Vysoké učení technické v Brně
Fakulta informačních technologií

Ukládání a příprava dat – Projekt, 1. část

Ukládání rozsáhlých dat v NoSQL databázích

3. října 2023 Ondřej Ondryáš, František Nečas, David Chocholatý

Obsah

Část I

Analýza zdrojových dat a návrh jejich uložení v NoSQL databázi

Kapitola 1

Analýza zdrojových dat

Cílem projektu je zanalyzovat způsob publikování strojově zpracovatelných dat o jízdních řádech veřejné osobní vlakové dopravy v ČR, navrhnout řešení pro získávání a ukládání těchto dat za účelem implementace jednoduché aplikace pro vyhledávání přímých spojů mezi stanicemi.

Ministerstvo dopravy ČR zajišťuje vedení **Celostátního informačního systému o jízdních řádech** (CIS JŘ, CIS), prostřednictvím kterého zveřejňuje data o jízdních řádech pro potřeby veřejnosti [2]. Do CIS vkládá data o jízdních řádech vlaků osobní veřejné dopravy (JŘ) provozovatel železniční dráhy, Správa železnic, státní organizace (SŽ) [1]. Jízdní řády jsou do CIS uloženy v podobě komprimovaných **XML** souborů, ve kterých jsou serializovány objekty **datových zpráv** (dále také jen „zprávy“), které nesou informace o konkrétním jízdním řádu nebo jeho zrušení.

1.1 Získání dat

Pro konzumenty dat má CIS podobu vzdáleného úložiště, resp. vzdálené adresářové struktury, ve které jsou uloženy soubory s daty JŘ. K CIS a v něm uloženým JŘ je možné přistoupit:

- pomocí protokolu HTTPS, kdy na URL `https://portal.cisjr.cz/pub/draha/celostatni/szdc/` je publikována adresářová struktura ve formě jednoduchých HTML stránek;
- nebo pomocí protokolu FTP na URL `ftp://ftp.cisjr.cz/draha/celostatni/szdc/`, kde je dostupná tatáž adresářová struktura.

Dostupné soubory jsou organizovány následovně: Na výše uvedených lokacích je adresář obsahující dokumentaci [1]; a adresáře označující roky. Uvnitř každého z nich je dostupný soubor `GVDyyyy.zip` (kde `yyyy` je daný rok), což je archiv komprimovaných dat ve formátu ZIP obsahující datové zprávy s informacemi o všech jízdních řádech, které jsou naplánovány pro daný rok (dále také „bázová data“). Dostupné mohou být také další soubory ve formátu ZIP s prefixem `GVD`, které obsahují větší sady aktualizací pro bázová data. Dále jsou zde dostupné adresáře pojmenované ve tvaru `yyyy-mm` (kde `yyyy` je rok a `mm` je měsíc), do kterých jsou ukládány datové zprávy nesoucí informace o změnách oproti bázovým datům. Tyto zprávy jsou uloženy:

- buď jako soubory XML v kompresi **gzip**, tyto jsou rozpoznatelné podle nestandardní přípony `.xml.zip`;
- nebo jako archivy ZIP, které uvnitř obsahují jediný soubor XML, tyto jsou rozpoznatelné podle přípony `.zip`.

Nutno podotknout, že se nám nepovedlo dohledat žádný dokument, který by organizaci datových souborů JŘ v CIS definoval, výše uvedené tedy vyplývá z vlastní analýzy stávajících zde uložených souborů.

1.2 Vydávání dat

§ 42 zákona č. 266/1994 Sb., zákon o drahách, stanovuje: „Jízdní řád [...] nabývá platnosti druhou sobotu v prosinci o půlnoci. Doba platnosti jízdního řádu je 12 měsíců. Pravidelná změna platného jízdního řádu nabývá platnosti druhou sobotu v červnu o půlnoci nebo v jiném termínu uvedeném v prohlášení o dráze.“ Dá se tedy očekávat, že bázová data nebo jiné balíky větších změn se v CIS objevují **maximálně několikrát do roka**. V době realizace tohoto projektu (říjen 2022) byl k dispozici pouze balík `GVD2022.zip` vytvořený 23. listopadu 2021; a balík `GVD2022-oprava_poznamek_KJR_vybranych_tras20220126.zip` vytvořený 26. ledna 2022. Bázová data obsahují **12 334** zpráv. Doplňkový balík v tomto projektu neuvažujeme.

Datové zprávy odesílá SŽ do CIS **bezprostředně po přidělení trasy a kapacity** v jiném informačním systému [1, kap. 4]. Dále § 54 vyhlášky č. 173/1995 Sb. stanovuje, že provozovatel dráhy předá do CIS jízdní řád a jeho změny **nejpozději 7 dnů před jeho platností**.

Frekvenci publikování zpráv do CIS je možné odhadnout analýzou aktuálně dostupných souborů z průběhu roku. Od 3. 12. 2021[1] do 19. 10. 2022 bylo vydáno **28 585** zpráv (doplňujících bázová data). Jednoduchým bodovým odhadem očekávaného počtu zpráv za den je aritmetický průměr, který zde činí 28 585/319 $\doteq$ **90** zpráv/den. Na tento údaj je však nutné nahlížet s vědomím, že rozložení četnosti zpráv není rovnoměrné: např. v červnu bylo vydáno 5 142 zpráv, zatímco v lednu pouze 1 297, v červnu tedy přišlo průměrně **171** zpráv za den, zatímco v lednu jich bylo pouze **42**.

Zprávy jsou publikovány nahodile a neexistuje možnost, jak být notifikován o přidání nové zprávy. Konzument tedy musí **pravidelně kontrolovat**, zda nebyla do CIS uložena zpráva, kterou ještě nezpracoval[2].

1.3 Struktura dat

Dokumenty publikované v CIS obsahují datové zprávy serializované ve formátu XML. Dokumenty jsou uloženy v kódování UTF-8, řádky jsou ukončeny sekvencí CRLF. Každá zpráva je právě jednoho z typů:

- `CZPTTCISMessage` – obsahuje data jízdního řádu vlaku osobní dopravy;
- nebo `CZCanceledPTTMessage` – obsahuje data o odřeknutých dnech jiného jízdního řádu vlaku osobní dopravy.

