FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Případová studie využití NoSQL databázi pro část datového skladu VZP
Student:	Bc. Stanislav Němec
Vedoucí:	Ing. Michal Valenta, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce zimního semestru 2022/2023

Pokyny pro vypracování

1. Seznamte se se strukturou části L2 schématu datového skladu (DS) VZP, který je implementován v technologii Oracle.
2. Analyzujte typické dotazy, které jsou nad schématem prováděny v nástroji Oracle Analytics Server (OAS).
3. Zpracujte rešerši NoSQL řešení používaných v datových skladech. Zaměřte se na open-source řešení, která jsou podporovaná nástrojem OAS.
4. Navrhněte, implementujte a vyhodnoťte případovou studii náhrady části L2 datového skladu (body 1 a 2) dvěma různými NoSQL technologiemi (bod 3). Jako referenční řešení použijte stávající implementaci v Oracle.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 27. května 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Případová studie využití NoSQL databází pro část datového skladu VZP

Bc. Stanislav Němec

Katedra softwarového inženýrství
Vedoucí práce: Ing. Michal Valenta, Ph.D.

4. ledna 2022

Poděkování

Děkuji společnosti Všeobecná zdravotní pojišťovna a Ing. Kristiánu Vadkertimu za příležitost pracovat na tomto zadání. Dále děkuji Ing. Michalu Valentovi, Ph.D. za cenné rady a pomoc při vedení této práce. V neposlední řadě děkuji své rodině a své partnerce za podporu při studiu a při tvorbě této závěrečné práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 4. ledna 2022 .....

Odkaz na tuto práci

Němec, Stanislav. *Případová studie využití NoSQL databází pro část datového skladu VZP*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

Předmětem této diplomové práce je provedení případové studie použití NoSQL databázových technologií pro část datového skladu Všeobecné zdravotní pojišťovny. To zahrnuje rešerši NoSQL databází, analýzu současného řešení a analýzu typických dotazů do datového skladu. Dalším cílem je výběr dvou NoSQL technologií a jejich porovnání se stávajícím řešením v databázi Oracle. Výsledky tohoto porovnání poslouží jako základ pro rozhodnutí o vhodnosti těchto technologií pro využití v části datového skladu.

Na základě analýzy současného řešení a typických dotazů jsem identifikoval požadavky na případné řešení. Na základě těchto požadavků a rešerše NoSQL databází jsem vybral nástroje Apache Impala a kombinaci Apache Impala s Apache HBase. Tato dvě řešení jsem nainstaloval do testovacího prostředí a na ukázkových datech jsem je porovnal se stávajícím řešením v databázi Oracle. Porovnání jsem provedl s pomocí naměřených časů odezvy na poskytnuté typické dotazy.

Z dat měření vyplynulo, že s rostoucí velikostí dat si z pohledu odezvy na ukázkové dotazy vedla lépe vybraná řešení s pomocí NoSQL technologií.

Klíčová slova datový sklad, NoSQL, případová studie, Apache Impala, Apache HBase, Oracle Database

Abstract

The subject of this diploma thesis is to carry out a case study of the use of NoSQL databases as a part of a data warehouse at VZP. That includes a research of NoSQL databases, an analysis of the current solution and an analysis of queries that are commonly issued to the data warehouse. Another partial goal is to select two of the NoSQL databases and compare them to the current solution which is an Oracle Database. The results of this comparison will be utilized when deciding whether the selected technologies are suitable replacements for a part of the data warehouse.

I identified some requirements for a potential solution based on the analysis of the current solution and sample queries. Based on these requirements and the research of NoSQL databases, I chose Apache Impala and Apache Impala connected to Apache HBase. I installed these two technologies and compared them to the original solution in Oracle Database using sample data. The comparison was based on measuring response times for the sample queries.

The results from the experiments suggest that with increasing volume of data, the selected NoSQL solutions provide better response times for the sample queries.

Keywords data warehouse, NoSQL, case study, Apache Impala, Apache HBase, Oracle Database

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Datové sklady jsou v dnešní době důležitou součástí organizací. Slouží k uchování historických dat vznikajících z činností organizace a umožňují tato data využít k podpoře rozhodování o chodu organizace. S rozšiřujícím se využitím výpočetních technologií a digitalizací procesů roste objem dat, která organizace vyprodukuje. Z toho důvodu se zvyšují nároky na technologie a systémy, které mají tato data uchovávat. Správci datových skladů hledají nástroje, které jsou schopné efektivně uložit a zpracovat velké objemy dat. Jinak tomu není ani ve Všeobecné zdravotní pojišťovně (dále jen VZP), kde hledají alternativní databázové řešení pro část datového skladu.

Tato práce položí základ pro zkoumání a srovnávání NoSQL technologií, které by mohly nahradit část datového skladu ve VZP. Výstup této práce nabízí základní srovnání vybraných NoSQL technologií s aktuálním řešením. Tyto údaje poslouží správcům datového skladu ve VZP jako podklad pro rozhodnutí, zda se vyplatí tyto technologie dále zvažovat jako možné řešení pro datový sklad. Zároveň vznikne v rámci praktické části prostředí, které může být použité pro navazující testování vybraných technologií.

Toto téma jsem si zvolil kvůli jeho užitečnosti popsané výše a také z důvodu mého zájmu o problematiku datových skladů a NoSQL technologií.

Cíle této práce plynou z jejího zadání. Základem pro další postup v této práci je seznámení se se současným řešením datového skladu ve VZP. Dalším cílem je analyzovat typické dotazy, které uživatelé nad daty vytváří. Po splnění prvních dvou cílů následuje rešerše možných řešení NoSQL technologií a jejich vyhodnocení na základě uvedených požadavků a zjištění z prvních dvou kroků. Poslední cíl zahrnuje návrh a implementaci dvou řešení pomocí NoSQL technologií a jejich porovnání se stávajícím řešením v Oracle.

Tato práce je zaměřena na výběr dvou řešení na základě informací získaných z rešerší NoSQL technologií. Dalším dílčím cílem práce je vybraná řešení porovnat se stávajícím řešením a předložit základní informace k posouzení jejich vhodnosti. Cílem práce není nalezení a nasazení hotového řešení, ale provedení prvních kroků a případové studie k nalezení možného řešení.

První dvě kapitoly poskytují teoretický základ a kontext pro další postup v této práci. V první kapitole uvádím základní informace o problematice datových skladů a databázových technologií. Druhá kapitola udává kontext této práce v podobě analýzy požadavků a stručného popisu aktuálního řešení. Obsahuje také analýzu poskytnutých typických dotazů do datového skladu.

Třetí kapitola obsahuje základní popis vybraných NoSQL technologií a posouzení jejich vhodnosti pro použití v datovém skladu. Posouzení vhodnosti je provedeno na základě vlastností těchto technologií a také na základě rešerše dalších prací, které tyto technologie testují při použití v datových skladech.

Ve čtvrté kapitole popisuji postup při instalaci a zprovoznění dvou vybraných řešení a přípravu dat pro měření. Pátá kapitola obsahuje popis postupu při měření a popis dat, které jsem při měření použil. Na závěr páté kapitoly uvádím souhrnné výsledky z měření a jejich vyhodnocení.

Kapitola 1

Teoretický základ

1.1 Data, databáze a systém řízení báze dat

Data jsou obecně vnímaná jako soubor surových faktů nebo čísel, se kterými se v životě setkáváme. Takovým datům se říká surová data. Surová data je potřeba zpracovat - zjistit jejich význam, identifikovat vztahy mezi údaji, popsat a uspořádat je. Zpracováním surových dat vznikají informace. Z informací můžeme vytěžit znalosti a ze znalostí pochopení dané problematiky. [1] V rámci této diplomové práce, ve které se pracuje s daty z VZP, mohou být příkladem surových dat jméno osoby, datum a název města. Až po zjištění významu a vztahů, přiřazení popisu a uspořádání k sobě patřících údajů do jedné řádky dostáváme informaci, že se jedná o jméno zákazníka, datum narození a místo jeho trvalého bydliště.

Databáze sdružuje soubory dat a datových objektů. Data se v databázi uchovávají systematicky organizovaná tak, aby se dala v případě potřeby efektivně získat. Běžné je data udržovat v tabulkách, které drží data, která k sobě například podle obchodního významu patří. Tabulky se dělí do řádek a sloupců, v řádku je většinou více různých údajů pro jednu instanci, ve sloupci je významově stejný údaj pro více instancí. Databáze by měla nabízet možnost provádět nad vybranými daty různé operace. Běžné databázové operace jsou čtení vybraných záznamů databáze, přidání definovaných nových záznamů, úprava existujících záznamů nebo odstranění záznamů z databáze. [1] V kontextu VZP je příkladem tabulka obsahující údaje o pojištěnci. V řádku této tabulky je více údajů o jednom pojištěnci. V jednom sloupci této tabulky jsou například jména pojištěných osob a v dalším data narození.

Software, který umožňuje řídit a spravovat data v databázích, se označuje jako systém řízení báze dat (anglicky Database Management System, DBMS). Systémy řízení báze dat jsou využívány pro efektivní a rychlý přístup a manipulaci s daty. Data se udržují centralizovaně v tomto systému a další systémy a uživatelé se k systému připojí a přistupují k datům skrz něj. Tím se zamezí udržování identických dat ve více systémech najednou a dosáhne se centralizace dat. [1]

K hlavním možnostem, které systémy řízení báze dat přináší, patří poskytnutí současného přístupu k datům více uživatelům najednou. Systémy řízení báze dat také pomáhají udržovat data v konzistentním stavu. Jedním z největších přínosů těchto systémů je poskytnutí dotazovacího jazyka, pomocí něhož mohou uživatelé definovat dotazy pro přístup k datům a příkazy pro manipulaci s nimi. Příkladem takového dotazovacího jazyka je Structured Query Language (SQL). Mezi běžně používané systémy řízení báze dat patří Oracle, MySQL, MS SQL nebo PostgreSQL. [1]

1.2 Transakce a ACID

Logický a atomický celek práce v systému řízení báze dat, skládající se z jednoho nebo více SQL příkazů, se nazývá transakce. SQL příkazy spojené do jedné transakce jsou buď všechny propsány do databáze (anglicky commit) nebo je efekt na databázi všech příkazů odvolán (anglicky rollback). [2]

Skupina vlastností, které by pro všechny transakce měly být zajištěny systémem řízení báze dat, je známá jako ACID vlastnosti. ACID je zkratka pro atomicitu, konzistenci, izolaci a trvalost (anglicky Atomicity, Consistency, Isolation a Durability). [2]

Transakce je atomická tehdy, když jsou buď všechny její dílčí příkazy propsané do databáze, nebo ani jeden z nich. Před i po provedení konzistentní transakce musí být databáze v konzistentním stavu. Transakce jsou izolované tak, že změny provedené jednou transakcí jsou zaznamenatelné v ostatních transakcích až po zaznamenání transakce (commit). Trvalost transakce znamená, že jsou změny zapsané transakce uloženy trvale. [2]

1.3 Datový model a relační databáze

Důležitou součástí návrhu databází je identifikace entit, což je něco, o čem ukládáme data. Údaje o entitách, které je popisují a určují jejich vlastnosti, se nazývají atributy. Jedno konkrétní ohodnocení atributů entity se nazývá instance. Skupina hodnot, kterých může atribut nabývat, je doména. Datový model je popis entit, jejich atributů a vztahů mezi entitami. [3] Příkladem entity v kontextu VZP je pojištěná osoba nebo zdravotnické zařízení. Některé z atributů entity zdravotnického zařízení jsou jeho umístění a název. Jedno konkrétní zdravotnické zařízení je instancí této entity.

