Vysoké učení technické v Brně
Fakulta informačních technologií

Ukládání rozsáhlých dat v NoSQL databázích

Matej Focko, **Maksym Tolstopiat**, **Nikita Zhukov**

Brno, 2023

Sloupcová wide-column databáze

Zvolená datová sada: *Average amount of pensions of pensioners with newly granted pensions in the Czech Republic*[1]

Vybraná distribuce z katalogu

CSV je v tomto případě nejlepší volbou, protože při ukládání velkého množství dat zabírá méně místa než formát JSON. Data v tomto datovém souboru jsou navíc strukturovaná, takže CSV je také nejlepší volbou, protože se dobře hodí pro ukládání tabulkově strukturovaných dat.

Zvolená databáze a důvod

Zvolená databáze: *Apache Cassandra*

Na základě vybraného souboru dat můžeme vyvodit následující závěry o tom, proč je lepší používat Apache Cassandra než jinou NoSQL databázi. V první řadě hraje důležitou roli velikost datové sady. Apache Cassandra funguje lépe než jiné databáze NoSQL na velkých souborech dat, protože poskytuje lepší výkon při čtení a zápisu než jiné databáze. Tento aspekt je velmi důležitý, protože tento soubor dat obsahuje finanční náklady pro důchodce a může být státem často využíván a analyzován. Kromě toho jsme analyzovali schéma této datové sady a všimli jsme si, že má tabulkovou formu a používá primární klíč k zajištění jedinečnosti záznamů. To znamená, že tato datová sada je strukturovaná a skutečnost, že obsahuje primární klíč, již omezuje výběr NoSQL databází, které jej mohou podporovat. Období aktualizace datové sady je každý rok. Ale navzdory skutečnosti, že je aktualizován poměrně zřídka, je do něj přidáno velké množství nových dat, což vyžaduje škálovatelnost. Cassandra je distribuovaná databáze, což znamená, že ji lze sdílet mezi více uzly. To umožňuje Cassandře horizontálně škálovat a podle potřeby přidávat další uzly. Navíc podporuje replikaci dat napříč

[1] https://data.gov.cz/dataset?iri=https%3A%2F%2Fdata.gov.cz%2Fzdroj%2FdatovÃĺ-sady%2F00006963%2F2a0d1ce1f451f0d4fb8677ed293a68bd

více uzly, což zlepšuje celkový výkon. Dalším důvodem pro výběr je analýza dat a vizualizace. Například analýza průměrné výše důchodu pro každý typ důchodu. Cassandra může být použita k analýze průměrné výše důchodu pro každý typ důchodu. To může být užitečné pro pochopení toho, jak se liší průměrné částky důchodu u různých typů důchodů. Vizualizace průměrné výše důchodu pro každý typ důchodu. Cassandra lze použít k vizualizaci průměrné výše důchodu pro každý typ důchodu. To může být užitečné pro pochopení trendů průměrných důchodů. Můžeme tedy dojít k závěru, že se nejlépe hodí pro velké objemy strukturovaných dat, která jsou předmětem časté analýzy a vizualizace. Například burzy, knihovny, sociální sítě.

Definice úložiště

Schéma je statická definice dat uložených v tabulce. Jakmile je tabulka definována, nelze ji změnit. Skládá se z názvů sloupců jejich typů a primárního klíče. Kde primární klíč je složený na základě schématu vytvořeného samotnými vývojáři tohoto datasetu a ke všemu přidáme řazení dat podle jednoho z klíčů sekcí, a to od velkého po malý podle data. Protože uživatel nejprve potřebuje vidět nejnovější platby. Takové schéma lze definovat pro daný soubor dat.

```
CREATE TABLE pensions (
    druh_duchodu_kod text,
    druh_duchodu text,
    pohlavi_kod text,
    pohlavi text,
    referencni_obdobi text,
    prumerna_vyse_duchodu_u_nove_priznanych_duchodu double,
    PRIMARY KEY (
        druh_duchodu_kod, pohlavi_kod, referencni_obdobi
    )
) WITH CLUSTERING ORDER BY (referencni_obdobi DESC);
```

Příklady příkazů pro manipulaci s daty

Vkládání dat

Nejdřív popíšeme příkazy pro vkládání dat.