Typ zprávy je zároveň kořenovým elementem XML dokumentu, který ji serializuje. Pro tyto XML dokumenty neexistuje formální popis (jako např. DTD nebo XSD), jejich strukturu však dostatečně popisuje dokumentace [1]. Ta také uvádí, že jsou zprávy podmnožinou blíže neurčených datových zpráv standardu TAF TSI[3], což je rozsáhlý mezinárodní standard vydávaný Agenturou Evropské unie pro železnice[4] (ERA) navržený pro kompletní výměnu

[1] Datum vydání nejstarší aktuálně dostupné zprávy.

[2] „*Zprávu CZPTTCISMessage pro každý CZPTT předává IS KADR do CIS. Veřejné subjekty si jej budou stahovat z FTP CIS.*“ [1, kap. 2.1] IS KADR je informační systém, který SŽ využívá pro řízení procesu přidělování kapacity a tras.

[3] `https://taf-jsg.info/?page_id=35`

[4] `https://www.era.europa.eu/`

informací o železniční dopravě.

Některé údaje uvedené v následujícím textu vycházejí z dat stažených z CIS **19. 10. 2022 v 19:00**. Tato data budou dále označována jako „stávající data“.

1.3.1 Zprávy typu `CZPTTCISMessage`

Zprávy s daty JŘ jsou strukturované datové typy. Následující vnořený seznam naznačuje pouze nejvýznamnější (v kontextu řešeného problému) části jejich struktury. Jednotlivé položky seznamu odpovídají názvu elementu XML, který danou položku popisuje; úroveň vnoření v seznamu odpovídá úrovni vnoření v datové struktuře. V závorce jsou uvedeny kardinality jednotlivých položek. Úplný popis je dostupný v [1] (kap. 3.1).

- `CZPTTCISMessage` – typ zprávy
  - `Identifiers` (1) – kolekce identifikátorů JŘ
    - `PlannedTransportIdentifiers` (2) – strukturovaný identifikátor (viz 1.3.3)
    - `RelatedPlannedTransportIdentifiers` (0..1) – strukturovaný identifikátor (viz 1.3.3)
  - `CZPTTCreation` (1) – datum a čas vytvoření datového JŘ
  - `CZPTTInformation` (1) – kolekce informačních údajů pro jednotlivé dopravní body (lokality) trasy JŘ
    - `CZPTTLocation` (2..*) – popis dopravního bodu (lokalitu)
      - `Location` (1) – identifikátory dopravního bodu (lokality), např. kód země, číselný kód nebo název
      - `TimingAtLocation` (0..1) – určení **času** vlaku v lokalitě
      - `TrainActivity` (0..*) – úkon vlaku v lokalitě (např. „nástup a výstup cestujících“ nebo „nečeká na žádné přípoje“)
    - `PlannedCalendar` (1) – popis dnů, ve kterých je JŘ platný (vlak jede)
      - `ValidityPeriod` (1) – datum počátku a konce platnosti JŘ
      - `BitmapDays` (1) – specifikuje, pro které dny z určeného intervalu platnosti tento JŘ skutečně platí

Nutno podotknout, že [1] poskytuje k některým položkám informace, které jsou **v konfliktu** s jinými informacemi tamtéž nebo se skutečnou strukturou stávajících dat:

- pro položku `PlannedTransportIdentifiers` specifikuje kardinalitu 0..*, ale v poznámce uvádí, že budou vždy uvedeny 2 identifikátory (to bylo nad stávajícími daty ověřeno);
- pro položku `RelatedPlannedTransportIdentifiers` specifikuje kardinalitu 0..*, ale vzhledem k tomu, že položka reprezentuje odkaz na právě jednu jinou zprávu, ve skutečnosti se může vyskytnout maximálně jednou (to bylo nad stávajícími daty ověřeno);
- pro položku `CZPTTInformation` specifikuje kardinalitu 1..*, ale neurčuje, jaký význam by opakování této položky mělo, v žádné ze stávajících zpráv se neopakuje;
- pro položku `TimingAtLocation` specifikuje kardinalitu 0..1, ale není jasné, jaký význam by vynechání této položky mělo, vyskytuje se ve všech stávajících zprávách (právě jednou v rámci jednoho elementu `CZPTTLocation`);
- pro položku `BitmapDays` specifikuje, že není povinná, ale v poznámce uvádí, že bude vždy povinně vyplněna.

1.3.2 Zprávy typu `CZCanceledPTTMessage`

Zprávy o odřeknutých dnech JŘ jsou strukturované datové typy. Ve dny uvedené v „kalendáři“ takové zprávy nepojede žádný vlak odkazovaného JŘ. Následující vnořený seznam naznačuje strukturu těchto zpráv (obdobně jako v 1.3.1). Úplný popis je dostupný v [1] (kap. 3.2).

- `CZCanceledPTTMessage` – typ zprávy
  - `PlannedTransportIdentifiers` (2) – položky kolekce strukturovaných identifikátorů (viz 1.3.3) odřeknutého JŘ
  - `CZPTTCancelation` (1) – datum a čas vytvoření datového JŘ
  - `PlannedCalendar` (1) – popis dnů, pro které je odřeknutí platné
    - `ValidityPeriod` (1) – datum počátku a konce platnosti odřeknutí
    - `BitmapDays` (1) – specifikuje, pro které dny z určeného intervalu platnosti odřeknutí skutečně platí

Ačkoliv použitá datová struktura podle [1] **připouští** i odříkávání částí tras JŘ s využitím položky `CZDeactivatedSection` (kap. 3.2.4), dále uvádí, že se částečné odřeknutí trasy a provozní odklony do CIS **nepředávají** (kap. 3.2.5). Analýza stávajících dat ukázala, že se tato položka ve zprávách skutečně nevyskytuje.

Dokumentace připouští pro identifikační položky `PlannedTransportIdentifiers` kardinalitu (1..*), ale vzhledem k tomu, že zpráva o zrušení by měla vždy odkazovat na jednu zprávu `CZPTTCISMessage` a tyto mají vždy uvedeny dva identifikátory (viz 1.3.1), předpokládáme i u zpráv `CZCanceledPTTMessage` vždy právě dva identifikátory (stávající data tento předpoklad potvrzují).

