UPA projekt, 1. časť: ukladanie rozsiahlych dát v NoSQL databázach

Bc. Ivan Halomi, xhalom00@stud.fit.vutbr.cz
Bc. Peter Koprda, xkoprd00@stud.fit.vutbr.cz
Bc. Adam Múdry, xmudry01@stud.fit.vutbr.cz

25. októbra 2022

Obsah

Časť I

Analýza zdrojových dát a návrh ich uloženia v NoSQL databáze

Kapitola 1

Analýza zdrojových dát

Cestovné poriadky verejnej dopravy je možné získať zo stránok Ministerstva dopravy Českej republiky[1]. Na týchto stránkach sa dá nájsť aj popis týchto dát, ktoré hovoria o tom, aký formát dát používajú jednotlivé súbory. Súbory sú rozdelené do hlavných zložiek, kde jedna zložka obsahuje dáta pre obdobie od 1. decembra predchádzajúceho roka do 30. novembra aktuálneho roka. Hlavný súbor v tejto zložke obsahuje najdôležitejšie dáta na spracovanie a je komprimovaný pomocou Zip. Nachádzajú sa v ňom súbory vo formáte `XML`, kde každý súbor obsahuje jednu správu `CZPTTCISMessage`. Okrem tohto súboru sa nachádzajú aj zložky pre každý mesiac. Súbory v týchto zložkách majú koncovku `.zip`, ale sú komprimované pomocou Zip alebo pomocou GZip. Súbory v týchto archívoch obsahujú ďalšie správy `CZPTTCISMessage`, ale okrem toho obsahujú správy `CZCanceledPTTMessage`, ktoré poskytujú dáta o odrieknutých dňoch cestovných poriadkoch vlaku osobnej dopravy. Detailný popis štruktúry XML dokumentov je popísaný v PDF dokumente dostupný na webstránke dopravcu[2].

Správy `CZPTTCISMessage`

Jednotlivé správy `CZPTTCISMessage` obsahujú dátové informácie o cestovných poriadkoch vlaku osobnej dopravy. Je rozdelená na 5 častí:

1. `Identifiers` – pole identifikátorov, ktoré obsahuje:
   - 0 až N elementov `PlannedTransportIdentifiers`
   - 0 až N elementov `RelatedPlannedTransportIdentifiers`

   Identifikátory uvedené v obidvoch elementoch majú zhodnú štruktúru.
2. `CZPTTCreation` – určuje dátum a čas vytvorenia `CZPTT`

[1] `https://portal.cisjr.cz/pub/draha/celostatni/szdc/`

[2] `https://portal.cisjr.cz/pub/draha/celostatni/szdc/Popis%20DJ%C5%98_CIS_v1_09.pdf`

3. `CZPTTHeader` – obsahuje 2 elementy popisujúce štartovný a cieľový bod v zahraničí:
   - `CZForeignOriginLocation` – uvádza sa, ak spoj vychádza zo zahraničnej lokality
   - `CZForeignDestinationLocation` – uvádza sa, ak spoj končí v zahraničnej lokalite
4. `CZPTTInformation` – najdôležitejší informačný element, udáva údaje o lokalite trasy, má definované minimálne 2 lokality – `CZPTTLocation` – obsahuje viacero elementov, ktoré definujú presnú lokalitu a kedy bude daný spoj v danej lokalite (element `TimingAtLocation`)
5. `NetworkSpecificParameters` – element obsahujúci národné parametre, tento element je nepovinný

Správy `CZCanceledPTTMessage`

Jednotlivé správy `CZCanceledPTTMessage` obsahuje dátové informácie o odrieknutých dňoch cestovného poriadku vlaku osobnej dopravy. Tieto správy sú rozdelené na 4 časti:

1. `PlannedTransportIdentifiers` – obsahuje pole identifikátorov, každý element obsahuje identifikáciu cestovného poriadku a vlaku
2. `CZPTTCancelation` – určuje dátum a čas odrieknutia `CZPTT`
3. `PlannedCalendar` – obsahuje dni, kedy je daný `CZPTT` odrieknutý
4. `CZDeactivatedSection` – popisuje úsek, kedy došlo k odrieknutiu cestovného poriadku, nepovinný element