1. vložení jedného záznamu

   ```
   INSERT INTO pensions (
       druh_duchodu_kod, druh_duchodu, pohlavi_kod, pohlavi,
   ```

```
    referencni_obdobi, prumerna_vyse_duchodu_u_nove_priznanych_duchodu
) VALUES (
    'starobni_duchod', 'Starobní důchod', 'M', 'Muž', '2022', 15000
);
```

2. vložení několik záznamů najednou

```
INSERT INTO pensions (
    druh_duchodu_kod, druh_duchodu, pohlavi_kod, pohlavi,
    referencni_obdobi, prumerna_vyse_duchodu_u_nove_priznanych_duchodu
) VALUES (
    'starobni_duchod', 'Starobní důchod', 'F', 'Žena', '2023', 10000,
    'invalidni_duchod', 'Invalidní důchod', 'M', 'Muž', '2023', 5000
);
```

Při zápisu dat do tabulky Cassandra je třeba dodržovat následující pravidla:

- Všechny sloupce definované ve schématu tabulky musí být vyplněny hodnotami.
- Hodnoty sloupců se musí shodovat s datovými typy zadanými ve schématu tabulky.
- Primární klíč musí být jedinečný pro každý záznam v tabulce.

Dotazy nad daty

Teď popíšeme dotazy nad daty.

1. všechny záznamy z tabulky

```
SELECT * FROM pensions;
```

2. hodnoty v konkrétním sloupci

```
SELECT druh_duchodu FROM pensions;
```

3. záznamy mužů, kteří pobírají starobní důchod v roce 2023

```
SELECT *
    FROM pensions
    WHERE druh_duchodu_kod = 'starobni_duchod'
        AND pohlavi_kod = 'M'
        AND referencni_obdobi = '2023';
```

4. druhy důchodů za podmínky, že jejich nový průměrný důchod je vyšší než 10000

```
SELECT druh_duchodu
    FROM pensions
    WHERE prumerna_vyse_duchodu_u_nove_priznanych_duchodu > 10000;
```

Při načítání dat z tabulky Cassandra je třeba dodržovat následující pravidla:

- Výrazy podmínky musí být platné pro datový typ vybraného sloupce.
- Výrazy podmínky musí být pravdivé pro šechny záznamy v tabulce.
- Pokud tato pravidla nejsou dodržena, Cassandra vyhodí chybu.

Aktualizace dat

Dále uvedeme příklady, jak můžete aktualizovat data v tabulce.

1. Aktualizace hodnoty sloupce v jednom záznamu

```
UPDATE pensions
    SET prumerna_vyse_duchodu_u_nove_priznanych_duchodu = 20000
    WHERE druh_duchodu_kod = 'starobni_duchod'
        AND pohlavi_kod = 'M'
        AND referencni_obdobi = '2023';
```

2. Aktualizace hodnot sloupců ve více záznamech

```
UPDATE pensions
    SET druh_duchodu = 'Invalidní důchod',
        prumerna_vyse_duchodu_u_nove_priznanych_duchodu = 10000
    WHERE druh_duchodu_kod = 'starobni_duchod'
        AND pohlavi_kod = 'F'
        AND referencni_obdobi = '2023';
```

3. Smazání záznamů

```
DELETE FROM pensions
    WHERE druh_duchodu_kod = 'starobni_duchod'
        AND pohlavi_kod = 'M'
        AND referencni_obdobi = '2023';
```

4. Smazání všech položek z tabulky důchodů

```
TRUNCATE pensions;
```

Při aktualizaci dat v tabulce Cassandra je třeba dodržovat následující pravidla:

- Všechny sloupce, které je třeba aktualizovat, musí být zadány v příkazu `UPDATE`.
- Hodnoty sloupců se musí shodovat s datovými typy zadanými ve schématu tabulky.
- Primární klíč musí být jedinečný pro každý záznam v tabulce.

Při odstraňování dat v tabulce Cassandra je třeba dodržovat následující pravidla:

- Smazání záznamu se provádí pomocí primárního klíče.
- Záznam, který má být odstraněn, musí v tabulce existovat.

Odstranění všech záznamů z tabulky lze také provést pomocí příkazu `DROP TABLE`, ale tento příkaz odstraní tabulku spolu s její strukturou.