1.3.3 Identifikátory

Jako identifikátory jsou využity položky typu `PlannedTransportIdentifiers`. Ten má následující strukturu:

- `PlannedTransportIdentifiers`
  - `ObjectType` (1) – typ označovaného objektu. Přípustné hodnoty jsou `TR` a `PA`
  - `Company` (1) – číslo společnosti, která identifikátor vytvořila
  - `Core` (1) – řetězec popsaný reg. výrazem `[\-\*0-9A-Z]{12}`
  - `Variant` (1) – řetězec popsaný reg. výrazem `[0-9A-Z]{2}`
  - `TimetableYear` (1) – rok (převážné) platnosti JŘ

Součástí typu může být také položka `StartDate`, u které uvádí [1] kardinalitu 0..1, ve stávajících datech se však tato položka nikdy nenachází.

V obou typech zpráv se vždy nachází dvě položky typu `PlannedTransportIdentifiers`. Jedna z nich je typu `TR` a označuje jeden „obchodní případ“ konkrétního železničního dopravce – fyzickým reprezentantem jednoho objektu typu `TR` v jeden konkrétní den je vlak (dále bude pro tuto skutečnost využíváno pouze slovo „vlak“). Druhá je typu `PA` a označuje jeden datový jízdní řád (tedy de facto také jednu konkrétní zprávu `CZPTTCISMessage`) [1, kap. 1.2].

Minimálním identifikátorem zprávy `CZPTTCISMessage` je tedy čtveřice (`Company`, `Core`, `Variant`, `TimetableYear`) z položky typu `PlannedTransportIdentifiers`, jejíž atribut `ObjectType` má hodnotu `PA`. Výskyt tohoto identifikátoru byl analýzou stávajících dat

potvrzen. Ačkoliv by měl být tento identifikátor jednoznačný napříč všemi zprávami, odhalili jsme několik anomálií. Ve stávajících datech se nachází celkem 63 dvojic zpráv s identickým identifikátorem typu `PA`.

V 51 případech jde o duplicitní výskyt **téže** zprávy: v datech z června (v adresáři `2022-06`) se nachází duplikáty zpráv obvykle z dubna (z adresáře `2022-04`; celkem 48 případů), případně z května (celkem 3 případy). Ačkoliv není jasné, proč k tomuto ději dochází, z hlediska využití není významný, neboť jsou duplicitní zprávy totožné s původními a nijak tedy neovlivní už uložená data.

Ve 12 případech ovšem zpráva s duplicitním identifikátorem obsahuje pozměněná data. Vždy jde o dvojici zpráv z prosince 2021 a června 2022. Charakter rozdílu ve zprávách je vždy takový, že zpráva vydaná v červnu obsahuje časy v lokacích specifikované s jinou časovou zónou: např. v prosincové zprávě je uveden čas `08:59:00.000000+01:00`, zatímco v červnové je `08:59:00.000000+02:00`. Některé červnové zprávy modifikují také „kalendář“ (položku `PlannedCalendar`). V takto duplicitních zprávách je však vždy shodné (prosincové) datum vydání zprávy `CZPTTCreation`, jen podle dat uvnitř zpráv tedy není možné určit, která z nich je novější nebo která popisuje platné informace.

Identifikátory typu `TR` naopak určují jeden konkrétní vlak, nejsou tedy unikátní napříč zprávami. Je možné jej využít pro **seskupení** zpráv, které se týkají jízdního řádu stejného vlaku. Jeden vlak může být tvořen několika jízdními řády, a to v podstatě ve dvou případech:

1. Základní („obvyklý“) jízdní řád vlaku je rozdělen do několika zpráv (se shodným id. typu `TR` a rozdílným id. typu `PA`), které mají navzájem doplňující se kalendář. Tento jev je vysvětlen v [1], kap. 5.1.

2. JŘ vlaku byl odřeknut a nahrazen „mimořádným“ odklonovým JŘ. V tomto případě zpráva o novém JŘ obsahuje položku `RelatedPlannedTransportIdentifiers`.

Datový typ `RelatedPlannedTransportIdentifiers` má strukturu shodnou s typem `PlannedTransportIdentifiers` a položky tohoto typu se vyskytují vždy maximálně jednou uvnitř položek `CZPTTCISMessage.Identifiers`, a to vždy, když zpráva reprezentuje odklonový JŘ, který nahrazuje jiný JŘ. Tyto položky tedy představují odkaz na jiný JŘ (jinou zprávu).

Odklonové zprávy typu `CZCanceledPTTMessage` neobsahují žádný vlastní identifikátor, který by jim přisuzoval identitu. Obsahují vždy dvojici identifikátorů, která ukazuje na konkrétní předtím vydanou zprávu `CZPTTCISMessage`, na které jsou tak závislé. Souslednost odklonových zpráv je možné určit podle hodnoty data a času vydání zprávy v položce `CZPTTCancelation` (trojici *(PlannedTransportIdentifiers, PlannedTransportIdentifiers, CZPTTCancelation)* by tedy bylo možné považovat za jednoznačný identifikátor odklonové zprávy).

1.3.4 Počty zpráv a vlaků

Tabulka 1.1 naznačuje, pro kolik vlaků se „obvyklý“ JŘ skládá z více zpráv a pro kolik vlaků byly vydány odklonové zprávy. Ve stávajících datech se nachází **17 342** jedinečných vlaků (identifikátorů typu `TR`). Z údajů je možné odhadnout, že pro více než 62 % vlaků nebyla vydána žádná odklonová zpráva, jsou tedy v podstatě „statické“. Také je vidět, že skutečnou trasu je nutné vypočítat z několika různých datových zpráv pro asi **39 %** vlaků, pokud bychom ale uvažovali pouze bázové zprávy, šlo by o asi 6,69 % vlaků.

Bázových zpráv	Vlaků
>0	17 342
>1	1 160
>2	505
>3	183
>4	22
>5	5
>6	3
>8	1

(a) Rozložení počtu vlaků v závislosti na počtu zpráv tvořících jejich základní jízdní řád.