Kapitola 2

Návrh spôsobu uloženia dát

Vytvorenie databázy

Na základe analýzy dát sme zistili, že vhodným spôsobom uloženia dát bude pomocou dvoch kolekcií, ktoré budú ukladané do jednej databázy. V jednej kolekcii budú ukladané dáta o cestovných poriadkoch a v druhej kolekcii budú ukladané opravy prvej kolekcie.

Ukladanie do databázy

Keďže každý súbor obsahuje jednu správu, je možné tieto správy identifikovať pomocou názvu súboru. To znamená, že každej správe môžme priradiť do premennej `_id` názov tohto súboru bez koncovky `.xml`. Opravné súbory majú ešte príponu podľa názvu zložky daného mesiaca. Napríklad pre súbor `PA_0054_KT--61917A_00_2022.xml` je `_id`: `PA_0054_KT--61917A_00_2022`. Celý súbor sme previedli do asociatívneho poľa a pridali sme k tomu túto hodnotu `_id`.

Kapitola 3

Zvolená NoSQL databáza

Zvolili sme NoSQL databázu MongoDB, ktorá dáta ukladá ako štruktúrované dokumenty vo formáte BSON, čo je binárna reprezentácia formátu JSON. Dáta sú zlúčené do kolekcii, ktoré sú ekvivalent tabuľky systému riadenia báze dát (SRBD) relačnej databázy. Pôvodné dáta sú vo formáte XML, ktorý je taktiež štruktúrovaný a prevediteľný na formát JSON alebo typ premennej "slovník"(asociatívne pole) v Pythone s ktorými vieme jednoducho pracovať. MongoDB ponúka rôznu funkcionalitu, ako napríklad vytváranie indexov na zrýchlenie vyhľadávania určitých dát alebo špecifikovanie pravdiel jazyka pri porovnávaní reťazcov pri vyhľadávaní v dátach.

Časť II

Návrh, implementácia a použitie aplikácie

Kapitola 4

Návrh aplikácie

Získavanie dát

Aplikácia by pri prvom spustení mala stiahnuť všetky dáta a vložiť ich do databázy. Tieto dáta sú voľne dostupné na stránke[1]. Dáta sa na stranke nachádzajú v dvoch formách. Prvou, tou podstatnejšou, je súbor GVD2022 vo formáte .zip, ktorý obsahuje všetky naplánované spoje na celý rok. Ďalej sa tam nachádzajú úpravy, zmeny a náhrady spojov počas roka. Tieto súbory sú rozdelené do adresárov podľa mesiacov a sú na stránku nahrávané počas roka. Súbory sú vo formáte .xml.zip, avšak v skutočnosti ide o formát .xml.gzip. Od aplikácie sa očakáva, že stiahne všetky tieto dáta, spracuje ich do vhodného formátu, nahrá spoje do databázy a následne aplikuje na ne získané úpravy a náhrady spojov.

Aplikovanie zmien

Ako bolo spomenuté železničná spoločnosť pravidelne vydáva zmeny cestovných poriadkov, zväčša ide o rušenie spojov, prípadne ich náhrady. Súbory sú teda rozdelené na dva formáty. Tie čo spoje len rušia a tie čo vytvárajú náhradné spoje. Náhradný spoj môže, ale aj nemusí rušiť nejaký už existujúci spoj. Zároveň je možné rušiť aj náhradné spoje. Aplikácia by mala byť schopná tieto zmeny aplikovať na existujúce dáta a uchovať si informácie o ich zmene. Zároveň by nemala vytvárať nové záznamy o upravených spojov, a teda bude len upravovať záznamy v databáze. Ak názov súboru začína slovom *cancel* jedná sa o typ správy **CZCanceledPTTMessage**, táto správa má v **PlannedTransportIdentifiers.Core** uložený spoj, ktorý upravuje. Záznam tohto spoja bude potrebné načítať z databázy a upraviť jeho **CZPTTInformation.PlannedCalendar.BitmapDays** tak aby odpovedal zmene. Cancel správu nie je vhodné ukladať k existujúcim spojom a preto je treba pre ňu vytvoriť novú kolekciu. Toto isté platí pre druhý typ správy a to **CZPTTCISMessage**. V tomto prípade môžeme uvažovať, že ide o klasický spoj, avšak s tým rozdielom, že ak existuje **CZPTTCISMessage.RelatedPlannedTransportIdentifiers** tak vlak nahrádza niektorý

