FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

UPA – 1. projekt

Ukládání rozsáhlých dat v NoSQL databázích

Filip Jahn (xjahnf00)
Michal Luner (xluner01)
Vojtěch Staněk (xstane45)

11. října 2023

Obsah

1 Úvod

Cílem projektu je analýza datových sad poskytovaných Národním katalogem otevřených dat, jejímž výstupem je návrh optimálního způsobu uložení těchto dat do vhodných NoSQL databází. Pro každý ze 4 typů NoSQL databází (sloupcové, dokumentové, grafové a databáze časových řad) byla vybrána konkrétní datová sada, která se využije k ukázce práce a vhodnosti výběru databáze.

Při navrhování uložení dat pracujeme s různými scénáři, jak jsou data vytvářena a jakým způsobem bude možné se na ně dotazovat. V potaz je také nutné vzít rozměr a množství dat, jak často se data mění, jak často dochází k jak velkým aktualizacím již stávajících dat apod.

Dokumentace obsahuje 4 kapitoly popisující jednotlivé druhy NoSQL databází. Každý druh má danou strukturu okruhů, na které je vhodné (i dle zadání projektu) se při analýze datové sady i databáze samotné zaměřit.

Všechny databáze běžely na virtuálním stroji staženém ze stránek cvičení[1].

[1] `https://rychly-edu.gitlab.io/dbs/nosql/nixos-dbs-vm/`

2 NoSQL databáze

2.1 Sloupcová NoSQL databáze

- **Název datové sady:** Kvalita ovzduší
- **Distribuce datové sady:** CSV formát (modifikován o smazání a prohození vybraných sloupců)
- **Zvolený zástupce NoSQL databáze:** Apache Cassandra
- **Zdůvodnění využití dané sady a databáze:**

  Databázový systém Cassandra je navržený pro ukládání a správu velkého množství dat distribuovaných na serverech a jeho potenciál se nejlépe využije při velice rychlém čtení a zápisu. Jedná se o NoSQL databázi „klíč-hodnota“, kde klíč je dále dělen na tzv. „partition“ a „clustering“. Partition klíč určuje, na kterém uzlu budou data uložena[2], clustering klíč pak data uspořádává již v samotném uzlu[3]. Partition klíč je efektivní na rovnost, jelikož se jedná o hashovaný index. Clustering klíč využívá B+strom pro indexaci (sežazený index), což umožňuje větší volnost užití operátorů při vyhledávání (větší, menší, nerovná se…).

  Cassandra se zároveň snaží data v tabulce komprimovat. Využívá toho, že klíč vede na nějaký uzel, kde je celý řádek uložen, a ten je schopna (resp. jeho hodnotu) zkomprimovat. Dále podporuje i zapomínání dat (Time-to-Live), což se hodí v případě, kdy ukládáme data z logů či senzorů. Obecně platí, že provádět mazání v distribuovaném clusteru není triviální záležitost (trvá, než se daná změna zpropaguje), typicky se tedy data pouze vkládají (příp. málokdy se udělá aktualizace, většinou je to UPSERT). Výhodou je v neposlední řadě i dotazovací jazyk CQL, který se velmi podobá známému a široce adoptovanému SQL.

  Dataset kvality ovzduší[4] sestává z naměřených hodnot senzorů z konkrétních míst Brně. Výše popsané vlastnosti sloupcové databáze Cassandra se dají aplikovat i na náš dataset. Data byla rozvržena tak, aby se dle nich dalo snadno vyhledávat. Konkrétně byly jako klíče zvoleny lokace a čas, což zabezpečuje datovou prostorovou lokalitu. Jelikož jsou data ze senzorů, očekáváme, že budou přibývat často a starší záznamy se nebudou aktualizovat, nýbrž spíše zapomínat (nastavení TTL). V případě, kdy chceme vyhledávat na základě neklíčových atributů, je třeba vytvořit index na daný sloupec. Neučiníme-li tak, dotaz skončí chybou. Dále očekáváme, že data se v průběhu měření nebudou lišit drasticky, což databázi může umožňovat komprimaci.

  Dataset by se dal využít i v databázi časových řad (např. InfluxDB). Ač je Cassandra obecně flexibilnější a univerzálnější, výhoda InfluxDB tkví především v práci s časovou složkou, kdy můžeme velmi snadno provádět dotazy na vymezené časové období a jednoduše shlukovat data.

  Dále by se mohla vytvořit další tabulka, která by shromažďovala agregovaná data, která jsou starší a nevyžadují tedy takovou přesnost hodnot (např. stačí průměry za určitou uplynulou dobu).

- **Příkazy pro definici úložiště:**

  Jednotlivé části databáze jsou rozděleny na tzv. „prostory klíčů“. Nejdříve se tedy musí vytvořit KEYSPACE, který si zároveň necháme vypsat a nastavíme jeho použití. Replikační strategie určuje, jakým způsobem mají být rozděleny repliky (kopie) dat na NoSQL clusteru. Jelikož máme pouze 1 virtuální stroj, kde Cassandra běžela, nemá tato replikační strategie tak velký význam.

[2] Řádky, které mají stejnou hodnotu partition klíče, jsou uloženy na stejném uzlu v clusteru.

[3] Více informací v dokumentaci: `https://www.baeldung.com/cassandra-keys`

[4] `https://data.gov.cz/zdroj/datov-sady/44992785/4907e5c199620d99a44c1ec388703cd5`

```
CREATE KEYSPACE IF NOT EXISTS project
WITH replication = {'class': 'SimpleStrategy', 'replication_factor' : 3};
DESCRIBE KEYSPACE project;
USE project;
```

Pomocí uvedeného příkazu se vytvoří tabulka pro údaje o znečištění vzduchu. Navzdory tomu, že se jedná o NoSQL databázi, u Cassandry schéma definujeme.

Z dotazu vidíme, že jako partition klíč se využívá `name` a jako clustering klíč `actualized`. V našem případě se tedy data mezi uzly rozmístí dle místa měření, v rámci uzlu se pak rozdělí v závislosti na časové složce.

```
CREATE TABLE IF NOT EXISTS air_pollution (
  name text,
  so2_1h double,
  no2_1h double,
  co_8h double,
  pm10_1h double,
  o3_1h double,
  pm10_24h double,
  pm2_5_1h double,
  actualized timestamp,
  PRIMARY KEY (
    (name),
    actualized
  )
) WITH comment = 'Air pollution data as measured by stations.';

DESCRIBE air_pollution;
```

- **Algoritmický popis importu dat:**

  Pro naplnění databáze daty slouží následující příkaz.

```
COPY air_pollution (
    name,
    so2_1h,
    no2_1h,
    co_8h,
    pm10_1h,
    o3_1h,
    pm10_24h,
    pm2_5_1h,
    actualized
    )
FROM 'air_pollution_data.csv'
WITH HEADER = true;
```

  Data v `air_pollution_data.csv` jsou výstupem skriptu `CSVColumnReorderingScript.py`, který vstupní dataset pro přehlednost očistí o zbytečné sloupce a převede časovou značku do jiného formátu.