Odklonových zpráv	Vlaků
=0	10 911
>0	6 441
>1	3 657
>2	2 409
>4	1 021
>6	508
>8	222
>10	177

(b) Rozložení počtu vlaků v závislosti na počtu odklonových zpráv.

Zpráv	Vlaků
>1	6 771
>2	4 067
>3	2 620
>4	1 699
>5	1 171
>6	842
>8	452
>10	278

(c) Rozložení počtu vlaků v závislosti na celkovém počtu zpráv.

Tabulka 1.1: Rozložení počtu vlaků podle počtu zpráv, které tvoří jejich základní jízdní řád (neobsahují položku `RelatedPlannedTransportIdentifiers`), a podle počtu zpráv, které nesou informaci o odklonu vlaku (obsahují jmenovanou položku).

Kapitola 2

Návrh způsobu uložení dat

Základním aplikačním požadavkem je rychlé vyhledávání spojů mezi dvěma lokalitami, které jsou platné v zadaném datu a čase. Je žádoucí, aby lokality byly vyhledatelné i při neúplném zadání názvu (uživatelé jen stěží zapíšou správně vždy celý název stanice včetně různých přívlastků typu „hl. n.“).

2.1 Efektivní stahování dat

Každý soubor s daty chceme stáhnout a naimportovat data z daného souboru do databáze pouze jednou. Při následných aktualizacích dat tedy jenom zkontrolujeme, které soubory jsme již dříve stáhli a importovali, a stáhneme pouze nové soubory od poslední aktualizace dat.

Tento problém řešíme zavedením kolekce v databázi pro udržování seznamu již importovaných souborů. Do kolekce ukládáme pouze tu část URL specifickou pro daný soubor na stránkách zdroje. Tímto zajistíme jednoznačnost při rozlišování souborů, ale zároveň přenositelnost řešení pro případ, že by došlo ke změně zdroje (ovšem struktura ukládání souborů na novém zdroji by zůstala zachována).

Přestože při prvním stažení všech souborů musíme stáhnout značný objem dat (všechny dosud vydané soubory pro dané období), tuto časově náročnou fázi vykonáme pouze jednou a po úspěšném importu dat vložíme do kolekce importovaných souborů části URL pro všechny nově importované soubory. Import stažených dat do databáze tedy poprvé bude také trvat déle.

Kolekce importovaných souborů má následující schéma:

```
imported_files : {
    _id: string  # cesta importovaného souboru
}
```

`link` slouží jako indexovaný klíč do kolekce pro rychlé vyhledání konkrétního souboru. Pokud se soubor v kolekci nachází, již nebude znovu stahován. V opačném případě dojde k jeho stažení a naimportování a vložení části URL pro tento soubor do kolekce `imported_files`.

2.2 Import stažených dat

Vždy pro nově stažená data chceme samotná data ukládat do databáze ve třech fázích do jedné hlavní kolekce. Nejdříve naimportujeme data hlavního souboru pro daný rok (soubory s názvem tvaru `GVD<year>.zip`). Následně k takto uloženým datům připojíme data

informující o změnách, aktualizacích, náhradních linkách, ... Jako poslední naimportujeme data ze souborů informujících o zrušení jízdního řádu v určitém termínu.

Všechny výše zmíněné fáze ukládají data do kolekce `trains`, která obsahuje informace o vlacích, jízdních řádech náležících k daným vlakům (a jejich změnách či zrušeních) a odpovídajícím trasám těchto jízdních řádů.

Kolekce `trains` má následující schéma:

```
trains : {
  train_id: { # Compound index.
    Company: int,
    Core: string,
    Variant: string,
    TimetableYear: int
  }
  paths: [
    {
      path_id: {
        Company: int,
        Core: string,
        Variant: string,
        TimetableYear: int
      },
      CZPTTCreation: Date,
      Locations: [
         { PrimaryLocationName: string,
           LocationPrimaryCode: int,
           ALD: { Time: Date },
           ALA: { Time: Date },
           <remaining_elements> }
      ],
      PlannedCalendar: {
        StartDateTime: Date,
        EndDateTime: Date,
        BitmapDays: string
      },
      Cancellations: [
         {
           CZPTTCancelation: Date,
           PlannedCalendar: {
             StartDateTime: Date,
             EndDateTime: Date,
             BitmapDays: string
           }
         }
      ]
      related_path_id: { ... },  # same as ids above
      <remaining_elements>
    }
  ]
  <remaining_elements>
}
```

Vlak je jednoznačně identifikován `train_id`, které tvoří složený index skládající se z kódu společnosti, hlavního kódu, variantního čísla a roku platnosti. Ostatní elementy vždy náleží určitému vlaku, tedy jsou všechny uloženy v dané struktuře vlaku s odpovídajícím `path_id` včetně neznámých elementů dat.

Vlaky mohou mít naplánováno více cest, tedy všechny cesty (původní i aktualizované) pro daný vlak chceme ukládat jako vnořenou strukturu do pole `paths`. Cesta je jednoznačně identifikována pro daný vlak `path_id`, které musí být, obdobně jako u `train_id`, složeným indexem pro možnost rychlého vyhledání cesty.

Pro vyhledání konkrétní cesty pro konkrétní vlak také potřebujeme tzv. `multikey` index nad dvojicí `(train_id, path_id)`, tedy pro klíč do kolekce `trains` a index elementu ze seznamu `paths` s odpovídajícím `path_id`.

Cesta dále obsahuje vnořené struktury obsahující informace o:

- platném kalendáři pro daný jízdní řád (pro danou cestu)
  - obsahuje informace o začátku a konci platnosti daného jízdního řádu a mapě dnů platnosti
- seznamu lokací, kterými vlak projíždí podle platného jízdního řádu
  - obsahuje veškeré informace o lokaci
- seznamu oznámení o zrušení platnosti jízdního řádu pro danou trasu na určitou dobu
  - obsahuje informace o začátku a konci platnosti zrušení daného jízdního řádu a mapě dnů platnosti zrušení
- `related_path_id` odpovídající jinému `path_id` pro daný vlak odkazující na související trasu pro předání informace o původní trase odklonovým a náhradním trasách
  - nesou identifikátor související cesty `path_id`

Protože se oznámení o zrušení jízdních řádů vážou právě na konkrétní jízdní řád, ale mohou rušit jen jeho části (například jen pro jediný den či krátký časový okamžik), je vhodné ukládat tato data jako atributy odpovídající trasy. Pro naši práci si potřebujeme uložit pro ovlivněný vlak a ovlivněný jízdní řád (trasu) hlavně datum počátku a konce platnosti zrušení jízdního řádu a mapu dnů, ve kterých ke zrušení dochází.