Databáze, které jsou založené na relačním datovém modelu jsou označované jako relační databáze. Logické struktury těchto databází jsou tvořeny pouze souborem relací. Relace je množina n-tic, které se také říká tabulka. Schéma relace specifikuje, jaké hodnoty a údaje mohou být ve sloupcích. Sloupec relace má svůj název, který je unikátní v rámci dané tabulky. Hodnoty, které se ve sloupci mohou nacházet, jsou určeny doménou sloupce. Ve striktním pojetí relací by řádek relace neměl obsahovat více hodnot pro jeden sloupec (atribut) a řádek by měl být v relaci unikátní. Každá kombinace atributů, pomocí které se dá jednoznačně identifikovat řádek relace, se nazývá kandidátní klíč. Primární klíč je jeden vybraný klíč z kandidátních klíčů, který se dále používá k identifikaci řádků. K reprezentaci vztahů mezi entitami se využívají cizí klíče. Cizí klíč je sloupec (nebo jejich skupina) v relaci, který odpovídá primárnímu klíči jiné entity. [3]

1.4 Normalizace datového modelu

Při tvorbě relačního modelu pro oblast s nějakým souborem entit a jejich atributů je vždy několik možností jak tyto entity převést na relace. Předcházení problémům způsobených špatně navrženým relačním modelem pomáhá proces normalizace. Při normalizaci se relační datový model upravuje a navrhuje tak, aby relace splňovaly určené normální formy. Normální forma je soubor pravidel, které relace musí dodržovat. Pro eliminaci většiny problémů plynoucí ze špatného návrhu relačního modelu ve většině případů stačí, aby relace splňovaly první, druhou a třetí normální formu (tyto 3 jsou relevantní pro tuto práci). Boyce-Coddova, čtvrtá, pátá a šestá normální forma specifikují pravidla, která zabraňují jen zřídka se vyskytujícím a specifickým případům (tyto formy nejsou v této práci pokryty). Tato pravidla jsou přísnější s každou další normální formou. Vyšší normální forma zahrnuje pravidla všech nižších normálních forem. [3]

Atribut, který nabývá více hodnot v jednom řádku, se nazývá opakující se skupina. Relace v první normální formě je taková, kde jsou data umístěna ve dvourozměrné tabulce a zároveň se v ní nevyskytují žádné opakující se skupiny. [3]

Funkční závislost je vztah mezi atributy A a B. Atribut B je funkčně závislý na atributu A, když se každá jedinečná hodnota atributu A vyskytuje v nějakém řádku společně pouze s jednou hodnotou atributu B. Relace splňuje druhou normální formu, když jsou všechny její atributy, které nejsou součástí klíče, funkčně závislé na celém primárním klíči a zároveň tato relace musí splňovat i první normální formu. [3]

Atributy jsou v tranzitivní závislosti, jestliže ze vztahů atribut B je funkčně závislý na atributu A a atribut C je funkčně závislý na atributu B plyne, že atribut C je funkčně závislý na A. Relace, která splňuje druhou normální formu a mezi jejími atributy není žádná tranzitivní závislost, je ve třetí normální formě. [3]

Při návrhu datových modelů, které splňují třetí normální formu, se cílí na odstranění redundancí v datech. Data v modelech ve třetí normální formě jsou rozdělena do oddělených entit, které sdílí co nejmenší množství dat. Těchto entit a jim odpovídajícím relačních tabulek pak bývá v rozsáhlých systémech velké množství. [3]

1.5 Denormalizace

Normalizace je proces převodu relací datového modelu do takového tvaru, aby relace splňovaly omezení udávané normální formou. Z povahy omezení normálních forem většinou normalizace vede k rozdělování tabulek s velkým počtem na sobě závislých atributů na více tabulek s malým počtem nezávislých atributů, které jsou na sebe vázané pomocí cizích klíčů. Tento přístup se hodí pro transakční zpracování dat, usnadňuje kontrolu integrity dat a zamezuje zbytečné duplikaci dat. Velké množství tabulek s malým počtem nezávislých atributů ale vede k potřebě spojovat data z více tabulek při analytických dotazech, což zvyšuje odezvu. Striktně normalizovaný datový model tedy není vhodný pro analytické databázové systémy a při návrhu tohoto datového modelu se využívá denormalizace. [1]

Denormalizace spočívá v přidávání atributů k relacím za účelem eliminace potřeby spojování více tabulek při dotazování. Nazývá se tak proto, že při tomto procesu dochází k porušování pravidel normálních forem a zavádí se redundance dat. Denormalizace probíhá přidáním atributů jedné relace k další relaci. Původní relace se buď může zachovat, nebo smazat. To vede k eliminaci spojování tabulek při dotazování, což snižuje dobu odezvy dotazů. Nevýhody, které denormalizace přináší jsou zvýšení objemu ukládaných dat a potenciální problémy se zachováním integrity dat při přidávání a mazání záznamů. [4]

1.6 Základní operace v relační algebře a dotazovacím jazyce SQL

SQL dotazy se skládají z jednotlivých operací, které databázový systém při vyhodnocování dotazů identifikuje a vytváří jejich vyhodnocováním výsledek. Tyto operace v dotazech v jazyce SQL jsou založeny na relační algebře. Každá z dále popsaných operací relační algebry má pouze jeden účel, žádná operace nedělá více věcí najednou. Výsledkem operace nad relační tabulkou je další relační tabulka. Databázový systém vyhodnocuje operace z dotazu sekvenčně jednu po druhé. Porozumění základním operacím relační algebry je užitečné pro pochopení toho, co se skrývá za SQL dotazem, což pomáhá k tvorbě efektivnějších dotazů a lepšímu návrhu datového modelu. Vyhodnocení jednotlivých operací v databázovém systému není vždy totožné s vyhodnocením výrazů relační algebry. Relace v relační algebře neobsahuje duplicitní řádky, tedy ani výsledek relační operace je nebude obsahovat. Oproti tomu většina databázových systémů odstraňuje duplicitní řádky ve výsledku pouze, pokud to uživatel specifikuje. [3]

Oprace, jejíž výsledkem je relace obsahující jeden nebo více vybraných sloupců z původní relace, se nazývá **projekce**. Odpovídajícím příkazem v dotazovacím jazyce SQL je **select**. [3]

Pro výběr řádků z původní relace na základě předloženého predikátu se používá operace **selekce**. Predikát je logický výraz složený například z podmínek na atributy původní relace. Vyhodnocení predikátu určuje, které řádky se do výsledné relace mají vybrat. K výběru řádků podle logického predikátu se v jazyce SQL využívá příkazu **where**. [3]

Pro získání relace obsahující řádky ze dvou relačních tabulek se používá **spojení**. Ve výsledné relaci po operaci spojení se nachází řádky z obou tabulek, které obsahují shodné hodnoty atributů, jejichž jména jsou stejná v obou tabulkách. Takovéto základní spojení se nazývá přirozené. Obecnější typ spojení se nazývá theta join, u kterého se data ze dvou tabulek spojí na základě specifikovaného predikátu. Spojení v dotazovacím jazyce SQL odpovídá příkaz **join**. [3]

Pro práci s více řádky najednou se hodí operace seskupování. Řádky tabulky se seskupí podle hodnot ve specifikovaném sloupci nebo skupině sloupců. Seskupování umožňuje vyhodnocení takzvaných agregačních funkcí (např. součet, počet, průměr, ...) nad seskupenými daty. Operace seskupení se v jazyce SQL provádí příkazem **group by**. [3]

1.7 Indexy, shlukování (clustering) a rozdělování (partitioning)

Aby byl databázový systém v praxi použitelný, je důležitá přiměřeně nízká odezva mezi zadáním dotazu uživatele a obržením odpovědi. Systémy řízení báze dat nabízejí různé nástroje pro úpravu návrhu databáze s účelem snížení odezvy odpovědi, jako jsou indexy, shlukování dat
a „rozdělování dat do přihrádek“ (anglicky partitioning)[1], které budou probrané v této kapitole. Procesu aplikace těchto nástrojů se říká optimalizace výkonu databáze nebo ladění výkonu. [3]

Přidávané řádky jsou ve většině databází zapisované připojením na konec tabulky. To vede k neuspořádanému rozložení hodnot v každém sloupci a jediná možnost, jak databázový systém může vyhledat hodnotu ve sloupci, je proskenovat všechny řádky. **Indexy** jsou struktury, díky kterým databázový systém může rychleji přistoupit k hodnotám ve sloupci nebo skupině
sloupců. [3]

Nejvíce časově náročnými operacemi, kterou při vyhodnocování dotazů systém řízení báze dat provádí, jsou diskové operace. Diskové operace zahrnují načítání dat z disku a jejich zápis na disk. Data jsou na disku organizovaná do stránek (jejich velikost záleží na konkrétním systému), při načtení konkrétního záznamu je načtena celá stránka. Z pohledu ladění výkonu lze z těchto poznatků vyvodit, že je výhodné ukládat data na disk tak, aby byly záznamy, které se často načítají společně, uložené nejlépe v jedné stránce. Toho lze v systému řízení dat dosáhnout využitím **shlukování** dat (anglicky clustering). Shlukování se definuje podle sloupce nebo skupiny sloupců. Řádky, které obsahují stejné hodnoty těchto sloupců jsou zapisovány na disk co nejblíže k sobě. [3]

Nástrojem, který se provádí postupem opačným ke shlukování, je **partitioning**. Data je možné rozdělit podle sloupce nebo skupiny sloupců. Data se pak rozdělují do „přihrádek“ (anlgicky partition)[2] podle hodnot těchto sloupců. Sloupce jsou vybírány tak, aby se při dotazování nečetla z disku a neprocházela celá tabulka, ale jen jedna nebo menší množství partition. [3]

[1] Dále v textu bude použit anglický název partitioning.
[2] Dále v textu bude použit anglický název parition.

1.8 Business Intelligence

Pojem business intelligence (BI) se v dnešní době (a kontextech jako je tato práce) objevuje v souvislosti se souborem konceptů, technik a počítačových systémů, které slouží k podpoře rozhodování o chodu organizací a podniků. Podpora rozhodnutí je většinou založena na datech, která vznikají při provozu organizace. Týmy provozující BI tato data přeměňují za pomocí různých technik a postupů ve znalosti. Dalším úkolem BI týmů je znalosti získané ze surových dat zpřístupnit a prezentovat těm, kteří mají na starosti samotné rozhodování o chodu podniku. [5]

Mezi činnosti spojené s podporou BI patří také návrh a údržba datového skladu a vývoj a obsluha systémů spojených s chodem datového skladu. Mezi tyto systémy patří samotné databázové systémy, kde jsou data ukládána. Dále jsou potřeba systémy pro zpracování a načítání dat a také techniky a nástroje pro jejich prezentaci a vizualizaci, kterým se říká BI aplikace. [5]

1.9 Datový sklad

V běžné organizaci se vyskytuje několik provozních systémů, které jsou vyvinuté a optimalizované pro rychlé zpracování transakcí či požadavků, které jsou většinou zpracovávané po jednom. Tyto systémy se zpravidla zaměřují na jednu konkrétní provozní oblast organizace, například zpracování objednávek, registrace zákazníků, nebo záznam zpětné vazby od zákazníka. Zde běžně není potřeba uchovávat kompletní historii, ale spíše udržovat aktuální informace. [6]

Informace jsou jedním z nejdůležitějších prostředků organizace. Použitelnost tohoto prostředku spočívá v uchovávání záznamů z provozu organizace a využití těchto záznamů pro podporu rozhodování při řízení organizace. Datový sklad je takový systém, který tyto informace z jednotlivých provozních systémů organizace sbírá, uchovává, zpracovává a poskytuje k nim přístup. Narozdíl od většiny provozních systémů se v datovém skladu uchovává dlouhodobá nebo kompletní historie záznamů o požadavcích a transakcích. V datovém skladu se většinou snažíme získat informace z provádění výpočtů a analýz nad mnoha transakcemi najednou. [6]

Údaje z různých systémů mohou obsahovat totožné informace, které jsou jinak strukturované či popsané. Jedním z účelů datového skladu je poskytnout jednotný a platný pohled na tyto informace, aby se daly využít pro podporu řízení organizace. V datovém skladu je potřeba zajistit konzistenci významu informací a metrik napříč různými systémy. Tedy například dvě metriky se stejným názvem by měly uvádět tu samou informaci. Informace v datovém skladu musí být konzistentní a důvěryhodné, to je zajištěno čistěním a ověřením kvality údajů posbíraných z dalších systémů organizace. Obsah datového skladu by měl být srozumitelný pro jeho uživatele. Názvy a popisy údajů poskytovaných uživatelům datového skladu by měly odpovídat obchodním názvům a významům příslušných údajů. Jedním z častých požadavků na datový sklad je, aby se požadované informace dostaly k uživatelům co nejrychleji. [6]

1.10 Dimenzionální modelování

Dimenzionální modelování je jedním z přístupů k vytváření datového modelu. Tento datový model je navrhován takovým způsobem, aby dotazy nad ním byly rychle vyhodnoceny a aby výsledná data byla srozumitelná pro koncové uživatele. Tento přístup staví na jednoduchosti návrhu, ze které plyne srozumitelnost datového modelu. Jednoduchý datový model umožňuje efektivní zpracování databázovým systémem při dotazování. Z těchto důvodů je tento přístup často využívaným způsobem při navrhování datových modelů pro prezentaci analytických dat. Modely ve třetí normální formě se hodí pro zpracování dat v provozu. Přidání a úprava dat se většinou týká jen jedné oddělené entity. Ovšem pro potřeby datových skladů a analytického dotazování jsou modely ve třetí normální formě příliš komplexní. Komplexnost modelu dělá složitější práci databázovému systému při vyhodnocování dotazů nad ním a vede k delší odezvě. Také pro koncové uživatele je složitější dotazy nad komplexním modelem tvořit. Dimenzionální modely obsahují 2 typy tabulek, tabulky faktové a tabulky dimenzionální. [6]

1.10.1 Faktová tabulka

Faktová tabulka obsahuje fakta, která odpovídají hodnotám naměřeným a zaznamenaným v provozu organizace. Nejužitečnější fakta jsou numerická a aditivní, jako například počty událostí nebo objemy transakcí. Každý řádek této tabulky odpovídá nějaké měřitelné události. Hlavním cílem analytických dotazů nad datovým skladem není práce s jedním konkrétním řádkem, ale s velkým množstvím řádků, proto je možnost fakta sčítat plynoucí z aditivity tak užitečná. Dalšími sloupci ve faktové tabulce jsou cizí klíče do dimenzionálních tabulek. Jiné údaje, které nejsou unikátní pro každý řádek, by se ve faktové tabulce neměly vyskytovat. V praxi mají tyto tabulky velký počet řádků a malý počet sloupců. Tato měřená data je doporučované ukládat pouze v jednom modelu, jelikož zpravidla dosahují největších objemů mezi daty organizace. Uložením faktových dat v jednom modelu také napomáhá konzistenci a sjednocení pohledů na tato data z různých oddělení organizace. [6]