Pokud bychom pro vložení chtěli využít příkazu INSERT, lze to provést příkazem níže. Obecně platí, že pokud bychom takových vložení dělali více a prováděli je odděleně, Cassandra po každém vložení začne vytvářet repliky a propagovat změny přes celý cluster. Proto se hodí skupinu vložení uložit do dávky za použití příkazu `BATCH`.

```
INSERT INTO air_pollution (
    name,
    so2_1h,
    no2_1h,
    co_8h,
    pm10_1h,
    o3_1h,
    pm10_24h,
    pm2_5_1h,
    actualized
) VALUES ('Brno-Arboretum', 0, 37.9, 6969, 35.8, 16.2, 28.9, 19.3,
          '2022-08-25 07:29:35.000000');
```

Pro ověření, že se data aktualizovala, použijeme následující dotaz.

```
SELECT * FROM air_pollution
WHERE name = 'Brno-Arboretum' and actualized = '2022-08-25 07:29:35.000000';
```

- Příkaz `UPSERT`

  Pro vyzkoušení UPSERT se vytvořil nový soubor `air_pollution_data_2.csv`, který obsahoval záznam již v databázi existující, další již existující záznam s upravenou hodnotou atributu, a nakonec ještě jeden úplně nový, v databázi nepřítomný záznam.

  Nejdříve se pomocí SELECT dotazu[5] analyzovaly aktuální hodnoty, poté se nová data do databáze nahrála a pomocí stejných dotazů se ověřilo, že vložení bylo úspěšné.

- **Dotazy v jazyce databáze:**

  Pro výpis všech dat na základě partition klíče slouží následující dotaz. Vyhledání je rychlé, jelikož hledáme podle indexovaného klíče. Zároveň platí, že partition klíč může bý složen z více složek – v takovém případě je třeba se v dotazech ptát na všechny složky.

```
SELECT * FROM air_pollution WHERE name = 'Brno-Zvonařka';
```

name	actualized	co_8h	no2_1h	o3_1h	pm10_1h	pm10_24h	pm2_5_1h	so2_1h
Brno-Zvonařka	2022-08-25 07:29:35.000000+0000	null	30	null	43.7	44.5	29.1	null
Brno-Zvonařka	2022-08-25 08:29:35.000000+0000	null	31	null	78	46.8	28.6	null
Brno-Zvonařka	2022-08-25 10:29:35.000000+0000	null	41.1	null	68	50.1	26	null
Brno-Zvonařka	2022-08-25 11:29:35.000000+0000	null	34.4	null	56.8	50.5	25.7	null
Brno-Zvonařka	2022-08-25 12:29:35.000000+0000	null	37.3	null	69	51.8	24.7	null
Brno-Zvonařka	2022-08-25 13:29:35.000000+0000	null	42.5	null	64	51.7	22.1	null
Brno-Zvonařka	2022-08-25 14:29:35.000000+0000	null	44.2	null	41.4	51.8	16.3	null
Brno-Zvonařka	2022-08-25 15:29:35.000000+0000	null	43.8	null	47.4	52.3	16.8	null
Brno-Zvonařka	2022-08-25 16:29:35.000000+0000	null	49.9	null	48.8	52.9	17.3	null

Obrázek 1: Část výstupu na dotaz znečištění vzduchu v části Brno-Zvonařka.

[5] Ukázky takových dotazů jsou k dispozici v následujícím bodu.

Ze snímku obrazovky 1 lze dle barev vyčíst, který atribut je partition klíč, clustering klíč a které hodnoty jsou neindexované a neslouží k vyhledávání. Abychom byli schopni vyhledat i dle neklíčových položek, musíme vytvořit index. Kdybychom to neudělali, dotaz skončí chybovou hláškou.

Chceme-li například vyhledat datum a čas nejhoršího znečištění ovzduší v určité části města, nejprve vytvoříme index. Za jeho použití tak víme, na jakém uzlu (dle indexovaného klíče, v našem případě `co_8h`) jsou data uložená.

```
CREATE INDEX IF NOT EXISTS worst_pollution_arboretum
ON air_pollution (co_8h);
```

a následně se dotážeme

```
SELECT MAX(co_8h)
FROM air_pollution
WHERE name = 'Brno-Arboretum';
```

Dotaz by šlo bez chybové hlášky i bez zavedení indexu provést s přepínačem `ALLOW FILTERING`. V takovém případě by databáze musela projít celý cluster, uzel po uzlu, tedy neoptimálně bez indexu[6].

Co má Cassandra oproti ostatním NoSQL databázím navíc, jsou materializované pohledy. Podobně jako v relačních databázích, materializované pohled je dotaz uložený v cache paměti. V těle pohledu je dotaz, jehož součástí je definice nových indexů. V uvedeném příkladě je hodnota `no2_1h` partition klíč, zbylé 2 položky pak clustering klíče.

```
CREATE MATERIALIZED VIEW temps_no2_pollution AS
    SELECT * FROM air_pollution
    WHERE no2_1h IS NOT NULL AND
          name IS NOT NULL AND
          actualized IS NOT NULL
    PRIMARY KEY (no2_1h, name, actualized)
    WITH comment = 'Hladina oxidu dusičitého po jedné hodině';
```

Výstupem dotazu jsou pak hodnoty oxidu dusičitého. Ze snímku 2 lze opět dle barev vidět, jak se změnily klíče.

```
SELECT * FROM temps_no2_pollution;
```

no2_1h	name	actualized	co_8h	o3_1h	pm10_1h	pm10_24h	pm2_5_1h	so2_1h
84	Brno-Svatoplukova	2022-11-14 08:34:36.000000+0000	728	null	49.4	43.1	37.7	null
63.1	Brno-Lány	2023-04-24 05:26:08.000000+0000	364	8.4	25	25.6	18.2	6.4
63.1	Brno-Lány	2023-05-11 22:29:06.000000+0000	204	1.2	23.3	16.4	12.5	5.1
63.1	Brno-Svatoplukova	2022-10-25 17:31:44.000000+0000	262	null	66.6	26.4	17.7	null
63.1	Brno-Svatoplukova	2022-12-29 09:37:53.000000+0000	582	null	41.9	34.4	35.3	null
63.1	Brno-Svatoplukova	2023-03-10 18:10:01.000000+0000	481	null	28.1	21.5	11.9	null
63.1	Brno-Svatoplukova	2023-03-31 08:06:15.000000+0000	553	null	29.1	25	23.7	null
63.1	Brno-Svatoplukova	2023-04-20 07:25:44.000000+0000	320	null	26.5	18.7	20.9	null

Obrázek 2: Část výstupu pro materializovaný pohled.

[6] V našem případě, kdy máme 1 uzel, je toto docela jedno, jelikož zde žádná distribuce dat není.