[1] `https://portal.cisjr.cz/pub/draha/celostatni/szdc/2022/`

existujúci záznam v danom dátume. V podstate sa teda jedná o dve správy v jednej. Preto tento záznam je potrebné uložiť do kolekcie k existujúcim spojom. Zmena **BitmapDays** by mala prebehnúť podľa nasledovnej logiky:

1. Náhradný spoj/cancel správa obsahuje v danom dni „1“ -> znak v pôvodnom spoji nastavíme na „0“
2. Náhradný spoj/cancel správa obsahuje v danom dni „0“ -> znak v pôvodnom spoji ostáva taký ako bol
3. Situáciu kedy cancel správa/náhradný spoj má rozsah platnosti mimo platnosť pôvodného spoja ignorujeme

Vyhľadávanie spojov

Aplikácia by mala byť schopná vyhľadať vlakové spojenie pomocou zadaných informácií. Tieto informácie sú: počiatočná stanica, cieľová stanica, dátum a čas. Zobrazené by mali byť spoje, ktoré idú vo zvolený dátum z počiatočnej stanice po zvolenom čase a zastavujú v cieľovej stanici. Zobrazujeme názov stanice a čas odchodu vlaku, z počiatočnej stanice, cieľovej stanice a takisto všetkých staníc, v ktorých vlak zastavuje medzi nimi. Potrebné informácie, ktoré potrebujeme získať z databázy: **názov stanice, čas odchodu zo stanice, akciu vlaku v danej stanici, bitmapu s dňami kedy vlak jazdí, platnosť tejto bitmapy**.

Zvolená MongoDB umožňuje vytvárať query pomocou agregačných pipelines, tie slúžia na spracovanie veľkého množstva „dokumentov“ v kolekcii pomocou postupného filtrovania a posúvania ich do ďalšej fázy. Agregačná pipeline pre najrýchlejšie vyhľadanie bude fungovať nasledovne:

1. Odfiltrovanie neplatných dát v daný dátum
2. Odfiltrovanie ciest, ktoré neobsahujú počiatočnú a cieľovú stanicu
3. Odfiltrovanie ciest, ktoré sú v počiatočnej stanici neskôr ako v zvolený čas

Tento výsledok obsahuje všetky spoje, ktoré sú platné v daný dátum a prechádzajú(nie nutne stoja) hľadanými stanicami v oboch smeroch. Preto je ešte potrebné skontrolovať či vlak stojí v hľadaných staniciach. A taktiež pomocou bitmapy či vlak skutočne v daný deň danú trasu ide. Tieto informácie však nie je vhodné kontrolovať v agregačnej pipeline z dôvodu zložitosti. Preto sa o to bude starať aplikácia. Ako prvé skontrolujeme či spoj v daný spoj premáva. Teda v bitmape je na **n-tom** mieste nastavený znak „1“. Číslo **n** získame ako rozdiel v dňoch medzi začiatkom platnosti a hľadaným dátumom. Ak spoj nepremáva, ďalej nepokračujeme a ideme na ďalší vrátený spoj. Následovne budeme hľadať počiatočnú a cieľovú stanicu. Do úvahy budeme brať len stanice, ktoré obsahujú slovník „TranActivity“ (môže priamo obsahovať slovník s kľúčom „TrainActivityType“ alebo pole slovníkov obsahujúcich „TrainActivityType“ v prípade viacerých akcií v jednej stanici). Pomocou neho zistíme, či vlak v stanici stojí pre nástup a výstup cestujúcich. Ak je stanica počiatočná, vlak v nej