Všechny atributy, jejichž význam neznáme nebo které by mohly být užitečné v budoucnosti, potřebujeme ukládat do databáze, přestože pro ně zatím nemáme použití. Vždy je tedy chceme ukládádat k odpovídajícím elementům v databázi, jako například elementy v `NetworkSpecificParameters` k odpovídajícím trasám apod. Tyto elementy jsme symbolicky znázornili ve schématu pomocí `<remaining_elements>`.

2.3 Vyhledávání cesty mezi lokacemi

Přestože struktura uložení dat popsaná v sekci 2.2 umožňuje rychle vyhledat informace o konkrétním vlaku (například zda v daný den jede), pro její využití však musíme nejprve získat ID daného vlaku. Toto schéma struktury není příliš vhodné pro efektivní podporu hlavního případu využití – vyhledání vlaku mezi počáteční a koncovou stanicí. Za tímto účelem tedy zavádíme předpočítaný materializovaný pohled na kolekci `trains`, jehož cílem je efektivně vyhledat ID vlaku na základě zdrojové a cílové stanice. Jeho schéma je následující:

```
locations: {
    _id: int,  # Start location primary code
    name: string  # Start location primary name (text index)
    connections: [
        {
            train: object  # Train ID
            to: [int, int, ...]  # IDs of destinations (multikey index)
            valid_from: datetime  # Validity start
            valid_to: datetime  # Validity end (index with valid_from)
            valid_bitmap: string
        }
    ]
}
```

Výhodou využití takto zaindexovaného materializovaného pohledu je, že jeho relativně náročný a pomalý výpočet je nutné provést jen jednou za čas, při aktualizaci dat. Při vyhledávání spoje pak ovšem zajistí, že možné vlaky mezi stanicemi budou nalezeny rychle, protože klíče a indexy v tomto pohledu prakticky přesně odpovídají uživatelskému vstupu.

Kapitola 3

Zvolená NoSQL databáze

Na základě kapitoly 2 jsme se rozhodli použít NoSQL databázi MongoDB[1].

MongoDB je multiplatformní open-source NoSQL databáze zaměřená na práci s dokumenty. Interně MongoDB používá JSON object pro ukládání dokumentů. Jelikož naše data představují XML dokumenty s mnoha *neznámými* či dosud *nevyužívanými* daty, je vhodné použít právě MongoDB, neboť MongoDB umožňuje snadné uložení celých takových XML souborů (převedených na JSON-like dokumenty) bez ztráty informace, zatímco nad známými a využívanými daty můžeme efektivně provádět potřebné operace: zavádět indexy nad ukládanými daty, aktualizovat existující data, vyhledávat a výhodně využívat pohledů (normálních i materializovaných) nebo se vhodně dotazovat na data pomocí agregačních pipeline. Všech těchto vlastností jsme použili při implementaci našeho řešení. Pro ukládání dat používáme vnořování struktur, což považujeme za jednu z největších výhod dokumentových databází, kolekce a reference na jiné záznamy. Všechny tyto požadované vlastnosti jsou typické právě pro dokumentové NoSQL databáze.

Očekávaná práce s databází je podle analýzy dat z kapitoly 1 dvojí: jednou za čas stažení dříve nestažených aktualizací a jejich import do databáze aktualizující stávající data, který tedy může být časově náročný, ale hlavně časté (a tedy nutně rychlé) vyhledávání spojení mezi dvěma stanicemi. Rozhodli jsme se tedy použít MongoDB pro možnost předpočítání předpokládaných dotazů při každé aktualizaci dat a uložení takto předpočítaných výsledků pro rychlé zpracování dotazů do samotné databáze (materializovaný pohled na lokace).

Protože pracovní data představují dokumenty typu klič-hodnota, kde klíč je složený z více hodnot, a hodnota samotná je strukturovaná, MongoDB představuje skvělou volbu pro uložení pracovních dat, neboť MongoDB podporuje indexy následujících typů, z nichž jsme všechny při návrhu databáze potřebovali pro vhodné a rychlé vyhledávání nad daty:

- jednoduché indexy (primární kód lokace aj.),
- složené indexy pro indexaci jednotlivých vlaků nebo tras, jejichž index sestává z několika hodnot (čísla společnosti, roku platnosti, základního a variantního identifikátoru vlaku)

Roli ve výběru vhodné databáze také hrál předpokládaný požadavek na budoucí distribuovanost ukládaných dat. Způsob uložení dat jsme navrhli tak, aby bylo možné data výhledově ukládat do více clusterů. Námi zvolený způsob uložení dat s navrženými indexy umožňuje do budoucna distribuovat data pomocí funkcionality MongoDB zvané sharding[2].