2.2 Dokumentová NoSQL databáze

- **Název datové sady:** Dopravní nehody
- **Distribuce datové sady:** GeoJSON formát (během předzpracování převeden na prostý JSON)
- **Zvolený zástupce NoSQL databáze:** MongoDB
- **Zdůvodnění využití dané sady a databáze:** Dokumentové databáze pracují s dokumenty ve formátech jako XML nebo JSON. Přidaná hodnota v používání dokumentových databází tkví v tom, že z dvojice klíč-hodnota je hodnota konkrétně strukturována a databází „chápána“. Jednou z výhod použitého zástupce dokumentových databází, MongoDB, je agregační pipeline, která umožňuje vytvářet komplikovanější dotazy. Z výchozí kolekce se tak postupně vytváří další „mezikolekce“, které se dále transformují na finální kolekci pro zisk výsledků. MongoDB tedy není pouze databáze, ale i výpočetní platforma. Při tvorbě databáze je třeba myslet na to, že nejefektivnější hledání se provede dle klíče (`_id`), který je hashovaný a zároveň určuje distribuci dat na clusteru. Vyhledání na základě hodnot, které nejsou klíčem, proběhne sice bez chybové hlášky, avšak není tak optimální.

  Dataset nehod[7] jsou dokumentová data a každý záznam obsahuje velké množství položek. Způsoby, jak tato data do databáze uložit, závisí především na záměru využití databáze klientskou aplikací. Celkově se pracovalo s několika scénáři rozvržení dat v závislosti na cíli.

  Předpokládalo se, že klientská aplikace bude sloužit především k častému zobrazování nehod v dané části Brna. Dále by se dalo očekávat, že nehody budou neperiodicky přibývat, neubývat a již existující záznamy musí být možno aktualizovat[8].

  Varianty, jak data uložit, jsou následující:

  1. 1 kolekce bez vnoření
     - V tomto případě jsou jednoduché dotazy pro zjištění obecných informací o nehodách, avšak agregovat nehody pro část města je složitější, jelikož musíme projít všechny záznamy a vybrat takové, u kterých odpovídá hledaný název.
     - Nabízelo by se tedy řešení uložit data dle části města a k tomu připojit všechny její nehody jako vnořenou strukturu (tj. 1 kolekce s vnořením).
  2. 1 kolekce s vnořením
     - Nalezení nehod v konkrétní části města by se oproti předchozímu případu značně zjednodušilo, avšak získání obecných informací o nehodách by se ztížilo, jelikož bychom museli procházet každý záznam, zanořit se do struktury a provést akci, která byla dotazem požadována (např. poměr nehod v různých ročních obdobích).
  3. 2 kolekce
     - Vytvoří se 2 kolekce – pro městské části a pro nehody, které se v nich staly. Vzájemné provázání se zajistí pomocí referencí v příslušných strukturách.
     - Tímto způsobem lze procházet jak nehody samotné, tak i snadno vybrat konkrétní městskou část a pracovat s jejími nehodami.
     - Toto řešení navíc lépe pracuje se scénářem, kdy by mětská část měla více nehod (což se dá předpokládat). Škálovatelnost se zde oproti předchozím řešením zlepšila. Podobnou myšlenku je možno najít i v manuálu MongoDB[9].

[7] `https://data.gov.cz/zdroj/datov-sady/44992785/f7604237598371dd478232df3ad93ce9`

[8] Např. pokud se zjistí nové závěry příčiny vzniku nehody, nové informace od svědků apod.

[9] URL: `https://www.mongodb.com/docs/manual/tutorial/model-embedded-one-to-many-relationships-between-documents/`. U produktů se očekává, že budou mít víc hodnocení, které není třeba vždy načítat (a jsou v oddělené kolekci).

  - Jelikož dochází k provázání pouze mezi 2 kolekcemi, využití DBRefs[10] není nutné.
  - Vyvstává pak otázka, jestli pole identifikátorů nehod u jednotlivých městských částí neroste příliš rychle a zdali vůbec přináší nějaké zrychlení[11]. Uvažujeme, že dotaz je natolik častý, že nám stojí za to pro něj mít takto strukturované kolekce a ušetříme opakované agregování dle městské části.

  S použitím vybrané databáze jde pomocí agregační pipeline vhodně přeuspořádávat data a využít je pro výpočet různých operací (součty, průměry atp.).

- **Příkazy pro definici úložiště:** Příkaz pro vytvoření databáze s názvem `test`, vytvoření kolekce `accidents` a její naplnění daty ze souboru `accidents.json`. Stejný postup se provedl pro vytvoření kolekce `locations`.

```
[demo@nixos:~]$ mongoimport --db test --collection accidents --file accidents.json --jsonArray
2023-10-05T18:37:32.755+0000    connected to: mongodb://localhost/
2023-10-05T18:37:35.759+0000    [######..................] test.accidents       28.9MB/109MB (26.5%)
2023-10-05T18:37:38.762+0000    [#############...........] test.accidents       60.0MB/109MB (55.0%)
2023-10-05T18:37:41.761+0000    [##################.....] test.accidents       88.5MB/109MB (81.0%)
2023-10-05T18:37:43.974+0000    [########################] test.accidents       109MB/109MB (100.0%)
2023-10-05T18:37:43.974+0000    69741 document(s) imported successfully. 0 document(s) failed to import.
```

Obrázek 3: Definice a naplnění MongoDB daty ze souboru.

- **Algoritmický popis importu dat:** Dataset lze stáhnout ve formátu GeoJSON, pro práci v MongoDB se skriptem `preprocess_accidents.py` očesaly některé atributy a provedly další drobné změny[12]. Skript produkuje celkově 2 JSON dokumenty: `accidents` a `locations`. První soubor obsahuje lehce upravená data z originální souboru, druhý soubor se skládá z jednotlivých městských částí, kde každá má své pole `accident_ids` s referencemi na nehody.

  Příkazy v předchozí odrážce se zajistilo úspěšné naplnění databáze těmito daty.

  Nasazení aplikace na správu nehod v Brně si lze představit například na úřadě, kde se dá očekávat, že se nehody budou aktualizovat (např. položka `nasledky`) a vkládat (nová nehoda). Mazání v tomto případě příliš smysl nedává, jelikož se záznam o nehodě uchovává delší dobu.

  Jednotlivé případy jsou řešeny dále a příslušné skripty jsou k nalezení ve složce:

  1. aktualizace
     - Skriptem `accident_update.py` s parametrem `--id` se identifikuje příslušná nehoda. Následně je uživatel skriptu dotázán na jednotlivé položky záznamu nehody. Nezadá-li nic, zůstanou původní hodnoty.
  2. vkládání
     - Skriptem `accident_insert.py` se vloží nový záznam nehody. Spouštění se provádí s parametrem `--location_id` následovaným názvem městské části existující v kolekci `locations`[13]. U záznamu nehody bude doplněno pole `MC` a zbytek informací doplní uživatel aplikace. Pro jednoduchost uvažujeme, že jelikož je městských částí konečně mnoho, všechny v kolekci již existují. Obdobným přístupem by se dal vytvořit skript pro mazání.

[10] `https://www.tutorialspoint.com/mongodb/mongodb_database_references.htm`

[11] Podobné zamyšlení lze nalézt i v manuálu MongoDB na adrese `https://www.mongodb.com/docs/manual/tutorial/model-referenced-one-to-many-relationships-between-documents/`

[12] Přejmenování `GlobalID` na `_id`, nekonzistence v malých a velkých písmenech se sjednotily pro pozdější rozumné agregování.