Zde je příklad příkazu k odstranění tabulky důchodů:

```
DROP TABLE pensions;
```

Import dat

read data set;
foreach *record in the data set* **do**
  get key of the record;
  if *record with the same key is already in the database* **then**
    execute `UPDATE` statement for updating the record;
  else
    execute `INSERT` statement for inserting the record;
  end
end

Algoritmus 1: Načtení dat

Příklad skriptu v Pythonu:

```
import pandas as pd
import cassandra

# Připojte se k databázi Cassandra
session = cassandra.Session("localhost", port=9042)

# Číst datovou sadu
df = pd.read_csv("data.csv")

# Získejte seznam klíčů záznamu
keys = df["key"].tolist()
```

```
# Zpracujte každý záznam v datové sadě
for i, row in df.iterrows():
    key = row["key"]
    value = row["value"]

    # Zkontrolujte, zda v databázi existuje záznam se stejným klíčem
    query = f"SELECT * FROM pensions WHERE key = {key}"
    results = session.execute(query)

    # Pokud záznam existuje, aktualizujte jej
    if results:
        session.execute(
            f"UPDATE pensions SET value = {value} WHERE key = {key}"
        )
    # V opačném případě přidejte nový záznam
    else:
        session.execute(
            f"INSERT INTO pensions (key, value) VALUES ({key}, {value})"
        )

# Zavřete připojení k databázi
session.close()
```

Výhody použití algoritmu:

- Algoritmus zabraňuje opakovanému vkládání dat do databáze.
- Algoritmus je snadno implementovatelný.
- Algoritmus lze použít na jakoukoli datovou sadu a jakoukoli databázi NoSQL.

Nevýhody algoritmu:

- Algoritmus nemusí být účinný pro velké soubory dat.

Dokumentová databáze

Zvolená datová sada: *Cemetery boundaries*[2]

Vybraná distribuce z katalogu

Vzhledem k tomu, že zde bude použita databáze orientovaná na dokumenty, bylo by nejlepší volbou použít formát GeoJSON, který tento typ databáze podporuje. Navíc obsahuje vložená data, konkrétně hranice hřbitova, což je další důvod pro použití formátu GeoJSON.

Zvolená databáze a důvod

Zvolená databáze: *MongoDB*

Pro tento datový soubor jsme zvolili MongoDB na základě datového typu. V tomto případě existují prostorová data, která ne všechny databáze NoSQL podporují, a kromě toho samotná datová struktura v tomto datovém souboru je spíše dokumentově orientovaná, protože existují jak prostorová data, tak zcela odlišné typy dat. Kromě toho MongoDB umožňuje používat velké množství již vestavěných funkcí pro analýzu a práci s daty. Například na základě délky a plochy hřbitova lze vytvořit jeho hranice pomocí zabudovaných funkcí MongoDB.

Definice úložiště

Následující kód vytvoří kolekci s názvem `cemeteries` a definuje schéma pro tuto kolekci.

```
// Vytvoření kolekce
const cemeteries = db.collection('cemeteries');

// Definice schématu
const cemeterySchema = {
```

[2] https://data.gov.cz/dataset?iri=https%3A%2F%2Fdata.gov.cz%2Fzdroj%2FdatovÃĺ-sady%2F44992785%2F08b91022e5338549b92e7a74cf1d679a

```
  _id: { type: 'ObjectId', required: true },
  ogcfid: { type: 'number', required: true },
  name: { type: 'string', required: true },
  datum_exportu: { type: 'date', required: true },
  GlobalID: { type: 'string', required: true },
  SHAPE_Length: { type: 'number', required: true },
  SHAPE_Area: { type: 'number', required: true },
};

// Přidání schématu do kolekce
cemeteries.createIndex(cemeterySchema);

// Vložení dokumentů
for (const cemetery of cemeteriesData) {
  cemeteries.insertOne(cemetery);
}
```

Schéma obsahuje následující pole:

- `_id`: Unikátní identifikátor dokumentu
- `ogcfid`: Identifikátor hřbitova v OpenGeoCodeFramework
- `name`: Název hřbitova
- `datum_exportu`: Datum exportu dat
- `GlobalID`: Globální identifikátor hřbitova
- `SHAPE_Length`: Délka hranice hřbitova
- `SHAPE_Area`: Plocha hřbitova

Příklady příkazů pro manipulaci s daty

V MongoDB je mnoho datových operací. Tyto operace lze rozdělit do dvou kategorií:

- Operace CRUD jsou operace vytváření, čtení, aktualizace a mazání dokumentů.
- Souhrnné operace jsou operace umístění podřízených čísel nad naducumi.

Dále uvedeme jeden příklad pro každou z možných operací založených na vybraném datovém souboru.