Za nevhodné databáze pro naše cíle jsme rozhodli následující NoSQL databáze:

[1] `https://www.mongodb.com/`

[2] `https://www.mongodb.com/docs/manual/sharding/`

- Grafové databáze (například Neo4j[3])
  - Naše data nemají strukturu, která by mohla využít výhod grafové databáze.
  - Bylo by sice možné teoreticky reprezentovat stanice jako uzly v síti stanic a hrany mezi nimi jako cesty mezi stanicemi existující, ale rozhodli jsme se tyto vazby řešit pomocí materializovaného pohledu v MongoDB z důvodu rychlejšího přístupu k vyhovujícím trasám, zanořených struktur v datech nebo neznámých elementů v datech.
  - Jiné části našich dat nemají žádnou topologickou strukturu, kterou by bylo vhodné reprezentovat pomocí grafové databáze. I Kdybychom tedy použili grafovou databázi pro síť stanic, pro ostatní data bychom nemohli ukládat vhodným způsobem využívajícím předností grafové databáze.
- Databáze časových řad (například InfluxDB[4])
  - V datech se nevyskytují časové řady. Tento typ databází je pro naše data zcela nevhodný.
- Sloupcové databáze (například Apache Cassandra[5])
  - Sloupcové databáze tohoto typu jsou založeny na předem známém schématu databáze, které my předem neznáme kvůli potenciálně neznámé struktuře dat (například *NetworkSpecificParameters*). Nemůžeme tedy použít ani tento typ databází pro ukládání našich dat.
  - Kdyby nebylo problémů s chybějícím schématem, sloupcové databáze by taktéž připadaly v úvahu. Sloupcové databáze dokážou reprezentovat námi navržený způsob uložení dat, a bylo by tak možné zvolit právě například Apache Cassandra pro ukládání našich dat.

[3] `https://neo4j.com/`
[4] `https://www.influxdata.com/`
[5] `https://cassandra.apache.org/_/index.html`

Část II

Návrh, implemetace a použití aplikace

Kapitola 4

Návrh aplikace

Pro stažení dat (a následné aktualizace dat), jejich uložení do MongoDB a následné dotazování nad daty jsme použili Python s Python distribucí PyMongo[1], umožňující napojení na MongoDB z Pythonu a provádění všech operací s databází. Pro stažení dat ze zdroje jsme použili Beautiful Soup[2].

Vytvořili jsme jedinou aplikaci pro příkazový řádek, kde uživatel může zadanými argumenty programu volit, jestli chce aktualizovat data (importovat do databáze všechna dosud nestažená data), nebo vyhledávat spojení v již stažených datech.

Aplikace se skládá ze tří hlavních modulů:

- **data fetcher** (soubor `data_fetcher.py`): stahování dat z CIS,
- **database importer** (soubor `db_importer.py`): import nově stažených dat do databáze a aktualizace stávajících dat v databázi,
- **path searcher** (soubor `lookup.py`): vyhledává spojení pro zadanou zdrojovou a cílovou stanici z dat v databázi.

Při stahování souborů je nejdříve ověřeno v databázi importovaných souborů, že daný soubor ještě nebyl stažen a importován do databáze. Soubor je případně stažen a pomocí *upsert* operací vložen do databáze. Následně se aktualizuje kolekce importovaných souborů s informací o importu právě staženého souboru, aby při příštím stahování aktualizací již tento soubor nebyl stažen.

Při importu souboru do databáze je aktualizována kolekce obsahující informace o vlacích a jejich příslušných jízdních řádech. Jakmile jsou všechny soubory stažené a importované v tomto spuštění aplikace v režimu aktualizace dat, dojde k přepočítání předpočítaného materializovaného pohledu obsahujícího informace o možných trasách mezi startovací a cílovou stanicí a jízdních řádů, které zajišťují spojení mezi těmito trasami. To je provedeno pomocí agregační pipeline, která pro každou stanici v každém spojení v kolekci `trains` vyhledá stanice, které v daném spojení následují po zpracovávané stanici. Tyto dílčí výsledky jsou nakonec spojeny do jednoho pro každou stanici.

4.1 Tvorba materializovaného pohledu

Výše popsaný materializovaný pohled je vytvořen pomocí jedné agregační pipeline, která přetransformuje kolekci vlaků `trains` na kolekci lokací `locations`. Nejprve jsou pro vlaky rozbalena pole obsahující cesty a pro cesty jsou poté rozbaleny zastávky s pomocí fáze

[1] `https://pymongo.readthedocs.io/en/stable/index.html`
[2] `https://www.crummy.com/software/BeautifulSoup/`

`$unwind`. Následně pro každou lokaci, ve které vlak staví (lze poznat dle pole `TrainActivity`) jsou ve fázi `$lookup` dohledány stanice, které v příslušné cestě následují po současně zpracovávané stanici a ve kterých vlak staví.

Poddotaz pro `$lookup` nejprve vyhledá odpovídající cestu z hlavní části dotazu, ve které následně odstraní stanice, ve kterých vlak nestaví (kombinace `$set` a `$filter`). Následně jsou ponechány pouze stanice, které následují po stanici zpracovávané v hlavní části dotazu (kombinace `$indexOfArray` a `$lastN` aplikovaných na pole). Výstup `$lookup` je poté upraven do požadovaného formátu pomocí fáze `$project`.

Po fázi `$lookup` je nutné zkombinovat dílčí výsledky z jednotlivých cest pomocí fáze `$group`, kdy pole možných cílových lokací jsou přidávána do jednoho společného pole, které je nakonec zploštěno operátorem `$reduce`. Výsledná kolekce je poté zapsána jako materializovaný pohled s pomocí fáze `$merge` v módu, kdy v případě existence takového dokumentu je dokument přepsán nově vypočítaným dokumentem.

4.2 Vyhledávání spojení

Vstupem vyhledávání je:

- dvojice textových názvů výchozí a cílové stanice (povinné),
- datum dne, pro který jsou spojení vyhledávána (povinné),
- časy „odjezd před“, „odjezd po“, „příjezd před“ a „příjezd po“ (nepovinné).

Pro vyhledání přímého spojení mezi dvěma stanicemi aplikace postupuje v několika krocích, které jsou podrobněji popsány níže:

1. vyhledání identifikátorů stanic (dotaz nad kolekcí `locations`),
2. vyhledání vlaků, které v daném dni potenciálně mohou spojovat stanice (agregace nad kolekcí `locations`),
3. přesné filtrování jízdních řádů potenciálních vlaků podle zadaného data (agregace nad kolekcí `trains`),
4. zploštění zbylých JŘ do skutečného seznamu stanic a ověření časů příjezdu/odjezdu (transformace v kódu).