1.10.2 Tabulka dimenzí

Tabulky dimenzí doplňují faktové tabulky. V tabulkách dimenzí se ukládají data, která popisují měřené události ve faktových tabulkách a doplňují k nim další informace. S pomocí dat z tabulek dimenzí se dá odpovědět na otázky o faktech typu kdy, kde, kdo a jak. Tabulky dimenzí obsahují malé počty řádků v porovnání s faktovými tabulkami a velké množství atributů. V analytických dotazech se podle atributů dimenzí data z faktových tabulek filtrují nebo seskupují. Pro zachování srozumitelnosti pro koncové uživatele by názvy a významy atributů dimenzí měly odpovídat těm v praxi používaným v dané organizaci. V praxi se tabulky dimenzí navrhují tak, že obsahují redundantní a duplikované údaje, nejsou tedy normalizované. To znamená, že jsou redundantní údaje uložené v jedné tabulce a nedochází tak ke spojování více tabulek. To usnadňuje jak dotazování koncových uživatelů nad daným modelem, tak práci databázového systému při vyhodnocování dotazů. [6]

1.10.3 Schéma tvaru hvězda

Nejjednodušší dimenzionální model často používaný v relačních systémech řízení báze dat je schéma ve tvaru hvězdy. Takové schéma je tvořené jednou faktovou tabulkou napojenou na jednu nebo více tabulek dimenzí. Další vazby v tomto schématu nejsou. Schéma ve tvaru hvězdy za dodržení srozumitelných popisů atributů dimenzí svou jednoduchostí podporuje vysokou přehlednost a snadné pochopení datového modelu koncovými uživateli. Malé množství vazeb v datovém modelu podporuje efektivitu při vyhodnocování dotazů, protože nedochází ke spojování údajů z mnoha tabulek. [6]

1.11 Architektura podle Ralpha Kimballa

Datový sklad se podle architektury Ralpha Kimballa skládá ze čtyř oddělených komponent. Tyto komponenty jsou zdrojové provozní systémy, systém ETL (extract, transform, load; z angličtiny extrahovat, transformovat a nahrát), vrstva pro prezentaci dat a BI aplikace. Zdrojové provozní systémy jsou ostatní provozní systémy v rámci organizace, ve kterých se odehrávají a zaznamenávají provozní transakce. Tyto systémy jsou na datovém skladu nezávislé, nejsou jím ovlivňovány ani řízeny. Do datového skladu se z nich pouze získávají data. Cílem těchto systémů není uchovávat data z dlouhodobé historie a zpracovávat analytické dotazy nad nimi. Naopak dobře navržený datový sklad těmto systémům ulevuje od zátěže a stará se o uložení historických dat a jejich analýzou. Jednotlivé provozní systémy na sobě také mohou být nezávislé a tak se často v praxi mezi různými provozními systémy nesdílí jednotná pojmenování ukládaných entit a jejich atributů. [6]

ETL systém v Kimballově architektuře slouží konkrétně ke zpracování dat ze zdrojových provozních systémů a jejich přenesení do vrstvy prezentace dat. Tato transformace probíhá ve třech krocích, kterými jsou extrakce, transformace a načtení dat. Obecně se v ostatních architekturách ETL systémy využívají ke zpracování a přesunu dat mezi dvěma vrstvami architektury datového skladu. První fází, ve které se data dostanou do systému datového skladu, je extrakce. V této fázi se načítají data ze zdrojových provozních systémů a požadovaná data jsou zkopírována do datového skladu. V další fázi, fázi transformace, se podle potřeby provádějí různé úpravy extrahovaných dat. Během transformace dochází například k čištění dat, tedy procesu, který se snaží odstranit chyby v datech nebo doplnit chybějící hodnoty. Dále se data mohou obohacovat o data z jiných systémů nebo s nimi spojovat. Třetí fáze je načtení dat do cílové vrstvy. V případě Kimballovy architektury se jedná o vrstvu prezentace dat. V této fázi se data fyzicky uloží v požadované formě odpovídající dimenzionálnímu datovému modelu, který je vytvořen na základě nějakého obchodního procesu. [6]

Ve vrstvě pro prezentaci dat jsou data uložena v dimenzionálním datovém modelu a připravena pro přímé dotazování koncovými uživateli nebo jako zdroj pro BI aplikace. BI aplikace nabízí možnosti, jak data z vrstvy pro prezentaci dat využít pro analýzy podporující obchodní rozhodnutí. Mezi BI aplikace patří třeba jednoduché rozhraní, které umožní uživateli zadat dotaz do databáze, vizualizační nástroj nebo systémy podporující vývoj statistických modelů. [6]

1.12 Architektura nezávislých datových tržišť

Datové tržiště je vrstva datového skladu, ve které se nachází již zpracovaná data vybraná na základě požadavků koncových uživatelů. Protože koncoví uživatelé mohou být z různých oddělení, každého z nich zajímá jiná doména zpracovaných dat nebo požadují odlišné výpočty nad stejnými daty. Z tohoto důvodu se podle této architektury prosazuje vytváření více nezávislých datových tržišť, která splňují požadavky jednotlivých oddělení společnosti. Takový postup přináší přehledný přístup k datům koncovým uživatelům, jelikož ve svém datovém tržišti vidí pouze ta data, která si vyžádali. [6]

Datová tržiště čerpají data přímo z uložených dat ze zdrojových provozních systémů. Mezi datovým tržištěm a daty ze zdrojových systémů se nenachází žádná vrstva, která by se starala o integraci dat za účelem poskytnutí sjednoceného pohledu na tato data. To přináší několik problémů. Prvním z nich je možnost nejednotného pohledu na ta samá data napříč odděleními. Dalším problémem z dlouhodobého hlediska jsou redundatně ukládaná analytická data, která vznikají, když má více oddělení zájem o jednu stejnou datovou doménu. [6]

1.13 Architektura Hub-and-Spoke

Tato architektura se skládá z více datových vrstev než předchozí dvě architektury. Data ze zdrojových systémů jsou nejdříve nahrána do vrstvy pro příjem dat. V této vrstvě se data ukládají většinou bez transformací. Další vrstva, podnikový datový sklad, tvoří jádro datového skladu. Zde jsou data zpracovaná z předchozí vrstvy, ukládaná atomicky a ve třetí normální formě. Klade se důraz na integraci dat se stejným významem, která ale pochází z různých zdrojových systémů, za účelem sjednocení pohledu na tato data. Koncovým uživatelům je většinou umožněno dotazovat se i přímo na data z této vrstvy. Dále se data z podnikového datového skladu transformují podle potřeb oddělení a ukládají se do datových tržišť. [6]

1.14 Hybridní architektura

Tato architektura je kombinací architektury podle Kimballa a architektury Hub-and-Spoke. Data ze zdrojových systémů jsou zpracovávaná do normalizovaných tabulek v podnikovém datovém skladu. Od Hub-and-Spoke architektury se tato vrstva liší v tom, že do ní nemají přístup koncoví uživatelé. Ti přistupují k datům pomocí vrstvy pro prezentaci dat, kde jsou data zpracovaná do dimenzionálních datových modelů na základě obchodních procesů jako v architektuře podle Kimballa. O přesun dat mezi jednotlivými vrstvami a jejich zpracování se starají ETL systémy. Na vrstvu pro prezentaci dat jsou napojené BI aplikace. [6]

1.15 NoSQL databázové systémy

V této kapitole popisuji obecné vlastnosti a význam NoSQL databázových systémů. Vysvětluji zde také základní a obecné principy několika druhů NoSQL databázových systémů. Konkrétní nástroje, které tyto principy implementují, jsou popsané v kapitole 3 popisující technologie jako možná řešení této práce.

Relační databázové systémy se se svou podporou ACID vlastností transakcí a pevně definovaným datovým modelem skvěle hodí pro transakční zpracování dat a kontrolu jejich integrity. S narůstajícími objemy dat a rostoucím zájmem o tvorbu analytických dotazů nad těmito daty se začaly objevovat požadavky na databázové systémy, na které ty relační přestaly stačit. Mezi tyto požadavky patří například možnost distribuce výpočtů a dat na více strojů, zejména dostupných a ne specializovaných. Žádoucí je možnost v případě potřeby navýšit výkon systému přidáním dalšího stroje. To se nazývá horizontální škálování. Začalo se také objevovat velké množství dat, které se v běžné relační databázi nedalo ukládat. Tato data se označují jako nestrukturovaná a patří mezi ně například zvukové nebo obrazové záznamy. [7]

Proto se začaly objevovat nové databázové systémy, které nabízely řešení těchto problémů a nové způsoby, jak data ukládat například pro účely efektivního analytického zpracování. Tyto nové databázové systémy nedodržovaly principy relačních databází, jako například zajištění ACID vlastností transakcí nebo pevně definované schéma datového modelu. Kvůli svým odlišnostem oproti relačním databázím, které používají dotazovací jazyk SQL, se začaly souhrnně nazývat jako NoSQL databázové systémy. [7]

Mezi NoSQL systémy se řadí několik různých skupin databázových systémů, které se liší zejména způsobem tvorby a ukládání datového modelu a v datových strukturách používaných v datovém modelu. Typy NoSQL databázových systémů, které jsou probírány v této práci, jsou zejména sloupcově orientované databáze a dokumentové databázové systémy. [8]

1.15.1 Sloupcově orientované databáze

V tradičních relačních databázových systémech se data ukládají po řádcích a jsou řádkově orientované. Data z jednoho řádku tabulky se na disk zapisují k sobě. Tento přístup se hodí zejména pro transakční zpracování dat, kdy se přistupuje k jednomu záznamu a většině jeho atributů v daný čas. V případě datových skladů a analytických dotazů se naopak většinou přistupuje k jednomu ze sloupců a všem jeho hodnotám v řádcích. [7]

Základní myšlenkou sloupcově orientovaných databázových systémů je zapisování dat z jednoho sloupce k sobě na disk. Velkou výhodou tohoto přístupu k ukládání dat je výkon při dotazech s agregačními funkcemi nad všemi hodnotami z některého z atributů tabulky. Výsledek agregační funkce lze vypočítat efektivně, protože jsou data ze sloupce uložena na disku blízko sebe a tak se snižuje potřebný počet načtených datových bloků z disku. Další z výhod sloupcově orientovaných databázových systémů je podpora efektivní komprese dat a to jak z pohledu ušetřeného místa na disku, tak rychlosti provedení komprese. Důvodem je, že hodnoty v jednom sloupci se často opakují, čehož některé kompresní algoritmy dokáží efektivně využít a docílí menší velikosti komprimovaných dat. Rychlosti zpracování kompresními algoritmy napomáhá blízkost uložených dat na disku, takže je při zpracování potřeba načíst malé množství datových bloků z disku. Sloupcově orientované databázové systémy jsou ale nevhodné při práci se všemi atributy jednoho řádku. Data z jednoho řádku je při dotazu na ně potřeba poskládat ze sloupcově orientovaného uložení dat. To vede ke čtení několika datových bloků z disku. Problémy představuje také zápis jednotlivých řádků, protože se při něm nelze vyhnout přístupu ke všem uloženým sloupcům a tedy přístupu k velkému počtu datových bloků na disku. [7]

1.15.2 Dokumentové databáze

Databázové systémy, které ukládají data pomocí dokumentů strukturovaných podle formátů jako JSON nebo XML, se nazývají dokumentové databáze. Od relačních databází se liší hlavně ve formátu ukládaných dat a struktuře datového modelu, ostatní vlastnosti mohou mít shodné. Některé z dokumentových databází například podporují ACID vlastnosti transakcí. Dokumentové databáze využívající formát JSON se ujaly kvůli snadnému převodu objektů ze světa objektově orientovaného programování do dokumentů formátu JSON. Výhodou uložení dat do strukturovaného dokumentu je to, že jsou data i jejich popis uloženy v jednom souboru a dají se tak přečíst a interpretovat nezávisle na použitém programu. Dokumentové databáze se svým modelem ukládání dat nabízí kompromis mezi databázovými systémy, které striktně vyžadují pevně definované schéma, a mezi systémy, které ukládají plně nestrukturovaná data bez schématu. [7]

1.15.2.1 Formáty XML a JSON

Formáty JSON a XML se používají ke strukturování dokumentů tak, aby byly jak čitelné člověkem, tak snadno strojově zpracovatelné. Oba slouží k uložení jak samotného obsahu dokumentu, tak informací popisujících, co jednotlivé části dokumentu znamenají. [9]

XML dokument je tvořen vlastním obsahem, který je ohraničen prvky popisujícími logickou strukturu dokumentu, těm se říká tag. Tag je uzavřen do špičatých závorek, je pojmenován a může mít i další popisné atributy reprezentované dvojicí jméno atributu a hodnota. XML dokument se dá popsat stromovou strukturou, kde uzly tohoto stromu jsou jednotlivé prvky popsané pomocí tagů. [9]

JSON popisuje objekty za pomoci seznamu dvojic tvořených názvem a vlastním obsahem. Tyto dvojice tvoří a popisují vlastnosti objektu. [9]

Kapitola 2

Analýza požadavků a aktuálního řešení

2.1 Aktuální stav datového skladu ve VZP

V této sekci poskytuji informace o aktuální architektuře datového skladu ve VZP. Jedná se o souhrn informací, které jsou relevantní pro ujasnění kontextu této práce. Dále uvádím používané nástroje a technologie, z jejichž znalosti plynou další požadavky na řešení.

