregáčne schopnosti vhodné pre časové rady. Toto umožňuje rýchle a efektívne vykonávanie dotazov, ako sú sumy, priemery alebo percentilové hodnoty pre konkrétne časové intervaly (napr. mesačné alebo ročné sumy nehôd, škôd, a podobne). Zaujímavosťou je to, že InfluxDB podporuje okrem retencie aj funkciu downsamplingu. To znamená, že staré dáta môžu byť automaticky odstránené (pri obmedzenej retencii) alebo zredukované na menej detailné verzie, čo šetrí úložný priestor a zvyšuje výkon dotazovania (v prípade, že na nehody z veľmi dávnej doby sa už nepotrebuje dotazovať).

Iné NoSQL databázy neboli veľmi vhodné. Aj keď je MongoDB výkonná dokumentová databáza vhodná na ukladanie rôznych dátových štruktúr, nie je špecificky optimalizovaná pre časové rady. Agregáčne a dotazovacie operácie pre časové rady by mohli byť v MongoDB menej efektívne ako v InfluxDB. Neo4j je grafová databáza, ktorá je ideálna pre dáta s komplexnými vzťahmi a spojeniami. Tento dataset o nehodách je založený skôr na časových dátových bodoch než na vzťahoch medzi nejakými entitami. A aj keď Cassandra je optimalizovaná pre veľké objemy dát, jej dotazovací jazyk a štruktúra môžu byť menej intuitívne pre časové rady v porovnaní s InfluxDB. InfluxDB je tiež navrhnutá s lepšou podporou pre časové funkcie a agregáčne dotazy špecifické pre časové rady. Navyše, umožňuje aj distribuované použitie.

5.2 Spracovanie datasetu a vytvorenie databázy

Dataset obsahuje veľa informácií, no pre potreby tohto projektu boli vybraté len niektoré. Ide konkrétne o stĺpce označujúce počet ľahko zranených, ťažko zranených, mŕtvych, a celková škoda na autách. Tieto dáta bolo ale potrebné predprípraviť. Kontrolou bolo zistené, že stĺpce, ktoré boli vybraté, majú rovnaké hodnoty pre všetkých účastníkov (riadky) jednej nehody. Tým pádom bol v Pythone napísaný skript, ktorý tieto dáta odfiltroval, a to tak, že nechal z duplicitných riadkov len ten, ktorý bol ako prvý. Ďalej boli všetky nehody pre jeden deň zlúčené do jedného riadku, kde boli ich údaje sčítané. Tieto dáta boli neskôr poslané do databázy, no pri prvotnom vytváraní bolo najprv potrebné vytvoriť tzv. Bucket, do ktorého sa tieto dáta posielali. Toto zaisťuje niekoľko knižníc. Zdrojový kód sa nachádza v skripte `InfluxDB_script.py`. Retencia dát je zaistená pri vytváraní daného Bucketu, a to tak, že sa nastaví tzv. `retention_policy`. Ak ostane prázdna, dáta sú udržiavané trvalo. Dáta v InfluxDB sú nemenné – namiesto toho sa pri nutnosti existujúce dáta zmeniť jednoducho dané dáta prepíšu. Dátový bod je totiž určený časom a

jeho tagmi. Ak sa tagy nikde nenachádzajú, stačí vytvoriť nový dátový bod s dátumom, ktorý chceme prepísať, a tento sa prepíše.

5.3 Vnútorne fungovanie dotazovania a uloženie dát

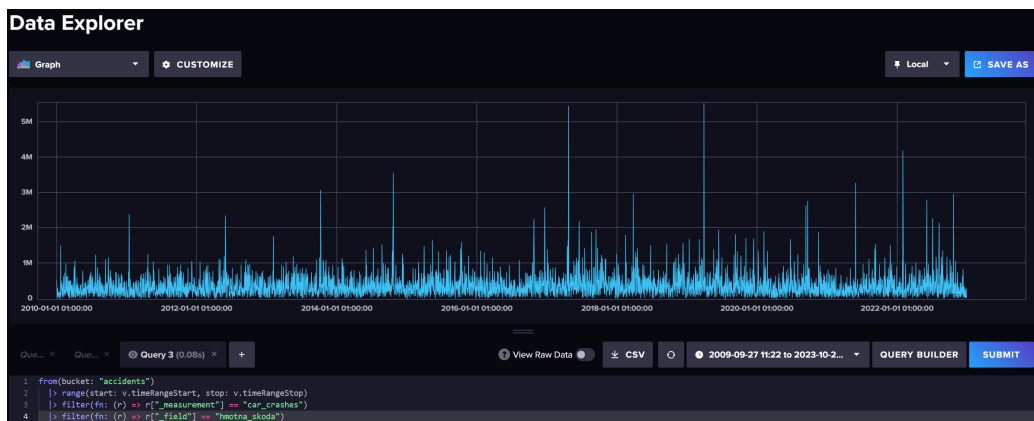
InfluxDB využíva *Write Ahead Log* na okamžité zabezpečenie prichádzajúcich dát, čo zaručuje odolnosť voči neočakávaným výpadkom počas zápisu. Paralelne s WAL je v pamäti uložená cache, ktorá umožňuje okamžité dotazovanie dát. TSM (Time-Structured Merge Tree) efektívne komprimuje a ukladá dáta podľa času. Aby InfluxDB rýchlo a efektívne reagoval na dotazy, aj keď rastie množstvo dát, používa *Time Series Index* (TSI). TSI optimalizuje vyhľadávanie a dotazovanie v databáze tým, že skupina kľúčov sérií je organizovaná podľa merania, značky a poľa.

5.4 Vybrané dotazy nad danou databázou

Narozdiel od Influx V1, ktorého jazyk pripomínal bežný SQL syntax, jazyk druhej verzie, Flux, je funkcionálny, a veľmi sa odlišuje od prvej verzie. Tento jazyk bol použitý v nasledujúcich dotazoch, ktoré sú vizualizované vo vstavanom webovom rozhraní pre Influx.



Obr. 9: Dotaz zobrazuje pre každý deň počty zranených, ťažko zranených, a mŕtvych, a to v rozmedzí určenom vo webovom rozhraní.



Obr. 10: Dotaz zobrazuje pre každý deň celkovú škodu.



Obr. 11: Tento agregáčny dotaz ukazuje počty mŕtvych za celý rok v danom rozmedzí. Tieto dáta sú zobrazené v grafe, no je možné ich dostať aj v tabuľke (toto je napríklad vidieť pri použití CLI rozhrania, tzv. InfluxCLI).

Literatúra

- [1] Statutární město Brno - Magistrát města Brna - Otevřená data, “Kvalita ovzduší / air quality,” May 2021. [Online]. Available: <https://data.brno.cz/datasets/mestobrna::kvalita-ovzdu%C5%A1%C3%AD-air-quality/about>