Agregační funkce

1. Vyhledávání podle dokumentu

   ```
   db.cemeteries.find({
     "name": "Líšeň"
   });
   ```

   Tento kód vrátí všechny dokumenty ze sbírky hřbitovů, které odpovídají kritériu `name = "Líšeň"`. V tomto případě bude vrácen pouze jeden dokument, protože ve fondu je pouze jeden hřbitov s názvem "Líšeň".

2. Seskupování dokumentů

   ```
   db.cemeteries.group({
       "_id": {
           "name": "$name"
       },
       "count": {
           $sum: 1
        }
   });
   ```

   Tento kód seskupuje dokumenty ze sbírky hřbitovů podle pole názvu a počítá počet dokumentů v každé skupině. Výsledkem bude sada dokumentů, kde každý dokument obsahuje následující:

   - `_id`: Název hřbitova
   - `count`: Počet hřbitovů s daným názvem

3. Třídění dokumentů

   ```
   db.cemeteries.sort({
     "area": -1
   });
   ```

   Tento kód řadí dokumenty ze sbírky hřbitovů podle pole oblasti v sestupném pořadí. V důsledku toho budou vráceny dokumenty s největšími plochami.

4. Omezení vzorku

   ```
   db.cemeteries.limit(10);
   ```

   Tento kód vrací pouze prvních 10 dokumentů ze sbírky hřbitovů.

Toto je jen pár příkladů agregačních funkcí, je jich mnohem více a nemá smysl rozebírat všechny funkce.

Vkládání a aktualizace dat

Zde jsou základní funkce pro záznam a načítání dat.

1. Vložení jedné hodnoty

```
db.cemeteries.insertOne({
  "name": "Nové hřbitovy",
  "ogcfid": 12,
  "datum_exportu": "2023-08-17T00:00:00+00:00",
  "GlobalID": "CA784074-3E2D-4D2E-AE3B-778934385FA4",
  "SHAPE_Length": 1086.471828,
  "SHAPE_Area": 671921.9,
});
```

2. Vkládání více hodnot

```
const cemeteries = [
  {
    "name": "Nové hřbitovy",
    "ogcfid": 12,
    "datum_exportu": "2023-08-17T00:00:00+00:00",
    "GlobalID": "CA784074-3E2D-4D2E-AE3B-778934385FA4",
    "SHAPE_Length": 1086.471828,
    "SHAPE_Area": 671921.9,
  },
  {
    "name": "Návesní hřbitov",
    "ogcfid": 13,
    "datum_exportu": "2023-08-17T00:00:00+00:00",
    "GlobalID": "CA784074-3E2D-4D2E-AE3B-778934385FA5",
    "SHAPE_Length": 1086.471828,
    "SHAPE_Area": 671921.9,
  }
];

db.cemeteries.insertMany(cemeteries);
```

3. Aktualizace dokumentu

```
db.cemeteries.updateOne({
  "name": "Líšeň"
}, {
  $set: {
    "name": "Nové Líšeň"
```

```
    }
  });
```

Komplexnější použití agregačních funkcí

V MongoDB se agregační operace používají k provádění složitých výpočtů na sadách dokumentů. Umožňují provádět operace, jako je seskupování, řazení, filtrování a výpočet agregačních funkcí.

Agregační operace se provádějí pomocí metody `aggregate()` kolekce. Metoda `aggregate()` bere jako argument pole agregačních operací. Každá agregovaná operace provádí určitou operaci na sadě dokumentů.

Zde je příklad použití agregovaných operací:

```
db.cemeteries.aggregate([
  {
    $match: {
      "name": "Líšeň"
    }
  },
  {
    $group: {
      "_id": {
        "name": "$name"
      },
      "count": {
        $sum: 1
      }
    }
  }
]);
```

Tento kód dělá následující:

1. Filtruje sadu dokumentů podle kritéria `name = "Líšeň"`.
2. Seskupuje sadu dokumentů podle pole názvu.
3. Spočítá počet dokumentů v každé skupině.

Výsledkem bude sada dokumentů, kde každý dokument obsahuje následující:

- `_id`: Název hřbitova
- `count`: Počet hřbitovů s daným názvem

Souhrnné operace mohou být velmi výkonné a lze je použít k provádění složitých výpočtů na sadách dokumentů.