Datový sklad ve společnosti VZP slouží zejména pro odbavování analytických dotazů nad daty v něm uložených. Tyto analytické dotazy provádějí byznys analytici přes rozhraní poskytnuté BI vrstvou. Architektura části datového skladu ve VZP, která je předmětem této práce, odpovídá architektuře podle Ralpha Kimballa (sekce 1.11). Tato část datového skladu je tvořena vrstvou, která obsahuje data ze zdrojových systémů. V této vrstvě se neuchovává dlouhodobá historie. Dále jsou data načtena do stage vrstvy, která slouží pro provádění datových transformací a přípravu dat pro nahrání do poslední vrstvy. Třetí vrstvu tvoří prezentační vrstva, která přímo poskytuje data do BI aplikace. Všechny vrstvy využívají databázový systém Oracle [10]. Tomu je přizpůsobený také hardware na testovacím serveru, který se skládá z jednoho výkonného specializovaného stroje.[3]

[3] Z konzultací s Ing. Kristiánem Vadkertim, zadavatelem práce.

BI aplikace je napojena na prezentační vrstvu datového skladu, která jí slouží jako datový zdroj. Vrstva BI aplikace je tvořena zejména nástrojem Oracle Analytics Server (dříve Oracle Business Intelligence).[4]

Oracle Analytics Server je nástroj pro podporu analyzování a získávání znalostí z dat. Nástroj umožňuje tvořit dotazy nad dostupnými daty a tato data zobrazovat ve formě tabulek a grafů. [11]

Schémata dat, která jsou uložená ve vrstvě datového tržiště datového skladu, jsou tvořena podle principů dimenzionálního modelování dat. Ta podporují analytické dotazování nad uloženými daty. Tabulky v datovém tržišti obsahují denormalizovaná data a tvoří schémata ve tvaru hvězdy. Datová tržiště tedy obsahují faktové tabulky, které jsou propojené s tabulkami dimenzí.[5]

[4] Z konzultací s Ing. Kristiánem Vadkertim, zadavatelem práce.
[5] Z konzultací s Ing. Kristiánem Vadkertim, zadavatelem práce.

2.2 Popis dat

Pro tuto práci byla poskytnuta data z datového tržiště se schématem ve tvaru hvězdy. Schéma je tvořeno jednou faktovou tabulkou a devíti tabulkami dimenzí. Faktová tabulka obsahuje informace z řádků dokladů, které obsahují informace o cenových složkách a objemu vykázané péče. Tabulky dimenzí obsahují doplňující data k faktové tabulce. Faktová tabulka je pomocí cizích klíčů propojená s tabulkami dimenzí. Jiná propojení ve schématu nejsou. Poskytnutá data byla anonymizována, v datech tedy nejsou obsaženy žádné skutečné citlivé údaje.[6]

2.3 Požadavky na řešení

Požadavkem na tuto práci ze strany zadavatele je zvolit NoSQL databázové technologie a použít je pro uložení dat ve vrstvě datového skladu, která poskytuje data BI aplikaci. Tato řešení budou porovnána z pohledu odezvy na dotazy s aktuálně používaným databázovým systémem Oracle. Dotazy budou vyhodnoceny nad poskytnutými daty ve schématu ve tvaru hvězdy. Společně s daty byla poskytnuta také sada typických dotazů, na kterých bude měření provedeno.[7]

Takto zvolená databázová technologie by měla být vhodná pro uložení dat v datovém tržišti jako zdroj dat pro BI aplikaci. Dále by zvolená technologie měla být vhodná pro vyhodnocování dotazů typově podobných těm poskytnutým.

Z popisu architektury a využitých nástrojů v současném řešení datového skladu plyne první požadavek na technologie vybrané k porovnání. Tímto požadavkem je možnost připojení Oracle Analytics Serveru na vybranou databázovou technologii.

Dalším požadavkem ze strany zadavatele je, aby zvolené technologie byly dostupné pod open-source licencí.

[6] Z konzultací s Ing. Kristiánem Vadkertim, zadavatelem práce.
[7] Z konzultací s Ing. Kristiánem Vadkertim, zadavatelem práce.

2.4 Analýza dotazů

Spolu s daty byla ze strany VZP poskytnuta i sada běžných dotazů, které byznys analytici tvoří pomocí nástroje Oracle Analytics Server. Tato sada obsahuje 100 dotazů, které reprezentují běžně pokládané dotazy nad poskytnutým schématem v datovém tržišti. Tři z poskytnutých dotazů jsou dostupné v Příloze C.

Dotazy se provádějí nad tabulkami s poskytnutými daty, jejichž schéma odpovídá schématu tvaru hvězdy. Schéma je tvořeno jednou faktovou tabulkou a devíti tabulkami dimenzí. Po analyzování poskytnuté sady dotazů jsem zjistil následující společné vlastnosti.

- Data z faktové tabulky se spojují s daty z několika tabulek dimenzí.
- Dochází k filtrování dat na základě atributů z tabulek dimenzí.
- Filtrovaná a spojená data se seskupují podle atributů z tabulek dimenzí.
- Provádějí se výpočty agregačních funkcí (např. součty a počty) nad takto seskupenými daty.

Uživatelé Oracle Analytics Server nejsou žádným způsobem omezeni při tvorbě dotazů v tomto systému. To znamená, že se mohou dotazovat na data z různých kombinací tabulek. Ve svých dotazech mohou využívat pro selekci a seskupování jakýkoliv atribut z jakékoliv tabulky dimenzí. To má za následek následující vlastnosti dotazů.

- Napříč dotazy se vyskytuje velké množství různých kombinací spojovaných tabulek.
  - Celkem 51 různých kombinací tabulek dimenzí při spojování s faktovou tabulkou.
  - Ve 100 dotazech bylo celkem využito 623 tabulek pro výběr dat. To znamená, že se v průměru vyskytuje zhruba 6 spojovaných tabulek v každém dotazu.
- V podmínkách pro filtrování dat se vyskytuje mnoho různých atributů z různých tabulek.
- Atributy, podle kterých se data seskupují, jsou napříč dotazy různé a tvoří různé kombinace.
  - Napříč 100 poskytnutými dotazy se provádí seskupení podle 62 různých kombinací atributů.
  - Z těchto 62 kombinací atributů je 38 takových, že množina použitých atributů není podmnožinou žádné jiné kombinace.

2.5 Poskytnutý hardware

Pro účely této práce byl poskytnutý server pro provedení experimentů. Na tomto serveru jsem instaloval a spouštěl obě dvě řešení a prováděl na něm měření pro porovnání výkonu.

Server má následující hardware a software specifikace.

- Operační systém: CentOS Linux x86_64
- Kapacita RAM: 256 GB
- Kapacita diskového úložiště: 2 TB
- Procesor: Intel Xeon, 10 jader

Kapitola 3

Rešerše možných řešení

Technologie jako kandidáty na řešení jsem vybíral podle dříve definovaných požadavků v sekci 2.3. V potaz jsem bral zejména následující vlastnosti.

- Jedná se o NoSQL technologii. Nevybíral jsem tedy z relačních databází.
- Jedná se o technologii s open-source licencí.
- Oracle Analytics Server podporuje propojení s danou technologií.

Ze seznamu technologií, které je možné propojit se systémem Oracle Analytics Server jsem tedy vybral technologie splňující další 2 kritéria. Tedy ty patřící mezi NoSQL technologie s open-source licencí. Takto vybrané technologie jsem dále studoval, seznámil jsem se s jejich základními vlastnostmi a možnostmi. Dále jsem studoval další zdroje, ve kterých byly popsané různé experimenty s použitím těchto technologií pro podporu analytických dotazů v datových skladech. Na základě těchto zdrojů jsem posuzoval vhodnost jednotlivých řešení. Procházel jsem informace o těchto technologiích:

- Apache Cassandra
- Apache Hive (a Apache Hadoop)
- Apache HBase
- Apache Impala (a Parquet)
- MongoDB

Popis těchto technologií spolu s posouzením jejich vhodnosti pro řešení této práce na základě dalších zdrojů následuje v dalších podkapitolách.

3.1 Apache Cassandra

Apache Cassandra (dále jen Cassandra) patří mezi NoSQL databáze. Je to distribuovaná databáze a je pod open-source licencí. Byla navržena tak, aby dokázala uspokojit požadavky na ukládání velkých objemů dat a také na rychlé odbavení velkého množství dotazů nad nimi. Další vlastností, kterou Cassandra nabízí, je horizontální škálovatelnost, tedy možnost navýšit výkon pomocí přidání dalších výpočetních serverů. Účelem Cassandry bylo navrhnout databázový systém, který dokáže odbavit dotazy s krátkou odezvou a splňuje vysokou dostupnost. Vysoce dostupné databázové systémy by měly zvládnout odpovědět na požadavek uživatele po většinu času jejich běhu. Nemělo by tedy docházet k nedostupnosti systému z důvodu výpadku. [12]

Cassandra je distribuovaná databáze, která ukládá data mezi více uzlů. V databázi Cassandra není vynucovaná pevná struktura tabulek. Cassandra nepodporuje cizí klíče a kontrolu referenční integrity. Příkazy databáze Cassandra jsou zadávány v jazyce Cassandra Query Language (CQL). Tento základní dotazovací jazyk nepodporuje operace pro spojování dat z více tabulek na straně serveru. O spojování dat se musí případně postarat klient. Není podporován ani výpočet agregačních funkcí tak jako v relačních databázových systémech. [12]

Datový model databáze Cassandra je navržen pro tvorbu široko-sloupcových tabulek. To jsou tabulky, které obsahují velké množství sloupců. Data v Cassandra jsou rozdělena do tabulek. Více tabulek může být uloženo v keyspace. Na úrovni keyspace se například specifikuje, jak budou data tabulek v daném keyspace replikovaná. Data v tabulkách jsou dělena do partition. Partition je určena hodnotou atributu, který tvoří povinnou část primárního klíče řádků. Hodnota partition určuje mimo jiné to, na jakém uzlu bude daný řádek uložen. Partition shlukuje řádky tabulky. Řádky tabulek obsahují hodnoty sloupců a jsou identifikované jednoznačným primárním klíčem. [12]

Z vlasností a možností, které Cassandra nabízí, plyne, že modelování dat pro tuto databázi by mělo být řízeno požadavky na dotazy. Tuto metodologii pro návrh modelu dat, který je řízený vlastnostmi dotazů, ve své práci popisují Artem Chebotko, Andrey Kashlev a Shiyong Lu. [13] Data by měla být modelována podle toho, jak se na ně uživatelé budou dotazovat. Protože Cassandra nepodporuje spojování dat z více tabulek, musí se uživatelé už při návrhu tabulek zaměřit na to, aby atributy, které mají být společně na výstupu jednoho dotazu, byly uloženy v jedné tabulce. Takový přístup k modelování vede k denormalizaci a duplicitě dat.

Důležitou součástí modelování dat v Cassandra je návrh primárního klíče, zejména jeho části, která určuje rozdělení do partition. Tato část určuje rozdělení dat mezi uzly, na které Cassandra data ukládá. Je dobré volit takové klíče, aby byla podle nich data rovnoměrně rozložena na všech uzlech. Rozdělení mezi jednotlivé uzly je nutné brát v potaz i při tvorbě dotazů. Data, která se čtou v rámci jednoho dotazu, by měla být rozdělena mezi co nejméně partition a uzlů. Čím méně partition je čteno v rámci jednoho dotazu, tím kratší je odezva odpovědi. Další atributy (po klíči určujícím partitioning) tvořící primární klíč určují pořadí dat v rámci partition. Primární klíč je tvořen atributem určujícím partitioning a dalšími volitelnými atributy, které určují pořadí dat v rámci partition. [13]

Dotazovací jazyk CQL má také omezení na podmínky pro výběr dat. Predikáty pro výběr v dotazech mohou obsahovat pouze atributy, které jsou součástí primárního klíče. Atributy, podle kterých se určuje partitioning, musí být součástí predikátů pro podmíněný výběr dat. Pokud je použit v predikátu atribut, který je součástí složeného klíče a určuje pořadí uložených dat, musí být součástí i všechny ostatní atributy, které tomuto atributu předchází v definici primárního klíče. Dotazovací jazyk Cassandra nepodporuje příkazy pro řazení dat. Seřazení dat v rámci výsledné odpovědi na dotaz je určeno pouze částí primírního klíče, která určuje pořadí uložených dat v rámci partition. Nad ostatními atributy, které nejsou součástí primárního klíče lze postavit pomocné indexy, které napomáhají efektivnějšímu vyhodnocování dotazů na hodnoty těchto atributů. [13]

Z uvedených vlastností databáze Cassandra a zjištěných požadavků na dotazy z jejich analýzy v sekci 2.4 plyne, že samostatná databáze Cassandra ve své základní podobě není vhodné řešení pro tuto práci.