stojí a zároveň ešte nebola nájdená cieľová stanica, teda ideme dobrým smerom, začneme konkatenovať do výsledného textu nasledovné informácie: názov stanice, čas odchodu vlaku z danej stanice až pokým nenájdeme cieľovú stanicu. Ak v cieľovej stanici taktiež existuje akcia na nástup a výstup cestujúcich, výsledný text zobrazíme na výstup. V opačnom prípade pokračujeme v kontrole ďalšieho spoja, ak nejaký ešte je.

Kapitola 5

Implementácia

Získanie dát

Všetko potrebné k stiahnutiu dát sa nachádza v súbore `download_data.py`. Ide o funkciu `download_page()`, ktorá získa obsah stránky pomocou `requestu` z knižnice `urllib`. Obsah stránky sa následne spracuje pomocou knižnice `BeautifulSoup`. Získajú sa všetky odkazy, ktoré sa buď stiahnu, alebo ak ide o odkaz na podstránku pre jednotlivé mesiace, pošlú sa do funkcie `get_subpages(link: str, month: str)`. Tu je postup takmer identický, s tým rozdielom, že už sa jedná len o súbory, ktoré jednoducho stiahneme.

Nahranie do databázy

Stiahnuté dáta preiterujeme a prečítame pomocou knižníc `zipfile` a `gzip`, aby sme ich nemuseli rozbaľovať na disk. Na komunikáciu s databázou využívame knižnicu `pymongo`. Extrahované dáta sú do databázy nahrané pomocou funkcie `save_data_to_db()`. Ak dáta už v databáze sú, sú len prepísané novými hodnotami. Ak nie, tak sú tam vložené. Databázu teda nie je potrebné pri zmenách celú mazať.

Aplikovanie fixov

Aplikovanie „fixov“ prebieha pomocou funkcie `save_fixes_to_db()`. Súbory preiterujeme a prečítame v opačnom abecednom poradí, keďže náhradné spoje musia existovať aby neskôr mohli byť zrušené. Toto by v abecednom poradí spracovania súborov nebolo možné v jednom prechode. Po spracovaní dát, načítame z databáze záznam, ktorý treba upraviť pomocou `coll.find_one()`,kde `coll` je odkaz na kolekciu v databáze so záznamami o spojoch získaný pomocou knižnice `pymongo`. V tomto zázname potrebujeme upraviť **CZPTTInformation.PlannedCalendar.BitmapDays**, podľa algoritmu popísanom v návrhu4. Správne rozmedzie získame odčítaním počiatočného dátumu platnosti spoja od dátumu začiatku platnosti zmeny. Následne iterujeme cez BitmapDays cyklom s počtom dní zmeny, alebo do konca platnosti pôvodného spoja. Následne zmenený

záznam spoja uložíme pomocou `coll.update_one()` a správu o zrušení spoja alebo náhradný spoj uložíme do správnej kolekcie pomocou `replace_one()`.

Vyhľadávanie spojov

Na vyhľadávanie spojov slúži funkcia `find_road(departure: str, target: str, start_datetime: str)` v súbore `find_data.py`. Funkcia prijíma ako parametre:

- názov stanice, v ktorej má spoj začať, resp. má cez ňu prechádzať a povoľovať nástup a výstup pasažierov,
- cieľovú stanicu v ktorej vlak taktiež musí zastaviť a
- dátum a čas, po ktorom daný spoj vyráža zo začiatočnej stanice