[13] Při každém spuštění se zobrazí výpis jednotlivých částí s příslušným číselným označením, aby se minimalizovala možnost chyby zápisu názvu.

Experimentální vyzkoušení tohoto postupu je k nalezení v souboru `update_insert_experiment.pdf`.

- Příkaz `UPSERT`
  Níže se uvažuje varianta, kdy vyjde nová verze datasetu a je třeba provést `UPSERT`. Žádoucím chováním je aktualizovat současné položky v databázi (je-li něco nového) a vložit nové záznamy. Postup je pak následující:

  1. Nová datová sada se nejprve zpracuje skriptem `preprocess_accidents.py`.
  2. Provede se import dat, který byl popsán dříve, navíc však s přepínačem `--upsert`.

  Jelikož globální identifikátor přichází jako součást datasetu, dochází skutečně k `UPSERT`u a nikoliv k celému shození databáze a jejímu opětovnému naplnění. K ověření se nejprve zjistil kompletní počet záznamů v databázi, nahrála se nová data obsahující úplně nový záznam (nové ID) a záznam, který již v databázi existoval a pouze se aktualizovaly některé položky.

  Pomocí `mongoimport` se nedají přidat nové položky v poli a zároveň ponechat původní, o tuto operaci se opět musí postarat skript (`update_locations_after_upsert.py`).

  Experimentální vyzkoušení tohoto postupu je k nalezení v souboru `upsert_experiment.pdf`.

- **Dotazy v jazyku databáze:**

  Následující dotazy demonstrují především agregační pipeline, která umožňuje pokládání komplikovanějších dotazů (distribuované výpočty). Obsah takového dotazu můžeme chápat jako pole akcí, které se sekvenčně vykonají a nějak upraví výchozí kolekci. Dále je třeba mít na paměti, že příkazy `$group` a `$sort` jsou výpočetně náročnější (přesupořádání dat mezi uzly), naopak příkazy jako `$match` nebo `$projection` nevyžadují výměnu dat mezi uzly.

```
> db.accidents.aggregate([
...     {
...         $match: { alkohol: { $ne: null } }
...     },
...     {
...         $group: {
...             _id: "$alkohol",
...             totalAccidents: { $sum: 1 },
...             avgDamage: { $avg: "$hmotna_skoda" },
...             totalFatalities: { $sum: "$usmrceno_os" }
...         }
...     }
... ])
{ "_id" : "pod vlivem alkoholu a drog", "totalAccidents" : 70, "avgDamage" : 71885.71428571429, "totalFatalities" : 0 }
{ "_id" : "ano, obsah alkoholu v krvi od 1,0‰ do 1,5‰", "totalAccidents" : 499, "avgDamage" : 71478.35671342685, "totalFatalities" : 3 }
{ "_id" : "ano, obsah alkoholu v krvi 1,5‰ a více", "totalAccidents" : 2058, "avgDamage" : 77586.39455782314, "totalFatalities" : 12 }
{ "_id" : "ano, obsah alkoholu v krvi od 0,8‰ do 1,0‰", "totalAccidents" : 139, "avgDamage" : 74611.5107913669, "totalFatalities" : 0 }
{ "_id" : "pod vlivem drog", "totalAccidents" : 310, "avgDamage" : 107148.3870967742, "totalFatalities" : 5 }
{ "_id" : "ano, obsah alkoholu v krvi od 0,24‰ do 0,5‰", "totalAccidents" : 501, "avgDamage" : 72289.62075848304, "totalFatalities" : 6 }
{ "_id" : "ano, obsah alkoholu v krvi do 0,24 ‰", "totalAccidents" : 325, "avgDamage" : 82541.53846153847, "totalFatalities" : 5 }
{ "_id" : "ano, obsah alkoholu v krvi od 0,5‰ do 0,8‰", "totalAccidents" : 255, "avgDamage" : 64447.05882352941, "totalFatalities" : 0 }
{ "_id" : "nezjišťován", "totalAccidents" : 30409, "avgDamage" : 30111.677463908713, "totalFatalities" : 20 }
{ "_id" : "ne", "totalAccidents" : 35175, "avgDamage" : 92927.5223880597, "totalFatalities" : 288 }
```

Obrázek 4: Výpočet celkového počtu nehod, usmrcení a průměrné škody v závislosti na požití návykových látek.

Data v dotazu 4 se agregují pomocí nového klíče `alkohol` (tj. uzly si mají rozdělit data na základě tohoto klíče) a následně se vykonají statistické operace. Z výchozí kolekce se tedy vytvoří „mezikolekce“ (tzv. „shuffling“ vytříděná dle nově zvoleného klíče, provede se požadovaná operace výpočtu průměru a sum, následně je výsledek vrácen klientovi.

```
> db.accidents.aggregate([
...     {
...         $group: {
...             _id: "$rok",
...             maxDamage: { $max: "$hmotna_skoda" },
...             minDamage: { $min: "$hmotna_skoda" },
...         }
...     },
...     {
...         $sort: { _id: -1 }
...     }
... ])
{ "_id" : 2022, "maxDamage" : 4000000, "minDamage" : 0 }
{ "_id" : 2021, "maxDamage" : 3220000, "minDamage" : 0 }
{ "_id" : 2020, "maxDamage" : 2350000, "minDamage" : 0 }
{ "_id" : 2019, "maxDamage" : 5131100, "minDamage" : 0 }
{ "_id" : 2018, "maxDamage" : 2230000, "minDamage" : 0 }
{ "_id" : 2017, "maxDamage" : 5360000, "minDamage" : 0 }
{ "_id" : 2016, "maxDamage" : 2105000, "minDamage" : 0 }
{ "_id" : 2015, "maxDamage" : 1550000, "minDamage" : 0 }
{ "_id" : 2014, "maxDamage" : 2370000, "minDamage" : 0 }
{ "_id" : 2013, "maxDamage" : 1790000, "minDamage" : 0 }
{ "_id" : 2012, "maxDamage" : 2100000, "minDamage" : 0 }
{ "_id" : 2011, "maxDamage" : 2200000, "minDamage" : 0 }
{ "_id" : 2010, "maxDamage" : 1360000, "minDamage" : 0 }
```

Obrázek 5: Výpis nejvyšších a nejnižších hmotných škod v jednotlivých letech seřazených sestupně.

Podobně jako v předchozím případě, v dotazu 5 se data mezi uzly přesupořádají pomocí `$group` příkazu s nově přiřazeným klíčem, následně se na takto seskupených datech vypočítá maximum a minimum. Aby výsledek byl zobrazen sestupně, využívá se `$sort` dle klíče s patřičným nastavením (-1 sestupně, 1 vzestupně).

```
> db.locations.aggregate([
...     {
...         $match: { _id: "Brno-střed" }
...     },
...     {
...         $lookup: {
...             from: "accidents",
...             localField: "accident_ids",
...             foreignField: "_id",
...             as: "accidentDetails"
...         }
...     },
...     {
...         $unwind: "$accidentDetails"
...     },
...     {
...         $limit: 5
...     },
...     {
...         $project: {
...             _id: 0,
...             hmotna_skoda: "$accidentDetails.hmotna_skoda",
...             ZSJ: "$accidentDetails.ZSJ"
...         }
...     }
... ])
{ "hmotna_skoda" : 180000, "ZSJ" : "Příční" }
{ "hmotna_skoda" : 31000, "ZSJ" : "Mášova" }
{ "hmotna_skoda" : 16000, "ZSJ" : "U stadiónu" }
{ "hmotna_skoda" : 210000, "ZSJ" : "Přízová" }
{ "hmotna_skoda" : 5000, "ZSJ" : "Radlas" }
```

Obrázek 6: Vypsání výší hmotných škod dané městské části.