To plyne zejména z velké volnosti uživatelů při tvoření velice variabilních dotazů s různými požadavky na spojované zdrojové tabulky a lišícími se atributy pro filtrování nebo seskupování dat. Kvůli tomu, že není možné identifikovat pevně danou a omezenou sadu dotazů, není možné navrhnout tabulky v Cassandra, ze kterých by bylo možné na dotazy efektivně odpovídat.

Problémem při modelování dat v Cassandra v tomto případě je v kombinaci s volností v dotazování také větší množství tabulek ve vzorovém poskytnutém schématu. Případný datový model, ve kterém by se nacházely všechny možné kombinace faktové tabulky s atributy tabulek dimenzí, by byl řešením, které by mohlo vést k efektivnímu vyhodnocení dotazů. Jelikož by těchto kombinací bylo mnoho, byla by duplicita dat, zejména těch z faktové tabulky, příliš velká na to, aby bylo rozumné a možné takové množství dat ukládat.

3.2 Apache Hive

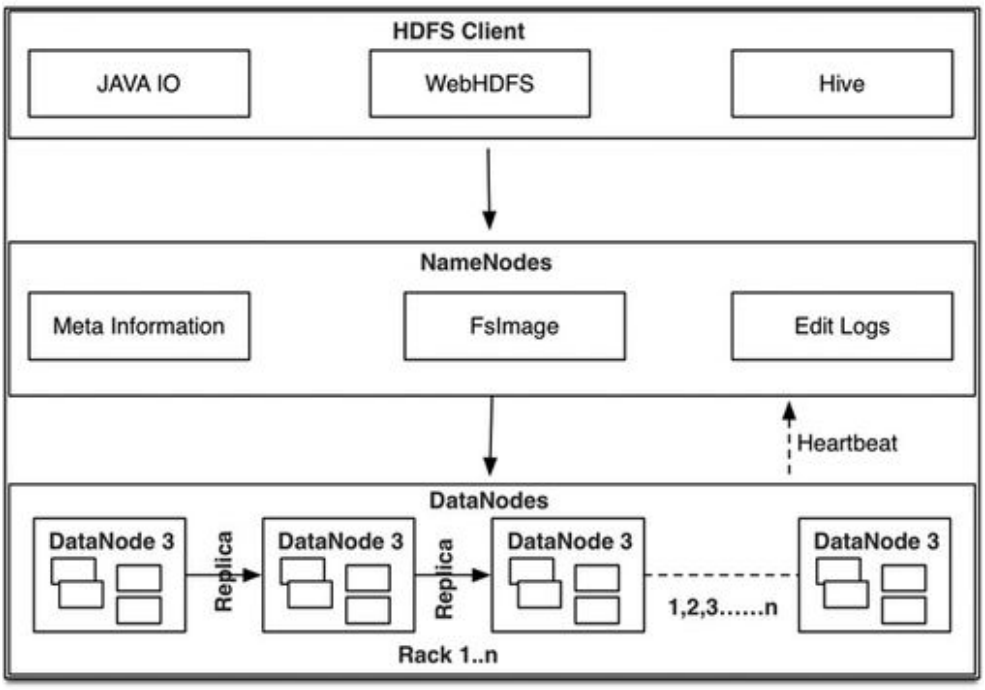
Apache Hive (dále jen Hive) je open-source software pro podporu činností datového skladu. Pomocí Hive je možné načítat a zapisovat velký objem dat do distribuovaného úložiště. Tyto operace je možné zadávat pomocí jazyka Hive Query Language (HQL), který se z velké části shoduje s jazykem SQL. Hive umožňuje vytvářet struktury (schémata tabulek) nad daty v různých formátech, která jsou již zapsána v úložišti (například jako výsledky výpočtů jiných aplikací). [14]

Hive je úzce propojen s Apache Hadoop (dále jen Hadoop). Hadoop je open-source projekt, který zahrnuje knihovny a nástroje pro distribuované výpočty a práci s daty. [15]

3.2.1 HDFS

Hadoop zahrnuje distribuovaný systém souborů Hadoop Distributed File System (HDFS). HDFS je navržen pro použití na více nespecializovaných a dostupných výpočetních strojích. Zajišťuje odolnost proti výpadkům jednotlivých strojů a ztrátě dat. Toho dosahuje mimojiné díky replikaci dat na různé uzly. HDFS se běžně využívá k ukládání velkých objemů dat a paměťově objemných souborů. HDFS nedovoluje náhodné úpravy uložených souborů, pouze připsání dat na konec souboru. Výkon HDFS je podporován principem, který udává, že výpočty nad daty by měly být prováděné lokálně tam, kde jsou data uložena. Namísto toho, aby se data přesouvala tam, kde běží aplikace provádějící výpočty. [16]

V distribuovaném prostředí jsou spuštěné instance HDFS (i ostatních nástrojů z projektu Hadoop) na jednotlivých strojích (uzlech). U HDFS se uzly dělí podle funkcionality na NameNode a DataNode. Hlavní NameNode běží pouze jeden a udržuje v sobě strukturu systému souborů. Aby nebyla funkčnost celého distribuovaného systému závislá pouze na jednom běžícím NameNode, je možnost udržovat sekundární NameNode, který v případě potřeby převezme funkcionalitu toho hlavního. DataNode uzly slouží pro vlastní ukládání dat. NameNode dále udržuje informace o jednotlivých souborech a o tom, na jakých uzlech (DataNode) jsou fyzicky uložené a kde jsou uložené jejich repliky. [15]



Obrázek 3.1: Obecný pohled na architekturu HDFS. Převzato z [15].

Na obrázku 3.1 výše je vyobrazena základní architektura HDFS. Komponenty v horní části obrázku jsou klientské aplikace, které k HDFS přistupují. Uprostřed je struktura NameNode, kde jsou uchované meta informace, obraz systému souborů a logy. NameNode je napojený na několik DataNode, které jsou ve spodní části obrázku a je na nich naznačena replikace uložených dat. [15]

3.2.2 MapReduce a YARN

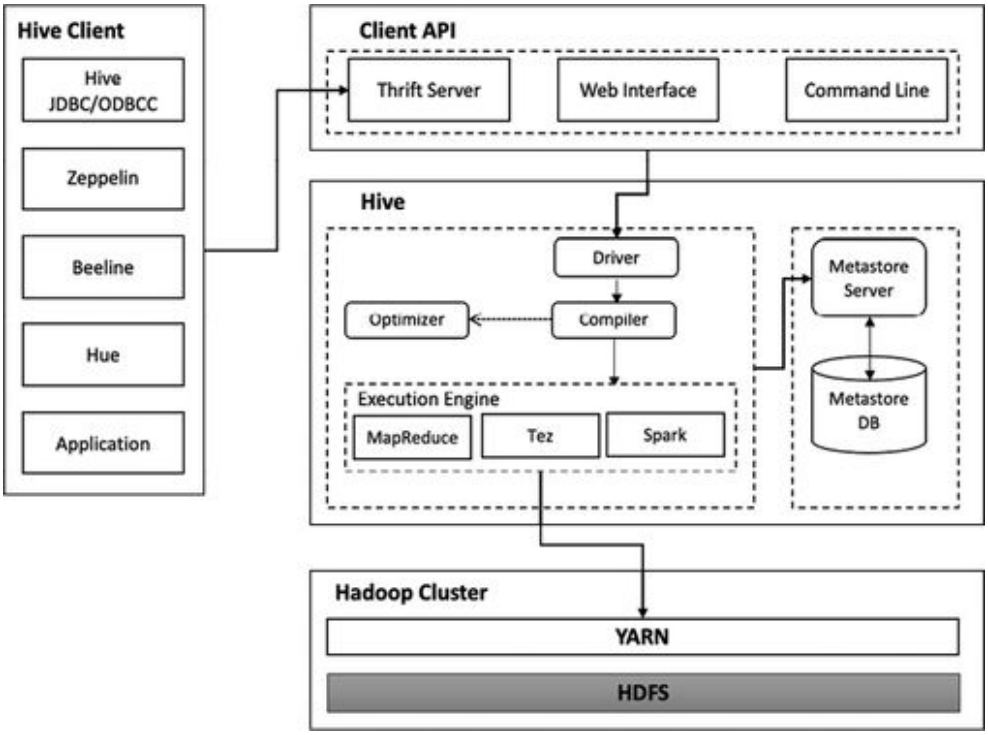
Další software v rámci Hadoop je MapReduce, což je rozhraní pro vytváření aplikací pro paralelní zpracování velkých objemů dat. MapReduce sdílí základní vlastnosti s HDFS, kterými jsou distribuování výpočtů na velké množství nespecializovaných a dostupných výpočetních strojů, kde zajišťuje odolnost proti výpadkům jednotlivých strojů. [17]

Pro přiřazování výpočetních prostředků ve výpočetním prostředí Hadoop slouží YARN. YARN je nástroj, který se stará o rozdělování výpočetních prostředků. Dalším účelem YARN je plánování a rozvrhování výpočetních operací, například výpočtů pomocí MapReduce. [18]

3.2.3 Výpočetní a datový model Hive

Pro účely této práce jsem se zaměřil na takové nastavení Hive, které využívá komponenty Hadoop. Hive usnadňuje analyzování dat a provádění výpočtů nad nimi s pomocí výpočetního modelu MapReduce tím, že do něj převádí uživatelské dotazy. Příkazy pro tvoření analytických dotazů jsou Hive předávané v jazyce Hive Query Language, který je z velké části shodný s jazykem SQL. Hive tyto příkazy překládá na kód pro výpočty MapReduce. Hive je možné nastavit tak, aby využíval pro provádění výpočtů i další výpočetní nástroje jako Apache Spark. [19] Indexy pro zvýšení efektivity při čtení dat nejsou v nových verzích Hive podporovány. Pro zvýšení efektivity při čtení dat je pro uložení možné využít k tomu navržené formáty souborů. Hive se hodí pro dávkové zpracování velkých objemů dat, při kterém nezáleží na příliš rychlé odezvě. [15]

Hlavním datovým objektem v Hive je tabulka. Data obsažena v tabulce mohou být dále dělena do partition na základě jednoho nebo více atributů. Při použití Hive spolu s distribuovaným systémem souborů HDFS se udržují informace o mapování tabulek a jejich partition na adresáře v HDFS. Hive podporuje řadu formátů pro ukládání souborů na úrovni tabulek. Volba formátu pro tabulku ovlivňuje způsob a efektivitu čtení dat v ní uložených. [15]



Obrázek 3.2: Přehled architektury Hive při vyhodnocení dotazu. Převzato z [15].

Přehled komponent, které jsou zapojené do vyhodnocování dotazu v Hive je na obrázku 3.2.

Hive překládá klientské dotazy v jazyce Hive Query Language na prováděcí plán. Prováděcí plán je dále zkompilován do kódu pro příslušný výpočetní nástroj a proveden na vybraných výpočetních uzlech distribuovaného systému Hadoop. [15]

Důležitou komponentou Hive je Metastore, což je databáze udržující informace o datech uložených a spravovaných v Hive. Udržují se zde informace o formátech ukládaných souborů, lokacích uložených souborů v rámci HDFS a jejich mapování na jednotlivé tabulky a další informace a statistiky o uložených datech a příslušných datových objektech. [15]

Hive je potenciálně vhodný kandidát pro řešení práce. Výhodou je podpora dotazovacího jazyka, který je z velké části shodný s SQL. Tento dotazovací jazyk nemá žádná omezení na operace spojení nebo na vyhodnocení podmínek pro filtrování dat. Díky tomu je možné přenést poskytnuté datové schéma do distribuovaného prostředí Hadoop téměř beze změny. Poskytnutá data je možné uložit do různých datových formátů, které mohou být vybrány na základě požadavků na dotazy (zejména sloupcově orientované formáty).

Rozhodl jsem se ale upřednostnit nástroj Apache Impala, který sdílí všechny výše zmíněné výhody. V sekci 3.4 popisující tento nástroj i porovnání těchto dvou kandidátů a odůvodnění pro výběr Apache Impala.