Import dat

Algoritmus lze popsat následujícím pseudokódem:

```
// načtení vstupu
zvolení vhodné distribuce dat;
načíst data z distribuce;
zkontrolovat integritu dat;

// připravení databáze
vytvořit kolekci pro data;
definovat vhodně klíč záznamů;

// načtení data
foreach záznam do
    if záznam v databázi existuje then
        aktualizovat záznam v databázi;
    else
        vložit záznam do databáze;
    end
end
```

Algoritmus 2: Načtení dat

Příklad skriptu v JavaScript:

```
// Import potřebných knihoven
const MongoClient = require("mongodb");
const fs = require("fs");

// Vytvořte připojení k MongoDB
const client = new MongoClient("mongodb://localhost:27017");

// Připojení k databázi
client.connect((err, db) => {
  // Zkontrolujte připojení
  if (err) {
    console.error(err);
    return;
  }

  // Vytvořte kolekci
  const collection = db.collection("cemeteries");

  // Čtení dat ze souboru GeoJSON
  const data = fs.readFileSync("cemeteries.geojson", "utf8");
  const features = JSON.parse(data).features;

  // Import dat do MongoDB
```

```
  features.forEach((feature) => {
    // Vytvořte dokument pro MongoDB
    const document = {
      name: feature.properties.name,
      ogcfid: feature.properties.ogcfid,
      datum_exportu: feature.properties.datum_exportu,
      GlobalID: feature.properties.GlobalID,
      SHAPE_Length: feature.properties.SHAPE_Length,
      SHAPE_Area: feature.properties.SHAPE_Area,
      boundary: feature.geometry.coordinates,
    };

    // Vložení nebo aktualizace dokumentu
    const result = collection.updateOne({ogcfid: document.ogcfid}, {
      $set: document,
    }, {upsert: true});

    // Vytiskni výsledek
    console.log(result);
  });

  // Uzavřete připojení
  client.close();
});
```

Grafová databáze

Zvolená datová sada: *BRNO BRZO: Development projects and plans*[3]

Vybraná distribuce z katalogu

Byl zvolen CSV formát jelikož nebyli využité prvky jiných formátů, např. geografická data.

Zvolená databáze a důvod

Zvolená databáze: *Neo4J*

Datová sada obsahuje seznam stavebních projektů v Brně, kde každý řádek reprezentuje jednotlivý projekt s informacemi o developerovi, studiu nebo firmy, typu investice, stavu stavby a jiné. Na základě toho můžeme považovat datovou sadu za heterogenní, co je možné využít v Neo4J. Neo4J nám umožňuje zobrazit různé entity na uzly spojené různymi vztahy.

Jedním z problémů dané datové sady je nepravidelná frekvence aktualizací. Největší problém datové sady spočívá v chybějicích datech v některých polech, toto vede k agregaci více projektů do jednoho a ve výsledku to snižuje kvalitu datové sady.

Jedno z pozitiv Neo4J je schopnost škálování[4] pro větší datové sady. Neo4J používa techniku zvanou "*sharding*", která umožňuje rozdelit databázi na menší ("*shards*"). Kromě toho je možné jednotlivé dotazy adaptovat umožňujíc extrakci jen nutných dat při vykonávaní dotazů.

Definice úložiště

Následuje definice v *Cypher*. Je lze si v ní povšimnout použití funkce `COALESCE`, která umožňuje nahrazení chybějicích dat nějakou smysluplnou hodnotou.

[3] https://data.gov.cz/datov%C3%A1-sada?iri=https%3A%2F%2Fdata.gov.cz%2Fzdroj%2Fdatov%C3%A9-sady%2F44992785%2Facb846315eaed4fba2473b12903b3267

[4] https://neo4j.com/product/neo4j-graph-database/scalability/#:~:text=Neo4j%20scales%20out%20as%20data,for%20a%20very%20large%20graph.

```
MERGE (p:Project {id: row.globalid}) ON CREATE SET
        p.name                  = row.nazev_projektu,
        p.address               = row.adresa,
        p.district              = row.mestska_cast,
        p.description           = row.poznamky
MERGE (
  d:Developer {name: COALESCE(row.developer_investor, "Neuvedeno")}
)
MERGE (
  a:Architect {name: COALESCE(row.architekt, "Neuvedeno")}
)

// Define relationships between nodes
FOREACH (
  _ in CASE row.stav WHEN "planovany" THEN [1] ELSE [] END
  | MERGE (p)-[:PLANNED]->(a)
)
FOREACH (
  _ in CASE row.stav WHEN "v realizaci" THEN [1] ELSE [] END
  | MERGE (p)-[:IN_PROGRESS]->(a)
)
FOREACH (
  _ in CASE row.stav WHEN "dokonceny" THEN [1] ELSE [] END
  | MERGE (p)-[:COMPLETED]->(a)
)
FOREACH (
  _ in CASE row.typ_investice WHEN "soukroma" THEN [1] ELSE [] END
  | MERGE (d)-[:INVESTED_PRIVATLY]->(p)
)
FOREACH (
  _ in CASE row.typ_investice WHEN "verejna" THEN [1] ELSE [] END
  | MERGE (d)-[:INVESTED_PUBLICLY]->(p)
)
FOREACH (
  _ in CASE row.typ_investice
    WHEN "Strategicky_projekt_mesta_Brna" THEN [1] ELSE [] END
  | MERGE (d)-[:INVESTED_STRATEGICALLY]->(p)
)

MERGE (p)-[:DEVELOPED_BY]->(d)
MERGE (p)-[:ARCHITECTED_BY]->(a)
MERGE (d)-[:WORKED_WITH]->(a);
```