Uvažujeme, že zobrazujeme všetky spoje premávajúce v daný dátum po danej hodine. Na samotné vyhľadávanie využívame nasledujúcu agregačnú pipeline:

```
cursor = coll.aggregate([
  {'$match': {  # Departure and target station are together in the same document
    '$and': [
      {'CZPTTCISMessage.CZPTTInformation.CZPTTLocation.Location.PrimaryLocationName':
        departure},
      {'CZPTTCISMessage.CZPTTInformation.CZPTTLocation.Location.PrimaryLocationName':
        target}],
      'CZPTTCISMessage.CZPTTInformation.PlannedCalendar.ValidityPeriod.StartDateTime':
        {'$lte': start_datetime},
      'CZPTTCISMessage.CZPTTInformation.PlannedCalendar.ValidityPeriod.EndDateTime':
        {'$gte': start_datetime},
      'CZPTTCISMessage.CZPTTInformation.CZPTTLocation.TimingAtLocation.Timing.Time':
        {'$gte': start_time},
  }},
  {'$project': {  # Output only these fields (get rid of unnecessary fields)
    'CZPTTCISMessage.CZPTTInformation.PlannedCalendar.ValidityPeriod.StartDateTime': 1,
    'CZPTTCISMessage.CZPTTInformation.PlannedCalendar.ValidityPeriod.EndDateTime': 1,
    'CZPTTCISMessage.CZPTTInformation.PlannedCalendar.BitmapDays': 1,
    'CZPTTCISMessage.CZPTTInformation.CZPTTLocation.Location.PrimaryLocationName': 1,
    'CZPTTCISMessage.CZPTTInformation.CZPTTLocation.TimingAtLocation': 1,
    'CZPTTCISMessage.CZPTTInformation.CZPTTLocation.TrainActivity': 1,
  }},
  {'$group': {  # Create a new group structe from aggregation with only relevant data
    '_id': '$_id',
    'locations': {'$addToSet': {
      'names':
        '$CZPTTCISMessage.CZPTTInformation.CZPTTLocation.Location.PrimaryLocationName',
      'times':
        '$CZPTTCISMessage.CZPTTInformation.CZPTTLocation.TimingAtLocation.Timing.Time'
    }},
    'calendar': {'$addToSet': {
      'valid_start_datetime':
        '$CZPTTCISMessage.CZPTTInformation.PlannedCalendar.ValidityPeriod.StartDateTime',
      'valid_end_datetime':
        '$CZPTTCISMessage.CZPTTInformation.PlannedCalendar.ValidityPeriod.EndDateTime',
      'bitmap': '$CZPTTCISMessage.CZPTTInformation.PlannedCalendar.BitmapDays'
    }},
    'activity': {'$addToSet': {
      'type': '$CZPTTCISMessage.CZPTTInformation.CZPTTLocation.TrainActivity'
    }},
  }}])
```

Výsledky vrátené databázou musíme ešte prefiltrovať, aby sme odstránili spoje, ktoré nestoja v daných staniciach alebo v daný dátum nepremávajú. Tak isto treba odstrániť spoje opačným smerom (cieľová stanica skôr ako počiatočná). Na toto filtrovanie využívame len jednoduché prechádzanie slovníkov a pár podmienok. Pre zistenie či daný vlak v daný dátum ide, musíme získať **n-tý** symbol v „BitmapDays“ a porovnať, či sa rovná *'1'*. Kde číslo **n** je rozdielom dátumov. Keďže však dátumy uchovávame ako *string*, je potrebné ho previesť na typ *date*. Na tento prevod využívame funkciu `strptime()` z knižnice *datetime*. Následne je možné dátumy od seba jednoducho odčítať, keďže vďaka pipeline vieme, že dátum je väčší ako začiatok platnosti dát a menší ako koniec platnosti, a výsledkom je **n**. Potom kontrolujeme stanice na trase, ak v stanici nie je žiadna „TrainActivity“, pokračujeme ďalšou stanicou, keďže aj keby bola daná stanica hľadaná, tak tam vlak nezastavuje a tým pádom nás nezaujíma. Ak „TrainActivity“ existuje, tak porovnávame názov stanice s počiatočnou alebo cieľovou. Ak nájdeme cieľovú stanicu, a zároveň počiatočná ešte nebola nájdená, ide o spoj opačným smerom a pokračujeme na ďalší spoj. Ak nájdeme počiatočnú stanicu, začíname si ukladať názvy staníc a odchod vlaku z nich do *result_text*. Ak nájdeme cieľovú stanicu a je v nej definovaná „TrainActivity“ *result_text* vypíšeme. Ak v cieľovej stanici nie je definovaná „TrainActivity“, text nikdy nevypíšeme.