3.3 Apache HBase

Apache HBase (dále jen HBase) je open-source databázový systém, který využívá Hadoop (popsaný v sekci 3.2.1) a je vyvinutý k práci s velkými objemy dat. Stejně jako Hadoop je navržen pro použití v distribuovaném prostředí složeném z nespecializovaných a dostupných výpočetních strojů a umožňuje horizontální škálování. Datový model HBase není relační a je sloupcově orientovaný. Datový a výpočetní model HBase je navržený tak, aby podporoval rychlý náhodný přístup k datům. [20]

Základním datovým objektem v HBase je tabulka. Tabulka sdružuje řádky, které obsahují klíč řádku a může obsahovat další sloupce. Klíč řádku je jednoznačný identifikátor daného řádku a jsou podle něj seřazeny řádky tabulky. Výběr klíče řádky je při tvorbě datového modelu velice důležitý, protože se od něj přímo odvíjí efektivita vyhodnocení dotazů. Sloupce patří do rodin sloupců a jsou tak identifikované kombinací názvu rodiny a svým názvem. Rodiny sloupců shlukují do skupin sloupce a zajišťují, že jsou hodnoty těchto sloupců fyzicky uloženy blízko k sobě na disku. Rodina sloupců také slouží k nastavení vlastností pro sloupce, které zahrnuje, jako jsou způsob komprese nebo formát uložení. [21]

Řádky, které patří do jedné tabulky, obsahují stejné rodiny sloupců, ale nemusí obsahovat všechny sloupce z rodiny. Každý řádek jedné tabulky tak může obsahovat různé atributy. To vede na flexibilní datový model. HBase uchovává informace o rodinách sloupců, ale neuchovává informace o tom, jaké sloupce se v tabulce nacházejí. Místo v tabulce identifikované klíčem řádku, označením rodiny sloupců a názvem sloupce se nazývá buňka. Buňka obsahuje uloženou hodnotu, která je verzovaná. Jednotlivé verze jsou odlišené typicky časovým záznamem o zápisu hodnoty. [21]

HBase má své vlastní rozhraní pro manipulaci s daty. Základní operace pro čtení dat jsou Get a Scan. Get slouží pro čtení atributů jednoho specifikovaného řádku. Scan slouží pro čtení více řádků a může být doplněný filtrem na hodnoty klíče řádku. Základní rozhraní pro dotazování nepodporuje spojování dat z více tabulek. K dosažení spojení dat z více tabulek se musí data denormalizovat nebo spojovat na straně klienta v aplikaci, která s daty pracuje. [20]

Lucas C. Scabora a kolektiv ve své práci [22] zkoumají a porovnávají tři různé přístupy pro modelování dat ve schématu tvaru hvězda pomocí HBase. Prvním přístupem je uložení dat ze všech tabulek dimenzí spolu s daty z faktové tabulky do jedné rodiny sloupců. Dalším přístupem je pro každou původní tabulku vytvořit vlastní rodinu sloupců. Třetím přístupem, navrženým autory práce, je spojit nejčastěji přistupovanou dimenzionální tabulku s faktovou tabulkou do jedné rodiny sloupců, pro ostatní tabulky dimenzí pak vytvořit vlastní rodiny sloupců. Z jimi provedených měření nejlépe vychází přístup s uložením všech dat do jedné rodiny sloupců při dotazech s velkým počtem dimenzí. Při dotazech na menší počet dimenzí si lépe vedly druhé dva přístupy. V dotazech, kde se vyskytovala dimenze připojená k faktové tabulce si nejlépe vedl dle očekávání tento přístup. [22]

Z probraných vlastností HBase a možných datových modelů ve výše zmíněné práci usuzuji, že databáze HBase není samostatně vhodná pro uložení celého schématu (faktové tabulky a tabulek dimenzí) pro účely dotazování. S přihlédnutím na analýzu poskytnutých typických dotazů (v sekci 2.4) nevidím způsob, jak efektivně navrhnout datový model v HBase. S ohledem na volnost, kterou mají uživatelé při dotazování jsem nenašel vhodný návrh pro schéma a klíč řádky, což je klíčová část návrhu datového modelu pro tuto databázi a také nejsilnější nástroj pro efektivní vyhodnocování dotazů.

U datových modelů probíraných ve výše zmíněné práci by nebylo možné při dotazech plně využít klíče řádky a při vyhodnocování dotazů by to vedlo k neefektivnímu skenování celých tabulek.

Další možností by bylo vytvořit denormalizované tabulky obsahující různé kombinace faktové tabulky spojené s daty z tabulek dimenzí. Vzhledem k poskytnutému schématu ale není reálné pokrýt všechny tyto kombinace a vytvořit jen část by vedlo k omezení možností při dotazování, což také není žádoucí.

Rozhodl jsem se ale využít možnosti propojení HBase s Hive nebo Apache Impala. V takovém řešení může být použita HBase pro uložení tabulek dimenzí a Apache Impala pro uložené faktové tabulky. Více o tomto řešení je v následující sekci 3.4. Toto řešení umožňuje uložit dimenzionální tabulky v HBase a zvolit jejich identifikátor jako klíč řádky. Při výběru dat dimenzí na základě cizího klíče, který se ve faktové tabulce vyskytuje, lze efektivně data z tabulek dimenzí uložených v HBase číst za pomoci klíče řádky, který odpovídá v tomto případě cizímu klíči.

3.4 Apache Impala

Apache Impala (dále jen Impala) je open-source projekt, používaný pro vyhodnocování dotazů nad daty uloženými v různých datových formátech v rámci platformy Hadoop. Tento nástroj na vyhodnocování dotazů je navržen pro vysokou výkonnost a vyhodnocování dotazů s co nejmenší dobou odezvy. Proto je podle autorů vhodný pro vyhodnocování interaktivních analytických dotazů. [23]

Impala využívá Hive metastore. To umožňuje zaměnitelné využití obou dvou nástrojů pro práci se stejnými datovými objekty. Propojení s Hive je také podpořené tím, že Impala podporuje vyhodnocování dotazů v jazyce Hive Query Language. [23]

Impala umožňuje vyhodnocování dotazů přímo nad datovými soubory uloženými v HDFS v různých formátech. Pomocí Impala je možné pokládat dotazy do databází v HBase pomocí Hive Query Language. V dokumentaci je také naznačeno, že Impala je navržena pro efektivní vyhodnocování analytických dotazů s dobou odezvy v řádu vteřin až několika minut, kdežto Hive pro rozsáhlejší datové transformace trvající řádově minuty až hodiny. [23]

3.4.1 Parquet

Parquet je sloupcově orientovaný datový formát, navržený tak, aby se nad soubory uloženými v tomto formátu daly efektivně vyhodnocovat rozsáhlé analytické dotazy. [15]

V datovém souboru ve formátu Parquet jsou data rozdělena do skupin řádků. Skupina řádků obsahuje data z množiny řádků. V rámci skupiny řádků jsou data jednotlivých sloupců pro příslušné řádky. Data z jednoho sloupce jsou zapsána na disk za sebe a jsou rozdělena do stránek. Stránky mají hlavičky, které obsahují informace o datech, s pomocí nichž je možné například přeskočit nepodstatné stránky při vyhodnocování dotazu. [15]

Tyto vlastnosti datového formátu Parquet podporují vyhodnocování analytických dotazů. Z povahy sloupcově orientovaného uložení dat plyne, že lze při vyhodnocování dotazu z disku číst pouze data jednotlivých sloupců, které jsou v dotazu požadované. Lze efektivně provádět výpočty agregačních funkcí nad daty z celých sloupců. Uložení dat po sloupcích také podporuje efektivnější kompresi dat. [24]

Nevýhodou datového formátu Parquet je velký nárok na výpočetní prostředky při zapisování dat. [15]

3.4.2 Architektura Impala Server

Impala Server slouží pro distribuované a masivně paralelní zpracování dat. Server Impala je tvořen více procesy, které jsou spuštěné na propojených výpočetních uzlech, celkem se skládá ze tří typů procesů. [25]

Impala Daemon je základní typ procesu, který se stará o výpočty. Mezi jeho úkoly patří čtení a zápis souborů, odbavování dotazů zadaných z různých klientských rozhraní, rozesílání dílčích úkolů na další výpočetní stroje po paralelizování dotazů a přenos dílčích výsledků dotazů zpět na uzel, který je zodpovědný za vyhodnocení daného dotazu. [25]

Impala Statestore je proces, který udržuje informace o stavu uzlů s procesy Impala Daemon. Tyto informace rozesílá všem procesům Impala Daemon. [25]

Impala Catalog Service proces udržuje a rozesílá metadata a změny v nich, které vznikly zadanými příkazy. Stará se o to, aby všechny procesy Impala Daemon měly aktuální informace o spravovaných databázových objektech. [25]

3.4.3 Posouzení vhodnosti Impala

Impala splňuje všechny stanovené požadavky na řešení (popsané v sekci 2.3). Podle výše sepsaných vlastností se hodí pro analytické dotazy, které dokáže vyhodnotit s krátkou odezvou. To je přesně potřeba u databáze, která má poskytovat data BI aplikaci prezentující data. Impala dále přijímá příkazy v jazyce Hive Query Language, který se z velké části shoduje se standardním SQL, takže není potřeba nijak měnit datový model a překládat dotazy.

Impala jsem dal přednost oproti Hive z důvodu výše zmíněné vhodnosti pro vyhodnocování analytických dotazů s krátkou dobou odezvy. Oproti tomu byl Hive původně navržen pro déle trvající datové transformace a výpočty. [23] V ostatních vlastnostech, jako je jazyk HQL a využití různých datových formátů, se tyto dvě technologie shodují, a proto nevidím žádnou výhodu v použití nástroje Hive místo Impala.

Podle spisu osvědčených postupů od Cloudera [26] je Impala a Hadoop vhodné použít na dotazování nad daty ve schématu tvaru hvězdy. To přesně odpovídá vzorovému schématu dat poskytnutých pro porovnání výkonnosti.

Jako zajímavé řešení vidím využití Impala s daty uloženými v HDFS ve formátu Parquet. Tento formát by měl podle výše popsaných vlastností podpořit efektivní vyhodnocení poskytnutých dotazů, které jsou analytické povahy.

Dalším řešením, které považuji za vhodné, je využití Impala s daty dimenzionálních tabulek uložených v databázi HBase a faktovou tabulkou v Impala ve formátu Parquet. V dokumentaci od Cloudera k využití Impala s HBase [27] je podobné řešení doporučené v případě analytických dotazů obsahujících spojení velkých faktových tabulek s malými tabulkami dimenzí a výpočty agregačních funkcí nad daty faktové tabulky, což odpovídá poskytnutým vzorovým datům.

3.5 MongoDB

Jako zástupce dokumentově orientovaných databázových systémů jsem vybral MongoDB. Databáze MongoDB byla navržena tak, aby dosáhla dobré škálovatelnosti a dalo se v ní snadno vyvíjet. Má vlastní dotazovací jazyk specificky pro dokumentový model. Komunitní verze MongoDB má volně dostupný zdrojový kód a je možné ji užívat zdarma. [28]

Základem datového modelu MongoDB je dokument, který obsahuje jeden konkrétní datový záznam. Tyto dokumenty jsou ukládané ve formátu BSON, což je zkratka pro binární JSON. Dokumenty jsou seskupeny do kolekcí. MongoDB nevyžaduje u těchto dokumentů pevně dané schéma a tak může v jedné kolekci být několik dokumentů se zcela odlišným schématem. Dokumenty obsahují data uložená v podobě párů klíče a hodnoty. Klíč je textový řetězec a hodnota může být například číslo, textový řetězec, ale například i pole hodnot nebo objekt, což je vnořený dokument. Každý dokument je jednoznačně určený identifikátorem objektu, který vzniká automaticky při vytvoření dokumentu, je ho ale možné definovat manuálně. [29]

Sdílení dat mezi dvěma dokumenty lze docílit buď zapouzdřením dokumentu nebo odkazem. Zapouzdřený dokument je součástí jiného dokumentu jako hodnota nějakého z atributů. Odkaz pak ukazuje na samostatný dokument. [29]

Max Chavalier a kolektiv ve své práci [30] navrhují několik možností, jak modelovat data v dokumentově orientované databázi MongoDB. Předkládají způsoby, jak převést dimenzionální model se schématem tvaru hvězdy do dokumentově orientovaného modelu. Data, se kterými pracují, jsou tvořena jednou faktovou tabulkou a několika tabulkami dimenzí. To odpovídá schématu dat poskytnutých pro experimenty v této práci. [30]

Prvním modelem, který popisují, je plochý model (označený jako DFL), který je založený na denormalizaci původního schématu. V tomto modelu každý dokument odpovídá jednomu záznamu z faktové tabulky a navíc obsahuje všechna příslušná data z atributů dimenzí. Druhý model (označený jako DSH) je založený na odkazech na data dimenzí ve faktové tabulce. Každá tabulka z původního modelu je uložena ve své kolekci. Faktová tabulka obsahuje identifikátory dokumentů, které obsahují příslušná dimenzní data. [30]

Další 2 způsoby převodu dimenzionálního modelu na dokumentový model spočívají v rozšíření OLAP kvádrů (Online Analytical Processing, z angličtiny online analytické zpracování). OLAP kvádr je pohled na data, který obsahuje předpočítané agregační funkce nad daty z faktové tabulky, která jsou shlukovaná podle vybraných atributů dimenzí. Autoři práce navrhují zajímavá rozšíření OLAP kvádru o zapouzdření nebo o detailní pohled na napočítanou hodnotu. Zapouzdření spočívá v zapouzdření dat z nižších úrovní hierarchie atributu. Pohled tedy obsahuje kromě agregovaných hodnot pro vyšší úroveň dimenze také agregované hodnoty pro nižší dimenze (například agregované hodnoty pro roky a zapouzdřené hodnoty pro jednotlivé měsíce). Obsažení detailu agregované hodnoty spočívá v zahrnutí dat jednotlivých dílčích složek, které agregaci tvoří (například jednotlivé položky součtu). [30]

Všechny čtyři způsoby dokumentově orientovaného modelování dat pak autoři práce porovnávají s původním schématem v relační databázi pomocí měření času odezvy prováděných analytických dotazů. Dotazy nad plochým modelem DFL mají kratší odezvu než dotazy nad modelem DSH při použití MongoDB databáze se třemi uzly. To autoři práce vysvětlují špatnou efektivitou operace spojování v MongoDB. V porovnání s relační databází jsou odezvy na dotazy při použití základních modelů v MongoDB zhruba desetkrát delší. Odezva na dotazy při řešení s využitím rozšířených OLAP kvádrů jsou srovnatelná s obdobným řešením pomocí relační databáze. [30]

Dva základní způsoby převodu dimenzionálního modelu do dokumentové databáze, uvedené v práci zmíněné výše, by bylo možné použít i pro řešení tohoto zadání. Z výsledků experimentů ve zmíněné práci lze ale odvodit, že použití takových řešení by nevedlo ke snížení odezvy dotazů oproti stávajícímu řešení v relační databázi Oracle. [30]

Řešení s pomocí rozšířených OLAP kvádrů je z pohledu efektivity vyhodnocování dotazů vhodné. Z experimentů ve zmíněné práci jsou tato řešení z pohledu odezvy na stejné úrovni jako řešení v relační databázi. Toto řešení ale z důvodu velkého počtu dimenzí a jejich atributů a volnosti při tvorbě dotazů není prakticky proveditelné. I kdyby toto řešení bylo možné, tak se stále nevyplatí migrovat databázi na jinou technologii, protože implementace OLAP kvádrů v relační databázi si podle experimentů ve studované práci vedla obdobně jako řešení v MongoDB. [30]

3.6 Shrnutí vybraných řešení

První řešení, které jsem na základě informací v této kapitole zvolil, je Impala s daty uloženými v distribuovaném systému souborů HDFS ve formátu Parquet. Jak je popsáno výše, Impala je navržena pro vyhodnocování analytických dotazů s krátkou odezvou, sloupcový formát Parquet pak podporuje vyhodnocování analytických dotazů.