V dotazu 6 se již kombinují 2 kolekce. Nejdříve se příkazem `$match` vyfiltrují dokumenty (konkrétně 1) s daným ID. Pomocí `$lookup` se provede ekvivalent JOIN operace u relačních databází a navážou se nehody ke konkrétní městské části. Jelikož stále pracujeme s indexovanými klíči, operace by měly být rychlé. Výstupem je pole `accidentDetails`, kde jsou všechny nehody (z dané městské části). Příkaz `unwind`[14] pak pole rozdělí do více dokumentů, které se následně zpracují v `$project` fázi. V těle projekce pak lze mít libovolné atributy záznamu nehody, avšak v tomto případě nás zajímala pouze finační výše škod a ulice. Nulová hodnota u `_id` určuje, kam dana půjdou; jestliže nastavíme identifikátor na určitou konstantu, data skončí právě na jednom uzlu. To přináší pro klienta výhodu v tom, že když čte takto získaná data sekvenčně, komunikuje pouze s jedním uzlem.

Výpočty však agregační pipeline nekončí a MongoDB podporuje i MapReduce. Na rozdíl od agregace v předešlých případech, kde se „mezikolekce“ ukládaly pouze do operační paměti, se u MapReduce tento výsledek uloží do nové kolekce, na kterou pak lze nahlédnout příkazem `db.newCollection.find()`.

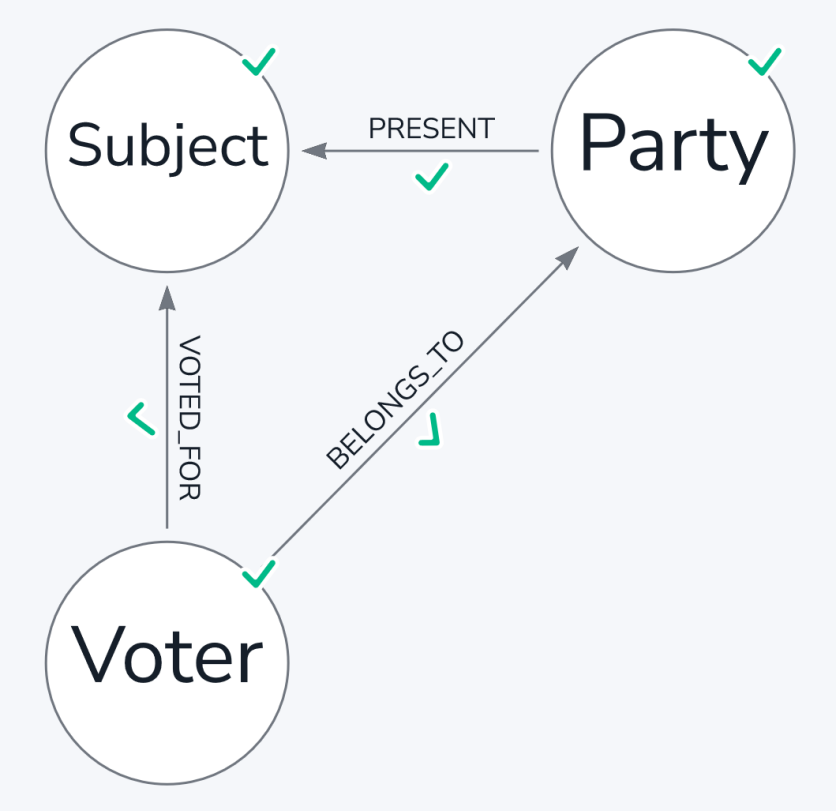
[14] `https://www.mongodb.com/docs/manual/reference/operator/aggregation/unwind/`

2.3 Grafová NoSQL databáze

- **Název datové sady:** Hlasování zastupitelstva
- **Distribuce datové sady:** JSON (jediný dostupný formát) převedený na CSV
- **Zvolený zástupce NoSQL databáze:** Neo4J
- **Zdůvodnění využití dané sady a databáze:** Grafové databáze využívají grafové struktury k reprezentaci dat (uzly) a vztahů mezi nimi (hrany). Struktura databáze a vztahy mezi jednotlivými uzly umožňují efektivní práci a dotazování nad provázanými daty – podobné dotazy by v jiných databázích mohly vést ke komplikacím (např. složité JOIN operace v SQL databázích). Grafové databáze jsou tedy vhodné pro reprezentaci topologií (sítě – dopravní, internetové, ...) nebo ukládání provázaných dat.

  Vybraná datová sada obsahuje informace o hlasování zastupitelstva města Brna. Nachází se v ní údaje o polických stranách a hnutích, členech zastupitelstva a každém jejich hlasování (příp. nepřítomnosti). Kromě toho v ní lze nalézt také témata, o kterých se hlasovalo a výsledky hlasování (přijato, nepřijato). Protože do zvolené grafové databáze Neo4J nelze jednoduše načítat data ve formátu JSON, byl vytvořen krátký skript v jazyce Python, který data převede do formátu CSV.

  Nová data vychází po každém zasedání zastupitelstva, tedy cca jednou za měsíc. Aktuální data jsou dostupná z webu města Brna[15]. Protože se jedná o dostupná data se zájmem veřejnosti, lze sadu považovat za poměrně důvěryhodnou. Ke změnám již publikovaných dat by docházet nemělo (avšak kvůli lidskému faktoru se to stát může), spíše lze očekávat přidávání dat nových z novějších zasedání.



Obrázek 7: Jednoduché schéma grafové databáze

[15] https://kod.brno.cz/zastupitelstvo/

Cílem projektu je vytvořit základ (back-end) pro klientskou aplikaci, která zveřejňovaná data následně může vizualizovat a prezentovat. Jedná se tedy zejména o informace o jednotlivých osobách a jejich hlasování. K tomu lze efektivně využít právě grafové databáze, reprezentující hlasujícího, jeho vztah k politické straně nebo hnutí a informaci o tom, jak pro konkrétní témata hlasoval.

Bylo vytvořeno jednoduché databázové schéma (viz obrázek 7). V databázi se tedy nachází uzly:

1. **Pro hlasující** - obsahuje jméno hlasujícího
2. **Pro politické strany/hnutí** - obsahuje jméno strany/hnutí
3. **Pro hlasování** - obsahuje téma hlasování, čas a výsledek

Vztah **BELONGS_TO** slouží k určení politické příslušnosti hlasujícího. Vztah **VOTED_FOR** vyjadřuje, jak daný hlasující hlasoval při konkrétním tématu – vztah samotný tedy obsahuje vlastnost **vote** s informací, zda hlasoval *ano*, *ne*, *zdržel se* nebo *nebyl přítomen*. Poslední vztah **PRESENT** vyjadřuje, zda byla daná strana/hnutí přítomna při hlasování. Tento vztah umožňuje pracovat s přehledy hlasování z pohledů stran, neboť v průběhu času mohou hlasující být členy více různých skupin, např. volebních koalic.