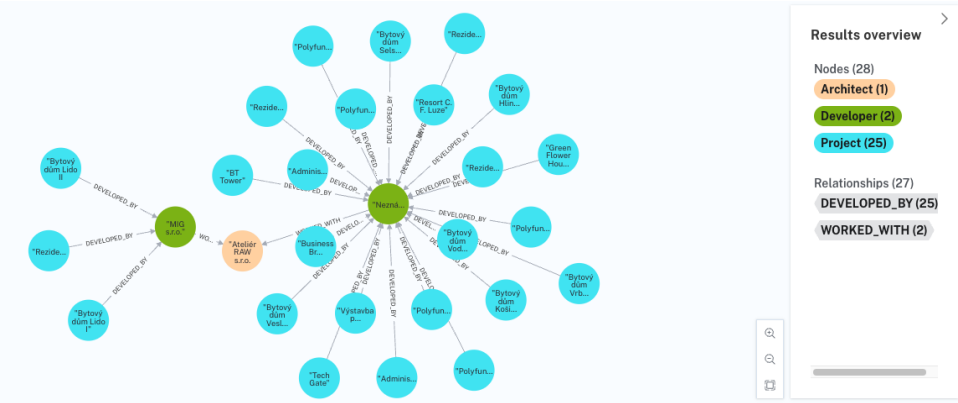
Příklady příkazů pro manipulaci s daty

1. Odstránění všech uzlů

   ```
   match (n) detach delete n;
   ```

2. Dotaz zobrazující vztahy architektů a developerů se společnými projekty

   ```
   match
       q=(a:Architect)<--(project)<-[:DEVELOPED_BY]-(developer)
       return q limit 25;
   ```



Obrázek 1: Výsledek dotazu

Import dat

K importování dat jsme použili následující nástroje:

- *Neo4j Aura DB*[5]
- *cypher-shell*[6]

Prvně byla vytvořená vzdálená instance Neo4J databáze s pomocí Aura DB. Následovalo navázání spojení mezi vzdálenou databází a lokálním zařízením pomocí cypher-shell s přihlašovacími údaji poskytnutými Aura DB (používatel, heslo a adresa běžící instance). Data byla nahrána pomocí dotazů v Cypher.

```
cypher-shell -a [link_to_instance] -u [username] -p [password]
```

[5] https://neo4j.com/cloud/platform/aura-graph-database/
[6] https://neo4j.com/docs/operations-manual/current/tools/cypher-shell/

Z důvodu zalomení byla `url`[7] nahrazená proměnnou.

```
LOAD CSV WITH HEADERS FROM url AS row
MERGE (p:Project {id: row.globalid}) ON CREATE SET
        p.name                  = row.nazev_projektu,
        p.address               = row.adresa,
        p.district              = row.mestska_cast,
        p.description           = row.poznamky
MERGE (d:Developer {name: COALESCE(row.developer_investor, "Neuvedeno")})
MERGE (a:Architect {name: COALESCE(row.architekt, "Neuvedeno")})

// Define relationships between nodes
FOREACH (_ in CASE row.stav WHEN "planovany" THEN [1] ELSE [] END |
        MERGE (p)-[:PLANNED]->(a)
)
FOREACH (_ in CASE row.stav WHEN "v realizaci" THEN [1] ELSE [] END |
        MERGE (p)-[:IN_PROGRESS]->(a)
)
FOREACH (_ in CASE row.stav WHEN "dokonceny" THEN [1] ELSE [] END |
        MERGE (p)-[:COMPLETED]->(a)
)
FOREACH (_ in CASE row.typ_investice WHEN "soukroma" THEN [1] ELSE [] END |
        MERGE (d)-[:INVESTED_PRIVATLY]->(p)
)
FOREACH (_ in CASE row.typ_investice WHEN "verejna" THEN [1] ELSE [] END |
        MERGE (d)-[:INVESTED_PUBLICLY]->(p)
)
FOREACH (
  _ in CASE row.typ_investice
    WHEN "Strategicky_projekt_mesta_Brna" THEN [1] ELSE [] END
  | MERGE (d)-[:INVESTED_STRATEGICALLY]->(p)
)

MERGE (p)-[:DEVELOPED_BY]->(d)
MERGE (p)-[:ARCHITECTED_BY]->(a)
MERGE (d)-[:WORKED_WITH]->(a);
```