Jako druhé řešení jsem zvolil opět Impala jako rozhraní pro zadávání a vyhodnocování dotazů a pro uložení dat faktové tabulky. Data faktové tabulky jsem se rozhodl uložit do HDFS ve formátu Parquet. Rozdíl je u tabulek dimenzí, které jsem se rozhodl uložit pomocí HBase. Uložení tabulek dimenzí v HBase by mohlo podpořit vyhledávání záznamů z dimenzí podle jejich primárního klíče při operaci spojování faktové tabulky s tabulkami dimenzí.

Technologie použité pro obě dvě řešení jsou součástí CDH (Cloudera Distribution Hadoop). CDH je komerční distribuce Hadoop a dalších technologií, které s Hadoop spolupracují. Distribuce CDH je spravovaná a poskytovaná společností Cloudera. [31]

Mimo samotné projekty pro práci s daty Cloudera poskytuje také webové rozhraní Cloudera Manager pro správu a monitorování všech běžících komponent CDH. [32]

Další webové rozhraní, které Cloudera poskytuje, je Hue, které slouží jako rozhraní pro zadávání databázových dotazů například do Hive nebo Impala a zobrazení výsledků těchto dotazů. [33]

Kapitola 4

Realizace vybraných řešení

4.1 Instalace CDH

CDH (Cloudera Distribution Hadoop) je distribuce Hadoop, která obsahuje komponenty jako HDFS, Spark, Impala a HBase. [31] Pro účely této práce jsem se rozhodl nainstalovat neplacenou zkušební verzi produktu CDH. Ta v sobě zahrnuje všechny potřebné dříve zmíněné nástroje a aplikace zvolené pro obě z řešení pomocí NoSQL technologií.

Prvním způsobem instalace CDH, který jsem zvažoval, bylo nainstalovat zkušební verzi přímo na operační systém serveru poskytnutého VZP. Protože se ale tento server skládá jen z jednoho stroje, znamenalo by to, že cluster komponent CDH by se skládal jen z jednoho uzlu. Nebylo by tak možné využít naplno vlastností těchto nástrojů, které jsou založené na distribuovaných výpočtech.

Proto jsem se rozhodl pro instalaci a zprovoznění CDH clusteru s využitím kontejnerizace pomocí technologie Docker. Kontejnerizace je způsob virtualizace procesů operačních systémů. Kontejner je izolované prostředí, ve kterém je možné spouštět aplikace. Kontejnery běžící na jednom stroji sdílí operační systém hostitelského počítače. [34] Docker je nástroj, s jehož pomocí lze kontejnery vytvářet, spouštět a spravovat. Definice a popis kontejnerů se označuje jako obraz. [35]

Protože je na poskytnutém stroji výkonný hardware zajišťující dostatek výpočetních prostředků, mohl jsem spustit více kontejnerů. Každý z těchto kontejnerů představuje jeden výpočetní uzel clusteru s komponentami CDH. Virtuálně jsem tak mohl zprovoznit cluster se čtyřmi výpočetními uzly na jednom fyzickém stroji. Řešení s více kontejnery běžícími na jednom stroji není tak efektivní z hlediska využití výpočetních zdrojů, jako kdyby se cluster skládal z více fyzických strojů. Naopak výhodou více virtuálních uzlů na jednom fyzickém stroji je snížení zpoždění způsobeným síťovou komunikací mezi jednotlivými uzly, které by v případě více fyzických strojů bylo vyšší. Porovnání výkonu takového řešení s aktuálním řešením považuji za více smysluplné než řešení s CDH clusterem s pouze jedním výpočetním uzlem.

Pro mnou zvolený postup bylo nejdříve potřeba nainstalovat samotný Docker. S jeho pomocí je možné zprovoznit, spustit a udržovat v chodu konečný cluster kontejnerizovaných uzlů s komponentami CDH. Instalaci včetně stažení instalačního balíčku programu Docker a dalšího software, na kterém Docker závisí, jsem nemohl nechat pouze na správci balíčků operačního systému. Důvodem je, že na poskytnutém serveru je blokovaný přístup k internetu mimo sítě VZP.

Nabízela se možnost vlastnoručně stáhnout instalační balíčky z veřejných repozitářů na jiném počítači a na cílový server je následně nějakým způsobem přenést. Při ručním stahování bych ale musel postahovat velké množství těchto instalačních balíčků kvůli závislostem na další software. Při tomto postupu je velké riziko chyby například v podobě zvolení špatné verze některého z programů.

Proto jsem se rozhodl zprovoznit si na jiném stroji virtuální počítač se stejným operačním systémem jako je na serveru. Tam jsem potřebné instalační balíčky nainstaloval s pomocí správce balíčků a přenesl je na cílový počítač. Protože jsem virtuální počítač zprovoznil pomocí Oracle VM VirtualBox [36] na jiném stroji, kde nebyl omezený přístup k internetu, bylo možné stáhnout balíčky automatizovaně pomocí správce balíčků operačního systému. Pro zjištění typu a verze operačního systému cílového stroje jsem použil následující příkaz v jeho příkazové řádce.

```
$ cat /etc/centos-release
```

Po zjištění specifikací operačního systému jsem zprovoznil virtuální počítač s odpovídajícím operačním systémem.

Na tomto virtuálním počítači jsem pak v příkazové řádce pomocí následujícího příkazu stáhnul instalační balíček Dockeru i se všemi ostatními balíčky, na kterých je Docker závislý.

```
yum install --downloadonly --downloaddir=./docker-packages docker
```

Po komprimování a přenesení na zařízení v síti VZP jsem z tohoto zařízení přenesl instalační balíčky na cílový server pomocí programu WinSCP, který slouží pro zabezpečený přenos souborů po síti. [37]

Na cílovém serveru jsem po dekomprimování instalačních balíčků Docker pomocí následujícího příkazu nainstaloval. Použité možnosti u příkazu zakazují využití online repozitářů při stahování balíčků a radí správci balíčků, aby programy instaloval z balíčků dostupných lokálně na disku stroje.

```
yum --disablerepo=* localinstall ./docker-packages/*.rpm
```

Při spuštění tohoto příkazu došlo k chybám způsobeným existencí některých balíčků, které se správce balíčků namísto instalace pokusil aktualizovat za pomoci stažení nové verze z repozitáře. Tyto balíčky jsem tedy z instalace vyloučil pomocí volby `--exclude <jméno-balíčku>`.

Po instalaci jsem službu Docker na cílovém serveru spustil pomocí příkazu níže.

```
sudo systemctl start docker
```

Po nainstalování a spuštění programu Docker jsem narazil na nedostatek místa na diskové jednotce, na které byl Docker nainstalován a kam také ukládal data z běhu programu. Proto bylo nutné Docker přesunout na jinou diskovou jednktu následujícím příkazem.

```
sudo cp -axT /var/lib/docker /inst/docker
```

Byla nutná také úprava konfiguračního souboru pro službu Docker pro upřesnění cesty k datovému adresáři pro Docker. Do konfiguračního souboru `/etc/docker/daemon.json` jsem přidal následující možnost.

```
{
"data-root": "/inst/docker"
}
```

Po úspěšné instalaci a spuštění služby Docker bylo potřeba připravit a spustit Docker obraz pro zkušební CDH cluster. Blokovaný přístup k vnější internetové síti zde opět představoval limitaci pro stažení a spuštění tohoto Docker obrazu. Opět jsem tedy využil zprovozněný virtuální počítač, na kterém jsem taktéž nainstaloval službu Docker. Vývojáři Cloudera připravili kód, spouštěcí skripty a Docker obraz, s jejichž pomocí lze ve službě Docker zprovoznit vlastní zkušební kontejnerizovaný cluster s komponentami CDH. Docker obraz i se základními instrukcemi pro jeho spuštění je dostupný na webovém repozitáři obrazů pro Docker. [38] V příkazové řádce jsem s pomocí příkazu docker pull stáhnul z repozitáře tento Docker obraz, který slouží pro zprovoznění a spuštění zkušebního CDH clusteru.

```
sudo docker pull cloudera/clusterdock
```

Docker umožňuje obraz exportovat jako soubor na lokální systém souborů. Toho jsem využil, abych vybraný Docker obraz mohl přenést na cílový server. S pomocí následujícího příkazu jsem tedy stažený Docker obraz uložil do souboru, který může být přenesen na cílový server a tam načten službou Docker.

```
sudo docker save --output ./cloudera-clusterdock.tar
```

Získaný soubor jsem zkomprimoval pro jednodušší přenos na cílové zařízení. Komprimovaný uložený soubor obsahující Docker obraz jsem přenesl na počítač v síti VZP a z něj pomocí WinSCP na cílový server, stejně jako v případě instalačních balíčků služby Docker.

V příkazové řádce na cílovém serveru jsem dekomprimovaný obraz načetl do běžící služby Docker. Takto načtený obraz z lokálně uloženého souboru následně může být využit pro spuštění kontejneru s daným obrazem, aniž by se musel stahovat z online repozitáře. Načtení Docker obrazu jsem provedl následujícím příkazem.

```
sudo docker load --input ./cloudera-clusterdock.tar
```

Stejně jsem postupoval také v případě stahování, přenesení a načtení Docker obrazů `cloudera/clusterdock:581_581_primary-node` a `cloudera/clusterdock:581_581_secondary-node`. Tyto Docker obrazy definují spuštění kontejnerů, které představují samotné výpočetní uzly s nainstalovanými a nastavenými komponentami CDH.

Na cílový server bylo také potřeba stáhnout a přenést pomocný skript, který zajistí instalaci a spuštění kontejnerizovaného clusteru v Dockeru. V tomto skriptu je definovaná funkce, která zařídí spuštění Docker kontejneru s obrazem `cloudera/clusterdock`. [38]

V tomto kontejneru jsou připravené skripty a spustitelné soubory, které zajistí instalaci a spuštění požadovaných strojů, které budou tvořit kontejnerizovaný cluster. Důležitou součástí tohoto skriptu je vytvoření síťového propojení mezi jednotlivými kontejnery, které představují uzly výpočetního clusteru. Ve spuštěných kontejnerech se pak spustí instalační proces CDH, uzly se propojí do clusteru a nastaví se specifikované role v rámci clusteru. Podle daných rolí se na jednotlivé kontejnery nainstalují a spustí příslušné komponenty a nástroje CDH. [38]

Následující příkaz umožní vyvolat funkce definované v pomocném skriptu.

```
source ./clusterdock.sh
```

Po prozkoumání a pochopení pomocného skriptu jsem zjistil, že bude potřeba nastavit určité proměnné prostředí. Díky těm bude zajištěné použití správných stažených Docker obrazů a také to, že se nic nebude stahovat z vnější internetové sítě. Nastavil jsem následující proměnné prostředí.

```
export CLUSTERDOCK_IMAGE="cloudera/clusterdock:latest"
export CLUSTERDOCK_PULL="false"
```

Protože jeden ze skriptů počítal s jiným pojmenováním Docker obrazů, než byly obrazy pojmenované na cílovém serveru, musel jsem pomocí následujících příkazů vytvořit alias (tag) pro dané Docker obrazy. Tento tag umožní vytvořit jiné pojmenování Docker obrazu, které se odkazuje na původní obraz, neduplikují se tak žádná data na disku a je možné použít různé názvy pro vytvoření kontejneru podle daného obrazu. [35]

```
sudo docker tag <existujici obraz> <nove jmeno>
```

Dále už bylo možné spustit samotnou funkci, která zajistila instalaci a zprovoznění kontejnerizovaného clusteru. Rozhodl jsem se spustit cluster s jedním kontejnerem představujícím primární uzel a se třemi kontejnery představující sekundární uzly CDH clusteru. Podle návodu ke zprovoznění testovacího clusteru CDH [38] jsem spustil následující příkaz.

```
clusterdock_run ./bin/start_cluster -n fit cdh \
    --primary-node=node-1 \
    --secondary-nodes='node-{2..4}'
```

Po úspěšném dokončení instalace vypsal instalační skript čísla portů, na kterých se lze připojit k nástrojům Cloudera Manager a Hue. Tyto nástroje jsou součástí CDH a pomocí nich lze administrovat cluster a jeho komponenty. [32] Hue nabízí také webové rozhraní pro editování a provádění příkazů pro různé komponenty, pro tuto práci jsou důležitá zejména dotazovací rozhraní pro Impala a Hive. V Hue je také možné spravovat uživatelské účty a jejich přístupy k jednotlivým komponentám a objektům v nich. [33] Já jsem pro tuto práci využil původní administrátorský účet.