- **Příkazy pro definici úložiště a algoritmický popis importu dat:**

```
// Constraints, make sure nodes are unique
CREATE CONSTRAINT ON (s:Subject) ASSERT s.id IS UNIQUE;
CREATE CONSTRAINT ON (v:Voter) ASSERT v.name IS UNIQUE;
CREATE CONSTRAINT ON (p:Party) ASSERT p.name IS UNIQUE;

// Nodes creation - use MERGE for upsert, ensuring uniqueness
LOAD CSV WITH HEADERS FROM "data.csv" AS row
MERGE (n:Subject {id: row.id})
ON CREATE SET n.subject = row.subject,
              n.time = datetime(row.time),
              n.result = row.result
MERGE (voter:Voter {name: row.name})
MERGE (party:Party {name: row.party})

// Edges creation, again using MERGE so only one relationship
// of a given type will ever be created between a pair of nodes
MERGE (voter)-[:VOTED_FOR {vote: row.vote}]->(n)
MERGE (voter)-[:BELONGS_TO]->(party)
MERGE (party)-[:PRESENT]->(n);
```

Kód v jazyce Cypher pro vytvoření a naplnění úložiště. Slouží jak k počátečnímu naplnění, tak k případnému doplnění nových či změněných dat – toho je dosaženo pomocí příkazu MERGE, který zaručí neduplicitní vložení a případnou aktualizaci již existujících uzlů a hran.

Nejprve jsou explicitně vytvořeny omezení zaručující jedinečnost všech tří typů uzlů, ale také automatické vytvoření indexů pro rychlejší vyhodnocení dotazů. Následně jsou z vytvořeného CSV vytvořeny uzly pro téma (Subject), hlasujícího (Voter), politickou stranu/hnutí (Party) a vztahů mezi nimi.

- **Dotazy v jazyce databáze:**

1. Základní dotaz na seznam hlasovacích témat. Tento dotaz správně vrátí 4706 řádků, t.j. hlasovacích témat uložených v databázi.

```
MATCH (s:Subject)
RETURN s.subject;
```

2. Dotaz na počet různých členů, které politická strana/hnutí celkem měla. Využívá vztahu mezi stranou a hlasujícím **BELONGS_TO** a agregační funkze SIZE k získání počtu takových vztahů každé strany. Výsledek je seřazen sestupně.

```
MATCH (p:Party)
RETURN p, SIZE((p)<-[:BELONGS_TO]-(:Voter)) as count
ORDER BY count DESC;
```

p	count
(:Party {name: "ANO 2011"})	25
(:Party {name: "ODS a TOP 09"})	16
(:Party {name: "ODS"})	15
(:Party {name: "Piráti"})	12
(:Party {name: "Nezávislí"})	12
(:Party {name: "Lidovci a Starostové"})	10
(:Party {name: "KDU-ČSL"})	8
(:Party {name: "SPD+Trikolora"})	6
(:Party {name: "ČSSD"})	6
(:Party {name: "Nezávislí pro Brno"})	6
(:Party {name: "SPD"})	4
(:Party {name: "SOCDEM"})	3
(:Party {name: "Zelení a Žít Brno"})	3
(:Party {name: "ČSSD VAŠI STAROSTOVÉ"})	3

Obrázek 8: Výsledné počty členů politických stran

3. Statistika hlasování pro konkrétního hlasujícího. Vztah **VOTED_FOR** obsahuje vlastnost `vote` s informací o konkrétním hlasování. Každý záznam je následně agregován pomocí funkce SUM.

```
MATCH (v:Voter {name: 'Petr Vokřál'})-[vote:VOTED_FOR]->(:Subject)
RETURN
    v.name,
    SUM(CASE WHEN vote.vote = 'Ano' THEN 1 ELSE 0 END) AS Ano,
    SUM(CASE WHEN vote.vote = 'Ne' THEN 1 ELSE 0 END) AS Ne,
    SUM(CASE WHEN vote.vote = 'Zdržel se' THEN 1 ELSE 0 END) AS ZdrzelSe,
    SUM(CASE WHEN vote.vote = 'None' THEN 1 ELSE 0 END) AS Nepritomen;
```

v.name	Ano	Ne	ZdrzelSe	Nepritomen
"Petr Vokřál"	2053	25	419	991

Obrázek 9: Statistika hlasování pro hlasujícího Petr Vokřál

Obdobně jednoduše lze zjistit například statistiku absence daného hlasujícího.

```
MATCH (v:Voter {name: 'Markéta Vaňková'})-[vote:VOTED_FOR]->(:Subject)
WITH
```

```
    v.name AS name,
    SUM(CASE WHEN vote.vote = 'None' THEN 1 ELSE 0 END) AS Nepritomen,
    COUNT(vote.vote) AS Celkem
RETURN
    name,
    Celkem - Nepritomen AS Pritomen,
    Nepritomen,
    (Nepritomen*100.0)/Celkem AS Absence;
```

```
+-----------------------------------------------------------------+
| name              | Pritomen | Nepritomen | Absence             |
+-----------------------------------------------------------------+
| "Markéta Vaňková" | 4539     | 171        | 3.6305732484076434 |
+-----------------------------------------------------------------+
```

Obrázek 10: Počty hlasování a procentuální absence hlasující Markéty Vaňkové

Zajímavý je pohled do profileru při zpracování tohoto dotazu:

```
+-----------------------------------------------------------------------------------------+
| Plan      | Statement   | Version      | Planner | Runtime       | Time | DbHits | Rows |
+-----------------------------------------------------------------------------------------+
| "PROFILE" | "READ_ONLY" | "CYPHER 3.5" | "COST"  | "INTERPRETED" | 163  | 18844  | 1    |
+-----------------------------------------------------------------------------------------+


+----------------------+----------------+------+---------+------------+-----------------------------------------------------+
| Operator             | Estimated Rows | Rows | DB Hits | Cache H/M  | Identifiers                                         |
+----------------------+----------------+------+---------+------------+-----------------------------------------------------+
| +ProduceResults      |             53 |    1 |       0 |       0/0  | name, Pritomen, Nepritomen, Celkem, Absence         |
| |                    +----------------+------+---------+------------+-----------------------------------------------------+
| +Projection          |             53 |    1 |       0 |       0/0  | name, Pritomen, Nepritomen, Celkem, Absence         |
| |                    +----------------+------+---------+------------+-----------------------------------------------------+
| +EagerAggregation    |             53 |    1 |    9420 |       0/0  | Celkem, Nepritomen, name                            |
| |                    +----------------+------+---------+------------+-----------------------------------------------------+
| +Filter              |           2765 | 4710 |    4710 |       0/0  | anon[61], cached[v.name], v, vote                   |
| |                    +----------------+------+---------+------------+-----------------------------------------------------+
| +Expand(All)         |           2765 | 4710 |    4711 |       0/0  | anon[61], cached[v.name], v, vote                   |
| |                    +----------------+------+---------+------------+-----------------------------------------------------+
| +NodeUniqueIndexSeek |              1 |    1 |       3 |       0/0  | cached[v.name], v                                   |
+----------------------+----------------+------+---------+------------+-----------------------------------------------------+
```