Pro efektivní práci s dokumenty reprezentujícími stanice byla zvolena identifikace pomocí celočíselného identifikátoru, který odpovídá položce `LocationPrimaryCode` ve zdrojových datech. Nejprve je tedy nutné vyhledat tento identifikátor podle zadaných textových názvů. Dává to smysl i z hlediska uživatelské interakce: uživatel použije např. pouze název „Praha“, ale v Praze je evidováno 46 stanic. V kolekci `locations` je proto zaveden také **textový index**, který umožňuje efektivnější textové vyhledávání (ideální by bylo fulltextové vyhledávání, to však MongoDB nabízí pouze v rámci svého komerčního cloudového řešení). Aplikace tedy pomocí dvou jednoduchých dotazů ve funkci `lookup.find_station` vyhledá odpovídající stanice a nabídne uživateli přesný výběr.

V dalším kroku probíhá nalezení potenciálních spojů mezi dvěma stanicemi. Dotaz je proveden opět nad kolekcí `locations`, tentokrát se vyhledává především podle indexovaného identifikátoru, tento dotaz je tedy maximálně rychlý. Realizován je jako jednoduchá agregace, která v první fázi `$match` provede vyhledání, ve druhé fázi `$set` vyfiltruje dokumenty popisující vlaky, které **nevedou do cílové stanice** (pole `to` neobsahuje daný identifikátor)

a jejichž rozsah platnosti neobsahuje požadované datum – tím se výrazně omezí počet dále zpracovávaných vlaků.

Z navrácené kolekce vlaků jsou poté *v kódu aplikace* odfiltrovány vlaky podle bitmapy dnů, kdy vlak skutečně jede[3]. Pole identifikátorů vyfiltrovaných vlaků je ve třetím kroku (ve funkci `lookup._train_lookup_cursor`) použito v první fázi `$match` složitějšího agregačního řetězce nad kolekcí `trains` – díky indexování je proto i zde efektivně provedeno (ještě před složitější částí řetězce) omezení na typicky maximálně desítky dokumentů. Druhá fáze přímo v databázi odfiltruje všechny JŘ příslušné vlaku (dokumenty v poli `paths`), které nevyhovují cílovému datu, a to **včetně** uvážení bitmapy dnů[4]. Zároveň jsou odfiltrovány všechny JŘ, pro které existuje v daném dni platná odřeková zpráva. V poslední fázi `$match` jsou odfiltrovány vlaky, ve kterých po předchozí operaci nezůstaly žádné platné JŘ.

Posledním krokem je výpočet skutečné trasy vlaku, resp. seznamu po sobě jdoucích stanic. Vstupem je pole JŘ, které jsou sice v daném dni platné, ale některé z nich mohou podávat „aktuálnější“ informaci o skutečné trase: ne všechny JŘ jsou před vydáním odklonového JŘ zrušeny příslušnou zprávou `CZCanceledPTTMessage`. Nahrazený JŘ je ale možné poznat tak, že se v položce `related_path_id` *jiného* JŘ (stejného vlaku) nachází jeho identifikátor. Zároveň také může být jedna trasa složena z několika JŘ. Následující pseudokód ukazuje algoritmus sestavování skutečné trasy. Využívá předpokladu, že platí vždy nejnovější JŘ – časová složitost je tedy shora omezená složitostí řazení, což zvládáme provést v $\mathcal{O}(n \log n)$.

```
paths.sort(key='CZPTTCreation') # Seřadíme JŘ podle data vydání
mappings = {} # Slovník {ID JŘ : ID nejnovějšího nahrazujícího JŘ}

for path in paths:
    mappings[path.id] = path.id # Na počátku každý JŘ „nahrazuje sám sebe“

for path in reversed(paths): # Procházíme JŘ pozpátku, od nejnovějšího
    if "related_id" not in path:
        continue
    else:
        # Pokud nahrazovaný JŘ ještě nebyl nahrazen,
        # nahradíme jej tímto JŘ
        if mappings[path.related_id] == path.related_id:
            mappings[path.related_id] = mappings[path.id]

locations = []
# Hodnoty nastavené ve slovníku mappings označují nejnovější
# nahrazující JŘ pro každý původní JŘ; nutné zbavit se duplicit
for path_id in unique_values(mappings):
    path = paths[path_id]
    for location in path["Locations"]:
        if location contains TrainActivityType == 0001:
            locations.append(location)
```

Pro každý vlak je takto získán seznam stanic, poté je ještě v každém z nich nutné lineárním průchodem najít výchozí a cílovou stanici a ověřit, že časy odjezdu a příjezdu vyhovují kritériím hledání.

[3] Tuto operaci by bylo možné provést i v agregačním řetězci databáze, ale v tomto případě je to zbytečné.

[4] Využívá operátory `$substrBytes`, `$eq` a `$dateDiff` – v této fázi je vstupních dokumentů málo, a jejich použití tedy nemá výrazný negativní časový dopad.

4.3 Získávání dat

Před prvním spuštěním i po aktualizaci dat je nutné spustit aplikaci s příslušným příkazem (argumentem) `fetch-updates`, který stáhne seznam souborů z CIS, najde ještě neimportované soubory a provede import do databáze. Vzhledem k časové náročnosti přepočítávání kolekce `locations` je vhodné jej spouštět v konzervativnějších časových intervalech – v produkční aplikaci by bylo vhodné tento proces spouštět na pozadí např. jednou za hodinu.

Kapitola 5

Způsob použití

Pro spuštění aplikace je třeba mít zprovozněné následující:

- Python (testováno na verzi 3.10.8; minimální nutnou verzí je 3.10),
- MongoDB (testováno na verzi 6.0.2-1), tu je možné spustit například pomocí nástroje Docker (testováno na verzi 1:20.10.19-1).

Je potřeba nainstalovat Python moduly z `requirements.txt` pomocí:
`$ pip install -r requirements.txt`.

Následně je nutné zprovoznit vytvořit a spustit MongoDB databázi například pomocí:
`$ sudo docker run -name mongodb mongo`,
případně následně spouštět databázi příkazem:
`$ sudo docker start mongodb`.

Aplikace předpokládá, že databáze bude dostupná na adrese `mongodb://localhost:27017`. Adresu je možné nastavit pomocí proměnné prostředí `CONNECTION_STRING`, případně v souboru `settings.py`.