[7] https://data.brno.cz/datasets/mestobrno::brno-brzo-stavebnÃŋ-projekty-a-zÃąmÄŻry-brno-brzo-develo
csv

Databáze časových řad

Zvolená datová sada: *Recorded temperatures in Bohumín*[8]

Vybraná distribuce z katalogu

Byl zvolen formát XML jelikož k disposici nebyly žádné jiné formáty.

Zvolená databáze a důvod

Zvolená databáze: *InfluxDB*

Tato datová sada obsahuje informace naměřených teplot v centru města Bohumín za jeden měsíc od 27. 9. 2023 do 27. 10. 2023. Data jsou reprezentována dátumem, časem a naměřenou teplotou. Interval měření je 5 minut umožňujíc přesnou analýzu v průběhu času.

V porovnání s předchozími datovými sadami je vidět, že jsou data homogenní a měří pouze jednou fyzikální veličinu. Z tohoto důvodu jsme se rozhodli použít InfluxDB, jelikož je přímo přizpůsobena[9] na tenhle účel. Kromě jiného není nutné definovat komplikované schémy protože data pozostávají pouze z *časové známky* a teploty, a jsou taky úplná (nechybí žádné hodnoty).

Jednotlivé záznamy nelze jednoznačně identifikovat, kromě data a času, které nespokytují žádnou přidanou hodnotu. Vhodné řešení by spočívalo v přidání identifikátorů nebo jmen zařízení, které měřili teplotu. Umožnilo by to více transparentní pohled na data k další analýze a porovnání mezi různými senzory v čase.

Dotazy probíhají nad časovými rozsahy a jsou omezeny na jeden měsíc, takže nevyžadují moc výpočetní síly. V případě větších časových rozsahů je lze použít *nepřetržité dotazy*[10], které běží periodicky nad daty dokuď dotaz není ukončen. Výsledek je tímhle způsobem vyhodnocen inkrementálně[11].

[8] https://www.mesto-bohumin.cz/export/teplota.xml.php

[9] https://docs.influxdata.com/influxdb/cloud-serverless/write-data/best-practices/schema-design/#design-for-performance

[10] https://docs.influxdata.com/influxdb/v1/query_language/continuous_queries/

[11] https://www.dbta.com/BigDataQuarterly/Articles/What-Is-Continuous-SQL-and-Why-Are-Companies-Chall

Definice úložiště

Pro vytvoření schématu byla použita knižnice `@influxdata` v JavaScriptu s následujícími balíčky:

- `@influxdata/influxdb-client`
- `@influxdata/influxdb-client-apis`

Prvním krokem je vytvoření *bucket* pro existující organizaci.

```
async postBucket(orgID: string, name: string) {
    const bucketsAPI = new BucketsAPI(this._influxDB);
    const bucket = await bucketsAPI.postBuckets({ body: { orgID, name } });
    this.log(
      JSON.stringify(
        bucket,
        (key, value) => (key === "links" ? undefined : value),
        2,
      ),
    );
  };
```

Po vytvoření je možné do něj ukládat data v rozsahu, který je udržován (starší data jsou zahozena). Každý řádek reprezentuje jeden bod. Úryvek z dokumentace InfluxDB[12] k bodům:

> A point represents a single data record that has four components: a measurement, tag set, field set, and a timestamp. A point is uniquely identified by its series and timestamp.

```
temperatures.forEach((t) => {
  const value = parseFloat(t.hodnota[0]);
  const timestamp = new Date(`${t.datum} ${t.cas}`).getTime();
  const point = new Point("temperature")
    .tag("sensor", "Bohumin")
    .floatField("value", value)
    .timestamp(timestamp);

  writeApi.writePoint(point);
});
```

aspx#:~:text=With%20traditional%20SQL%2C%20queries%20are,efficiently%20recomputing%20the%20current%20state

[12] https://docs.influxdata.com/influxdb/v1/concepts/key_concepts/#:~:text=Understanding%20the%20concept%20of%20a,by%20its%20series%20and%20timestamp.