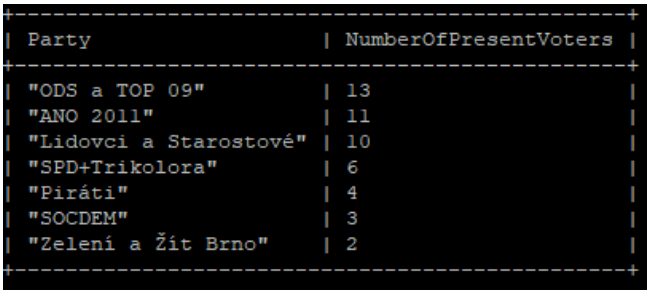
Obrázek 11: Výstup profileru (query PROFILE) posledního dotazu na absenci hlasujícího

Ačkoliv stačilo zjistit 4710 hlasování, resp. přítomnosti při nich, profiler hlásí skoro devatenáct tisíc přístupů do databáze (`DbHits`). Při bližším prozkoumání lze zjistit, že dotaz na hlasující je rychlý a efektivní díky indexům a cachování, nicméně vztahy **VOTED_FOR** jsou dotazovány několikrát: Nejprve jejich vyhledání a expanze `Expand(All)`, následné filtrování (část query **CASE WHEN**) a následná okamžitá agregace `EagerAggregation`, která ale probíhá dvakrát – jednou pro statistiku *Nepritomen* a podruhé pro *Celkem*, čímž výrazně roste počet přístupů do databáze.

Pokud bychom tedy takovéto dotazy chtěli používat častěji, bylo by vhodné je optimalizovat. Možných přístupů je několik, např. načíst záznamy o hlasování z databáze jen jednou a poté je teprve zpracovat, zefektivnit použití filteru, aby počítal jak absenci, tak přítomnost apod.

4. Počet hlasujících každé strany přítomných při konkrétním hlasování (seřazeno sestupně)

```
MATCH (s:Subject {id: 'Z9/09-130'})<-[vote:VOTED_FOR]-(v:Voter)
WHERE NOT vote.vote = 'None'
MATCH (v)-[:BELONGS_TO]->(p:Party)-[:PRESENT]->(s)
RETURN p.name AS Party, COUNT(v) as NumberOfPresentVoters
ORDER BY NumberOfPresentVoters DESC;
```



Party	NumberOfPresentVoters
"ODS a TOP 09"	13
"ANO 2011"	11
"Lidovci a Starostové"	10
"SPD+Trikolora"	6
"Piráti"	4
"SOCDEM"	3
"Zelení a Žít Brno"	2

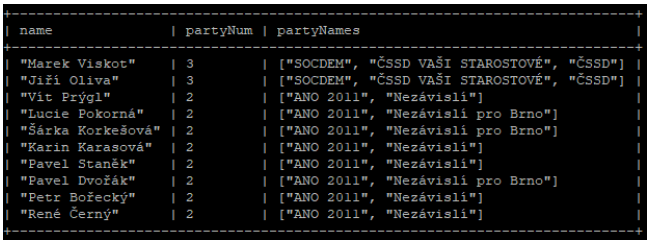
Obrázek 12: Počet přítomných hlasujících každé strany při hlasování Z9/09-130

Tento dotaz krásně znázorňuje přednosti grafových databází. Využívá všech vztahů mezi jednotlivými uzly a bez nutné vysoké redundance dat v databázi dokáže ”zjistit”i takovouto komplexní informaci. Nejprve tedy najde všechny přítomné hlasující (`vote.vote != NONE`) pro dané téma `Z9/09-130`. Následně je využito vztahů s uzly stran, které jsou dále ve vztahu **PRESENT** s konkrétním hlasováním, čímž je dosaženo požadovaného výsledku.

- **Další zajímavé dotazy**

  Hlasující, kteří v minulosti změnili politickou příslušnost. Obsahuje i výsledky pro zapojení do volební koalice, případně přejmenování stejné politické strany nebo hnutí:

```
MATCH (v:Voter)-[:BELONGS_TO]->(party:Party)
WITH v.name AS name, COLLECT(DISTINCT party.name) AS partyNames
WHERE SIZE(partyNames) > 1
RETURN name, SIZE(partyNames) AS partyNum, partyNames
ORDER BY partyNum DESC LIMIT 10;
```



name	partyNum	partyNames
"Marek Viskot"	3	["SOCDEM", "ČSSD VAŠI STAROSTOVÉ", "ČSSD"]
"Jiří Oliva"	3	["SOCDEM", "ČSSD VAŠI STAROSTOVÉ", "ČSSD"]
"Vít Prýgl"	2	["ANO 2011", "Nezávislí"]
"Lucie Pokorná"	2	["ANO 2011", "Nezávislí pro Brno"]
"Šárka Korkešová"	2	["ANO 2011", "Nezávislí pro Brno"]
"Karin Karasová"	2	["ANO 2011", "Nezávislí"]
"Pavel Staněk"	2	["ANO 2011", "Nezávislí"]
"Pavel Dvořák"	2	["ANO 2011", "Nezávislí pro Brno"]
"Petr Bořecký"	2	["ANO 2011", "Nezávislí"]
"René Černý"	2	["ANO 2011", "Nezávislí"]

Obrázek 13: Hlasující, kteří byli členové různých politických stran nebo hnutí

Deset témat s nejvíce přítomnými hlasujícími:

```
MATCH (s:Subject)<-[:VOTED_FOR]-()
RETURN s.time, s.subject, COUNT(*) AS voteCount
ORDER BY voteCount DESC LIMIT 10;
```

2.4 NoSQL databáze časových řad

- **Název datové sady:** Sčítače cykl. dopravy – data z měření
- **Distribuce datové sady:** CSV formát
- **Zvolený zástupce NoSQL databáze:** InfluxDB
- **Zdůvodnění využití dané sady a databáze:** Databáze pro časové řady se jeví jako ideální řešení pro uložení (a případné zpracování) dat ze senzorů a podobných zařízení, která chrlí data v různě dlouhých intervalech. Obecně platí, že v takových databázích dochází často k zápisu a méně k hledání, zároveň starší záznamy se neaktualizují, ale spíše agregují, případně úplně mažou[16]. Obecná struktura se skládá z názvu měření, tags a fields:

  ```
  measurement(,tag_key=tag_val)*
  field_key=field_val(,field_key_n=field_value_n)*
  (nanoseconds-timestamp).
  ```

  Nové záznamy proudící přes tzv. influx line musí být tohoto formátu[17] a zároveň platí, že položky, které mají tyto položky (measurement, tag_keys, field_keys) stejné, budou uloženy na stejném uzlu v clusteru, což zajišťuje škálovatelnost. Při návrhu databáze je nutné myslet na to, které položky se budou využívat při dotazech, a tedy kdy se vyplatí je indexovat. Tags indexované jsou, fields nikoliv. InfluxDB dále nabízí tzv. „continuous query“, tj. opakované záznamy, které lze využít např. pro periodickou agregaci starších záznamů.