Aplikaci lze nyní spustit příkazem ve tvaru `sudo python __main__.py <arguments>` (pro vypsání nápovědy k používání aplikace můžeme použít parameter `-h` a znovu tentýž parameter použít pro získání nápovědy k jednotlivým příkazům podporovaným aplikací `<command> -h`), kde `arguments` podporuje následující příkazy:

- `fetch-updates`: Zkontroluje existenci nových, ještě nestažených aktualizací jízdních řádů, tyto aktualizace stáhne a naimportuje do databáze.
- `search-station`: Nalezne všechny stanice odpovídající zadanému jménu.
- `search`: Umožňuje zadání dvou stanic a data a času, podle kterých bude vyhledáno spojení.

Pro stažení dosud nestažených dat tedy můžeme spustit:
`$ sudo python __main__.py fetch-updates`.

Pro vyhledání trasy mezi dvěma stanicemi potom:
`$ sudo python __main__.py search Domašín Vlašim 2022-11-03T12:13:00`.
Tímto bychom dostali celou trasu vlaku jezdícího přes Domašín a Vlašim v zadaném termínu.

Kapitola 6

Experimenty

Následující experimenty byly prováděny na operačním systému Fedora, databáze byla spuštěna v `podman` kontejneru. Pro spouštění Python programu bylo využito virtuální běhové prostředí z balíku `virtualenv` (s verzí Pythonu 3.10) s nainstalovanými závislostmi dle `requirements.txt`.

V rámci experimentů jsme nejprve vyzkoušeli rychlost stáhnutí a importu do naprosto čisté databáze.

Během tohoto experimentu jsme změřili dobu trvání jednotlivých fází, tj. stažení, importu dat do databáze a vytvoření materializovaného pohledu. Samotné stažení všech datových souborů trvá zhruba od 31 minut 15 sekund do 48 minut a 30 sekund. Tento údaj je ovšem silně zatížen rychlostí internetového připojení a také možným vytížením cílového serveru. Následný import všech souborů do databáze pak trval 9 minut a 30 sekund. Nakonec výpočet materializovaného pohledu trval 2 minuty a 30 sekund.

Při znovuspuštění importu dat po nějaké době zabrala kontrola nových souborů 1 minutu a 8 sekund, následný import řádově nízké sekundy (zde silně závisí na množství nových souborů, které je nutné importovat) a přepočet materializovaného pohledu pak 2 minuty a 28 sekund. Z těchto údajů je možné vidět, že v případě pravidelného importování v určitém intervalu bude hlavním zpomalujícím faktorem výpočet materializovaného pohledu. Za rozumné bychom považovali například import každých 10 minut, při takové frekvenci se vzhledem k době výpočtů dostanou informace do databáze nejpozději (v nejhorším možném případě) do 15 minut od jejich původního nahrání, což by mohla být pro hlavní případ užití přijatelná prodleva.

Následně jsme testovali efektivitu vyhledávání spoje mezi zadanými stanicemi. Při měření času byla zanedbána doba výběru přesného jména zdrojové a cílové stanice (např. při zadání lokace "Brno"výběr "Brno hl. n."), neboť se jedná o uživatelskou akci. Měřen tedy byl čistě čas od potvrzení výběru uživatelem po navrácení výsledku. Obrázek 6.1 ukazuje část vyhledaných spojení mezi Brnem a Prahou pro den 26. 10. 2022. Vyhledání všech takových spojení, včetně přesné cesty jednotlivých spojů, trvalo 1 milisekundu.

Dalším provedeným experimentem bylo vyhledání spoje, který je zrušen. Na dotaz na spoj mezi Blanskem a Brnem (kde momentálně probíhá výluka kvůli opravě trati[1]) aplikace pro vyhledávání vrátila chybu, že takové spojení pro daný den neexistuje.

Pro všechny další experimenty jsme dostali obdobné výsledky. Z těchto údajů můžeme usoudit, že použitá struktura uložení dat (společně s jejími indexy a pohledy) umožňuje provádět velmi rychlé dotazy, což je hlavním cílem projektu.

[1] `https://www.idsjmk.cz/vvv2021.html`

```
Connections between 'Brno hl. n.' and 'Praha hl. n.' on 2022-10-26 00:00:00:
Train 576, dep. 03:10:00, arr. 06:19:00:
    Station                         Arrival         Departure
  > Brno hl. n.                     -               03:10:00
  > Brno-Kr. Pole                   03:21:30        03:22:30
  > Kolín                           05:30:00        05:31:00
  > Praha-Libeň                     06:12:00        06:13:00
  > Praha hl. n.                    06:19:00        -

Train 574, dep. 04:38:00, arr. 07:42:00:
    Station                         Arrival         Departure
  > Břeclav                         -               04:07:00
  > Brno hl. n.                     04:36:00        04:38:00
  > Brno-Kr. Pole                   04:49:30        04:50:30
  > Kolín                           06:57:00        06:58:00
  > Praha hl. n.                    07:42:00        07:51:00

Train 1042, dep. 05:09:00, arr. 08:19:00:
    Station                         Arrival         Departure
  > Brno hl. n.                     -               05:09:00
  > Brno-Židenice                   05:13:30        05:14:00
  > Žďár n.S.                       06:07:30        06:08:30
  > Havlíčkův Brod                  06:31:00        06:32:00
  > Kolín                           07:30:00        07:31:00
  > Praha hl. n.                    08:19:00        08:27:00
```

Obrázek 6.1: Vyhledání spoje mezi Brnem a Prahou

Literatura

[1] Futera, M. *Popis datových jízdních řádů předávaných do Celostátního Informačního Systému o jízdních řádech veřejné osobní dopravy* [online]. Verze 1.09. Praha: Správa železnic, září 2021 [cit. 2022-10-16]. Dostupné z: `https://portal.cisjr.cz/pub/draha/celostatni/szdc/`.

[2] Ministerstvo dopravy ČR. *Jízdní řády veřejné dopravy* [online]. 2022 [cit. 2022-10-16]. Dostupné z: `https://mdcr.cz/Dokumenty/Verejna-doprava/Jizdni-rady,-kalendare-pro-jizdni-rady,-metodi-%281%29/Jizdni-rady-verejne-dopravy`