Příklady příkazů pro manipulaci s daty

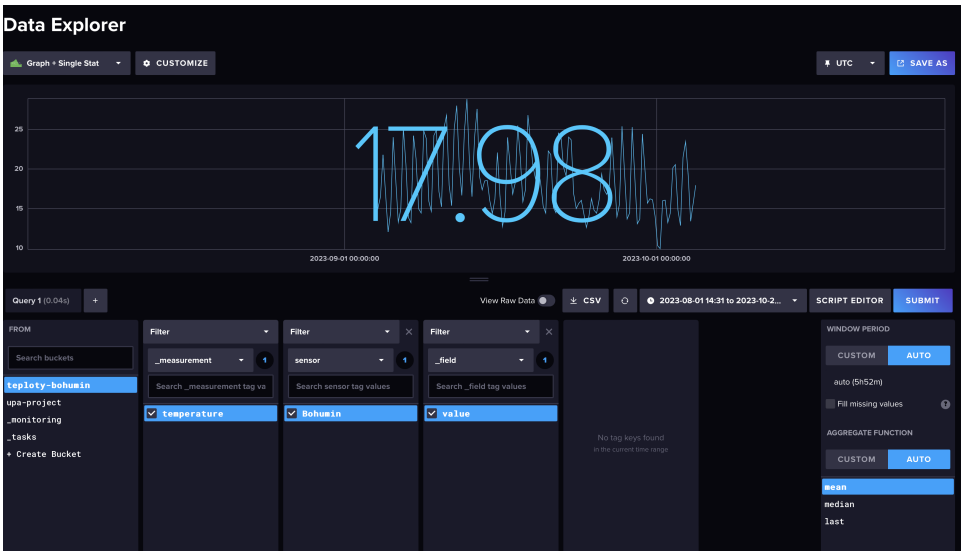
Jako příklad uvedeme nalezení průměrné teploty za 11 hodin v Bohumínu.
V terminálu lze použít následující dotaz:

```
from(bucket:${process.env.BUCKET})
    |> range(start: 2023-09-14T12:00:00.000Z, stop: 2023-09-14T23:00:00.000Z)
    |> filter(fn: (r) =>
        r._measurement == "temperature" and r["sensor"] == "Bohumin"
       )
    |> mean()
```

Pro tento dotaz dostaneme následující výstup:

```
[+] Querying data with from(bucket:"teploty-bohumin")
  |> range(start: 2023-09-14T12:00:00.000Z, stop: 2023-09-14T23:00:00.000Z)
  |> filter(fn: (r) =>
       r._measurement == "temperature" and r["sensor"] == "Bohumin"
     )
  |> mean()
{
  result: "_result",
  table: 0,
  _start: "2023-09-14T12:00:00Z",
  _stop: "2023-09-14T23:00:00Z",
  _value: 17.062878787878795,
  _field: "value",
  _measurement: "temperature",
  sensor: "Bohumin"
}
[+] Query completed
```

Následuje ukázka použití InfluxDB konzole:



Obrázek 2: Graf teplot za celý měsíc se zobrazeným průměrem

Import dat

Data jsou podána jako XML soubor, který je pak spracován přes Bun (JavaScript interpreter) s pomocí XML parseru. Po převedení dat do JSON formátu jsou podána do *Influx Client*, kterého implementace zapisuje data, následovně:

```
async writeData(bucket: string, data: Temperature) {
    const writeApi = this._influxDB.getWriteApi(this.org, bucket, "ms");
    const temperatures = data.teploty.teplota;

    this.log('Writing data...');

    temperatures.forEach((t) => {
      const value = parseFloat(t.hodnota[0]);
      const timestamp = new Date(`${t.datum} ${t.cas}`).getTime();
      const point = new Point("temperature")
        .tag("sensor", "Bohumin")
        .floatField("value", value)
        .timestamp(timestamp);

      writeApi.writePoint(point);
    });

    try {
      await writeApi.close();
```

```
  } catch (e) {
    if (e instanceof HttpError && e.statusCode === 401) {
      this.log("Set up a new InfluxDB database");
    }
  }

  this.log('Writing completed.');
}
```