  S podobným záměrem uložení a práce s daty by šlo využít i MongoDB, nicméně InfluxDB umí lépe pracovat s časovou složkou. Nemají-li data příliš komplikovanou strukturu, kterou by lépe zpracovala dokumentová databáze, je InfluxDB vhodnější volbou. Vlastnosti sloupcových ani grafových NoSQL databází pro tento typ datasetu by taktéž nebyly využity.

  Dataset výstupů sčítače cyklistické dopravy[18] sestává z periodických měření. Každý záznam v databázi má tedy svoji časovou značku, název měření, místo měření a počet cyklistů přijíždějících zleva a zprava. Indexovaný tag zajišťující rychlé dotazování/filtrování je lokace, samotné sčítání cyklistů jsou pak neindexované fields. V InfluxDB pak lze agregovat výstupy na základě časových intervalů a na takto seskupených datech dále provádět základní statistické a matematické operace[19]. Z klientské strany pak použitím takových agregačních dotazů můžeme získat data, která poslouží k tvorbě grafů či jakýchkoliv jiných reprezentací s požadovanou abstrakcí[20].

- **Příkazy pro definici úložiště:** Na virtuálním stroji se vytvořila databáze příkazem

  ```
  influx -execute "CREATE DATABASE cyclists_database;",
  ```

  soubor se zdrojovými daty se pak do vytvořené databáze nahrál příkazem

  ```
  influx -import -path=cyclists.csv -precision=s
  -database=cyclists_database,
  ```

[16] možnost nastavení TTL (time-to-live)

[17] Tato data jsou většinou ukládána v komprimované podobě, např. s rozdílem časů a hodnot měření.

[18] `https://data.gov.cz/zdroj/datov-sady/00266094/cc18af56dbfab9b259bb88f149a275ee`

[19] Výčet funkcí je k nalezení v dokumentaci: `docs.influxdata.com/influxdb/v1/query_language/functions/`

[20] Např. není potřeba mít data po vteřině, ale mít graf, ve kterém se využívají agregované hodnoty po minutě.

kde název měření byl `cyclistsCount`.

- **Algoritmický popis importu dat:** Soubor `cyclists.csv` se zdrojovými daty byl vytvořen skriptem `preprocess_cyclists.py`, který z původního datasetu vybral potřebná data a uložil je v kýženém formátu (popsaném v předchozích částech).

  Vzhledem k typu databáze nemá příliš smysl uvažovat aktualizaci dat. Jak bylo popsáno v úvodním bodu této kapitoly, databáze je určena k masivnímu zápisu a starší data se spíše seskupují či přímo zahazují.

- **Dotazy v jazyce databáze:**

  Příkazem na snímku 14 si lze vypsat přítomné časové řady.

```
> show series
key
---
cyclistsCount,location=M016
cyclistsCount,location=M027
cyclistsCount,location=M028
```

Obrázek 14: Využití `SHOW SERIES` příkazu

InfluxDB poté využije jednu z uvedených kombinací z 14, aby zjistil, na kterém uzlu je daná časová řada uložena, tj. data jsou distribuována na základě názvu časové řady. To ve výsledku znamená, že všechna měření patřící do příslušné časové řady spadají do stejného uzlu, což následně vede k dobrému výkonu při výpočtech (data jsou u sebe). Když chceme data distribuovat přes cluster, využívá se tzv. `randtag`, jehož hodnota rozhodne o tom, na kterém uzlu bude časová řada umístěna.

Další dotazy budou popsány již pouze z hlediska sémantiky (a případně syntaxe).

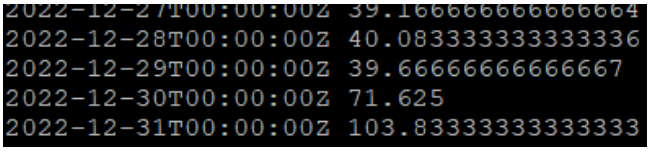
```
SELECT MEAN("cyclistsFromLeft")
FROM "cyclistsCount"
GROUP BY *
```

```
> SELECT MEAN("cyclistsFromLeft") FROM "cyclistsCount" GROUP BY *
name: cyclistsCount
tags: location=M016
time                 mean
----                 ----
1970-01-01T00:00:00Z 16.530998338233413

name: cyclistsCount
tags: location=M027
time                 mean
----                 ----
1970-01-01T00:00:00Z 3.512987012987013

name: cyclistsCount
tags: location=M028
time                 mean
----                 ----
1970-01-01T00:00:00Z 7.832792207792208
```

Obrázek 15: Využití GROUP BY * klauzule pro vypočítání průměrů na jednotlivých lokacích.

Dotaz 15 vrací průměrné hodnoty průjezdu cyklistů (zleva) z jednotlivých lokací. Hvězdička u GROUP BY udává, že klauzule vrací všechny možné kombinace tags. Jelikož jsou však tags pouze lokace (a nabývají 3 hodnot), výstup jsou průměry pro jednotlivé lokace.

```
SELECT MEAN("cyclistsFromLeft") + MEAN("cyclistsFromRight")
AS "averageDailyCyclistCount"
FROM "cyclistsCount"
WHERE "location" = 'M016'
  AND time >= '2022-01-01T00:00:00Z'
  AND time < '2023-01-01T00:00:00Z'
GROUP BY time(24h)
```



Obrázek 16: Ukázka části výstupu dotazu na průměrnou denní vytíženost v lokaci M016.

Dotaz 16 demonstruje seskupování dat po určitém intervalu a ukazuje schopnost InfluxDB efektivně pracovat s časem. V dotazu se nejdříve sečtou denní průměry průjezdů cyklistů zleva i zprava, dále je specifikována konkrétní lokace a vymezen časový interval odpovídající 1 dni. To se následně seskupí po 24 hodinách a výpisem jsou pak průměry pro jednotlivé dny.

Další dotazy by mohly směřovat k analýze provozu v určitou denní dobu, příp. provést podobný dotaz na všech lokacích a srovnat vytíženost cyklistickou dopravou.

3 Závěr

Cílem projektu bylo provést analýzu datových sad poskytovaných Národním katalogem otevřených dat a navrhnout optimální způsob jejich uložení do vhodných NoSQL databází. Demonstrovaly se různé typy NoSQL databází, včetně sloupcových, dokumentových, grafových a databází časových řad, a pro každý z těchto typů databází byla vybrána datová sada. Následně se argumentovalo, proč je daný typ NoSQL databáze pro tuto sadu dat vhodnou volbou.

Byly popsány charakteristické vlastnosti datové sady a důvody, proč je daný typ NoSQL databáze ideální pro její uložení a dotazování. Pro každý druh databáze byly také poskytnuty příkazy pro definici úložiště, import dat a ukázky dotazů nad uloženými daty.

V neposlední řadě byla snaha o navržení optimálního způsobu uložení dat, které reflektovaly potenciální požadavky aplikace na databázi.

Všechny využité skripty jsou k nalezení ve složce s projektem v odevzdaném archivu.