Toto je vnútorná štruktúra dostupná po 2. fáze pipeline `project` *coll.aggregate()*:

```
"CZPTTCISMessage": {
  "CZPTTInformation": {
    "CZPTTLocation": [{
      "Location": {
        "PrimaryLocationName": Nazov stanice,
      },
      "TimingAtLocation": {
        "Timing": [{
          "Time": Cas odjazdu vlaku,
        }]
      },
      "TrainActivity": [{
        "TrainActivityType:" Kod nastupu ludi- "0001"
      }]
    }],
    "PlannedCalendar": {
      "BitmapDays": "1111100...111",
      "ValidityPeriod": {
        "StartDateTime": "2021-12-13T00:00:00",
        "EndDateTime": "2022-12-09T00:00:00"
      }
    }
  }
}
```

S nad touto štruktúrov potom prebehne 3.fáza pipeline `group`, ktorá ju zjednoduší a premenuje jednotlivé kľúče hodnôt na ľahšie používanie pri vyhľadávaní spojov.

Kapitola 6

Spôsob používania

Program je napísaný v jazyku Python a potrebné knižnice vypísané sú v súbore `requirements.txt`. Nainštalovať je ich možné prostredníctvom programu `pip`:

```
$ python -m pip install -r requirements.txt
```

Príkaz `python main.py -h` zobrazí pomocnú správu s popisom argumentov programu. Pred prvým spustením je potrebné spustiť MongoDB databázu a definovať jej prihlasovacie údaje v súbore `process_data.py`.

Následne môžme stiahnuť dáta:

```
$ python main.py -d
```

A nahrať ich do databázy:

```
$ python main.py -s
```

Teraz program umožnuje prehľadávať databázu spojov. Napríklad:

```
$ python main.py --from STANICA --to STANICA \
  --date DATUM --time CAS
```

Ukážka použitia:

```
$ python main.py \
  --from Fulnek \
  --to "Suchdol nad Odrou" \
  --date 2021-12-21 --time 20:29:00
```

Kapitola 7

Experimenty

Nastavenie prostredia

Na testovanie programu pomocou experimentov sme používali pôvodne nakonfigurované prostredie, ktoré sme používali pri vývoji programu. Každú akciu sme merali pomocou príkazu `time`[1], ktorý vypíše dobu trvania programu. Rýchlosť stiahnutia dát z webových stránok pomocou tohto nástroja nebola meraná.

Parametre stroja, ktorý sme používali pri skúšaní experimentov:

- Operačný systém: Linux pop-os
- Architektúra: x86_64
- Lenovo Ideapad 720S-15IKB
- Procesor: Intel Core i5-7300HQ CPU @ 3.50GHz
- RAM: 16GiB

Databáza MongoDB bola spustená vo virtuálnom stroji s operačným systémom NixOS.

Pri ukladaní hodnôt do databázy sme namerali tieto hodnoty:

```
real    2m51.291s
user    1m8.504s
sys     0m1.618s
```

Čo znamená, že celkový reálny čas uloženia dát do databázy (dáta o cestovných poriadkoch a aj opravené dáta cestovných poriadkov) bolo skoro 3 minúty.

Po uložení dát do databázy sme mohli začať skúšať rýchlosť jednotlivých požiadavkov. Priemerná dĺžka spracovania požiadavku a vypísanie výsledku:

[1] `https://man7.org/linux/man-pages/man1/time.1.html`

```
$ time python main.py \
  --from Stachovice --to "Suchdol nad Odrou" \
  --date 2021-12-21 --time 08:43:31

             real    0m0.790s
             user    0m0.242s
             sys     0m0.049s
```