4.2 Nahrání dat do Impala

Při vytvoření databáze v nástroji Impala a její naplnění poskytnutými daty jsem se na základě předchozích zkušeností a znalosti daných nástrojů rozhodl postupovat následujícími kroky:

1. přenesení poskytnutých dat na distribuovaný systém souborů HDFS
2. transformace dat a jejich zápis ve formátu Parquet za pomoci nástroje PySpark
3. vytvoření tabulek v Impala pomocí příkazů dotazovacího jazyka Hive Query Language

Všechny služby a technologie využité v této části jsou součástí CDH a byly nainstalované a spuštěné v předchozím kroku.

4.2.1 Přenos dat do HDFS

Prvním krokem při transformaci a načítání dat do tabulek v Impala bylo přenesení dat na cílový server poskytnutý pro účely této práce, na kterém byl nainstalován kontejnerizovaný CDH cluster. K přenosu komprimovaných dat jsem použil nástroj WinSCP.

Z cílového serveru jsem data nahrál na jeden z běžících Docker kontejnerů, ve kterém byly spuštěné služby CDH a tvořil část testovacího clusteru.

Níže popsané kroky jsem provedl pomocí interaktivní konzole připojené na cílový server.

Nejdříve jsem zjistil identifikátor kontejneru následujícím příkazem, který vypisuje informace o běžících Docker kontejnerech.

```
sudo docker ps
```

Dále jsem přenesl poskytnutá data do systému souborů operačního systému běžícího v Docker kontejneru. Po zjištění identifikátoru kontejneru jsem přenos provedl následujícím příkazem.

```
sudo docker cp ./DATA.zip <identifikátor kontejneru>:/data
```

Po provedení předchozího příkazu byla data nahrána do Docker kontejneru, který představuje uzel CDH clusteru.

Po připojení k interaktivní konzoli operačního systému běžícího v tomto kontejneru jsem mohl data načíst do distribuovaného systému souborů HDFS. Příkazem níže jsem spustil zmíněnou interaktivní konzoli.

```
sudo docker exec -it <identifikátor kontejneru> /bin/bash
```

Po provedení příkazu výše jsem mohl zadávat příkazy přímo v operačním systému běžícím v Docker kontejneru. V tomto systému jsem data nahrál do HDFS, kde jsou již přístupné pro další nástroje CDH. Abych se vyhnul problémům s přístupovými právy k souborům jak v lokálním systému souborů, tak v systému souborů HDFS prováděl jsem příkazy pod uživatelským účtem *hdfs*. Tento uživatelský účet je vytvořený automaticky při instalačním procesu CDH.

Příkazy níže jsem nejdříve vytvořil v HDFS cílový adresář a pak do něj nahrál poskytnutá data.

```
sudo -u hdfs hdfs dfs -mkdir /tmp/data
sudo -u hdfs hdfs dfs -put /<adresář s daty>/*.csv /tmp/data
```

4.2.2 Transformace dat pomocí PySpark

K datům nahraným v systému HDFS mají přístup i další komponenty a nástroje z ekosystému CDH. Po nahrání dat jsem použil nástroj PySpark [39] k transformaci a zápisu dat. Data jsem převedl do takové podoby, aby se s nimi dalo dobře pracovat v nástroji Impala.

PySpark je rozhraní pro Apache Spark v programovacím jazyce Python. Apache Spark obsahuje různé moduly pro provádění analýz, transformací a výpočtů nad velkým množstvím dat, zejména v distribuovaném výpočetním prostředí. Pro tuto část práce jsem využil zejména moduly Spark SQL a Spark DataFrame. [39]

Protože nainstalovaný testovací cluster obsahuje Apache Spark ve verzi 1.6., která ve svém základu nenabízí rozhraní pro práci se soubory ve formátu CSV (angl. Comma Separated Values, přeloženo jako čárkou oddělené hodnoty), musel jsem využít rozšiřujících balíčků, které toto rozhraní poskytují. Jedná se o balíčky spark-csv [40] verze 2.11 a commons-csv [41] ve verzi 1.2. Tyto balíčky jsem stáhnul a přenesl na cílový server stejným způsobem jako instalační balíčky v průběhu instalace CDH. Z cílového stroje jsem balíčky přenesl do jednoho z běžících kontejnerů tvořícího testovací CDH cluster.

```
sudo docker cp /<cesta k balíčkům> \
    <identifikátor kontejneru>:/<cílová cesta>
```

Transformaci dat jsem se rozhodl provést pomocí rohrazní PySpark, ve kterém je možné interaktivně provádět příkazy v programovacím jazyce Python a jsou v něm dostupné funkce modulů Apache Spark. Aby v interaktivním prostředí bylo možné využít funkcionalitu stažených rozšiřujících balíčků, musí být prostředí spuštěno následovně.

```
pyspark --jars <cesta k prvnímu balíčku> \
    ,<cesta ke druhému balíčku>
```

Po spuštění interaktivní konzole PySpark jsem načetl data z HDFS ve formátu CSV do datové struktury DataFrame. Struktura DataFrame je definovaná v Apache Spark a nabízí rozhraní pro práci se strukturovanými daty. [39] Příkaz pro načtení dat do DataFrame jménem *df* je následující.

```
df = sqlContext.read \
    .format('com.databricks.spark.csv')\
    .options(header='true', delimiter=';')\
    .load('/tmp/data/RADDOK.csv')
```

Specifikoval jsem, že data jsou ve formátu CSV a že soubor obsahuje popisující hlavičku a jednotlivá pole na řádce jsou oddělena pomocí středníku.

Dále jsem si nechal vypsat schéma načtených dat tak, jak jej implicitně identifikoval program PySpark.

```
df.printSchema()
```

Předchozí příkaz na výstupu konzole vypsal schéma načtených dat. Všechna pole PySpark načetl jako textový řetězec a proto je nutná transformace některých z nich. PySpark umožňuje načíst data s předem definovaným schématem a není pak potřeba následná transformace datových typů jednotlivých sloupců. Jelikož ale každá datová entita obsahuje velké množství atributů a jen malé množství z nich není textový řetězec, bylo pro mě výhodnější jednotlivé sloupce transformovat než předem definovat celé schéma.

Pro následující práci s daty jsou potřeba pomocné knihovní funkce a objekty. Ty jsem zpřístupnil takto.

```
from pyspark.sql.types import *
from pyspark.sql.functions import col
```

U sloupců, které je potřeba převést na jiný datový typ, jsem tuto transformaci provedl následovně.

```
df = df.withColumn('CAS_KLIC', col('CAS_KLIC')\
    .cast(DecimalType(38,0)))
```

Jako příklad jsem uvedl převod sloupce `CAS_KLIC` na číselný datový typ. Pro ostatní sloupce, u kterých je potřeba převod datového typu, jsem tak provedl analogicky. Výjimkou byly sloupce s hodnotami obsahujícími reálná čísla. Jelikož v poskytnutých datech byla desetinná čárka, se kterou PySpark nepočítá (operuje s desetinnou tečkou), bylo potřeba provést následující transformaci.

```
colname = 'RADDOK_POCET'
df = df\
    .withColumn(colname,regexp_replace(col(colname), ',', '\.')\
    .cast(DecimalType(38, 4)))
```

V uvedeném příkazu jsem v hodnotách sloupce `RADDOK_POCET` přepsal čárku na tečku a následně převedl hodnoty sloupce na číselný datový typ. Další sloupce obsahující reálná čísla jsem transformoval analogicky.

Po provedení převodu na požadované datové typy u všech sloupců už jsem mohl transformovaná data zapsat zpět na HDFS. Transformovaná data jsem zapsal ve formátu Parquet, který je vhodný pro zpracování nástrojem Impala.

```
df.write.parquet('/user/hdfs/data/cas')
```

Výše jsem uvedl jako příklad zápis datového souboru pro tabulku dimenze `D_CAS`. Data pro další tabulky dimenzí jsem transformoval a zapsal analogicky, pouze s jinými názvy tabulek, jejich sloupců a jinými cestami, kde se soubory nachází.

U faktové tabulky byl po transformaci dat a schématu jediný rozdíl při zápisu dat. Faktová tabulka je rozdělena do více partition. Pro zapsání dat podle partition jsem použil při zápisu klauzuli `paritionBy`.

```
df.write.partitionBy('cas_klic')\
    .parquet('/user/hdfs/data/raddok')
```

4.2.3 Vytvoření tabulek v Impala

Po nahrání všech dat do HDFS a jejich transformaci na správné datové typy a do formátu Parquet jsem mohl přejít k vytváření tabulek v Impala. Impala efektivně a uživatelsky přívětivě pracuje s daty ve formátu Parquet. Využil jsem toho, že v datových souborech Parquet jsou uložené i informace o schématu dat, tedy názvy a datové typy jednotlivých atributů. [24] S těmito informacemi umí Impala pracovat a umí z nich automaticky odvodit schéma tabulky, do které mají být data uložena. Z výše zmíněných důvodů jsem se rozhodl nechat vytváření vlastních tabulek až jako poslední krok a nechat službu Impala, aby tabulky automatikcy vytvořila na základě dat uložených ve formátu Parquet. Příkazy v jazyce Hive Query Language pro modelování dat a dotazování jsem zadával do interaktivní konzole Impala shell, která se na některém z uzlů kontejnerizovaného clusteru spustí následujícím příkazem.

```
impala-shell
```

Níže je uvedena ukázka příkazu v jazyce Hive Query Language, po jehož zadání do interaktivní konzole Impala byla vytvořena tabulka časové dimenze `D_CAS` v databázi `VZP_DW_L2_VIEW_SCHEMA`, kterou jsem vytvořil před spuštěním příkazu vytvářejícího samotnou tabulku.

```
create database VZP_DW_L2_VIEW_SCHEMA;

create external table VZP_DW_L2_VIEW_SCHEMA.D_CAS
like parquet '/<cesta k souboru parquet na HDFS>'
stored as parquet
location '/user/hdfs/data/cas';
```

Nejdříve jsem specifikoval jména tabulky a databáze, v níž se má tabulka nacházet. Dalším příkazem jsem uvedl, že má Impala vyčíst schéma tabulky z již existujícího souboru ve formátu Parquet. Poslední 2 řádky příkazu znamenají, že se data tabulky mají ukládat do souborů typu Parquet a specifikoval jsem lokaci, kam se na distribuovaném systému souborů HDFS data mají ukládat.

Pro zbytek tabulek dimenzí je postup jejich vytvoření analogický, stačí změnit název tabulky a přepsat lokace existujícího Parquet datového souboru a lokace, kam se mají data z tabulky zapsat. Přesný název souboru typu Parquet, který slouží jako vzor pro vytvoření schématu tabulky, se dá zjistit například vypsáním souborů v adresáři HDFS, kam byla data zapsána pomocí PySpark. Pro vypsání těchto souborů jsem následující příkaz spustil v interaktivní konzoli jednoho z kontejnerizovaných uzlů clusteru.

```
sudo -u hdfs hdfs dfs -ls /user/hdfs/data/cas
```

U faktové tabulky, která využívá partitioning, je potřeba tuto skutečnost zohlednit ve vytvářecím skriptu. Následující příkaz zahrnuje klauzuli `parition by`, díky které se použije partitioning podle zvoleného atributu.

```
create external table VZP_DW_L2_VIEW_SCHEMA.FMT_RADEK_DOKLADU_ZP
like parquet '/<cesta k souboru parquet na HDFS>'
partitioned by (cas_klic decimal(38,0))
stored as parquet
location '/user/hdfs/data/raddok';
```

Po vytvoření tabulky, která využívá partitioning, je potřeba přidat jednotlivé partition. To jsem provedl následujícím příkazem.

```
alter table fmt_radek_dokladu_zp
add partition (cas_klic=2459155)
location '/user/hdfs/data/raddok/cas_klic=2459155';
```

Jako příklad jsem uvedl přidání jedné partition. Pro ostatní partition jsem jejich vytvoření provedl analogicky, pouze se změněnou hodnotou atributu, který je pro partitioning použitý.

4.3 Nahrání dat do HBase

Všechny nástroje a komponenty potřebné pro tuto část jsou součástí již nainstalovaného CDH clusteru. Při zaplňování tabulek v HBase v této části řešení jsem využil již provedených kroků v předchozí sekci 4.2. To bylo možné díky tomu, že pomocí Hive Query Language je možné vytvořit tabulky v Impala, jejichž data jsou zapisována do HBase tabulek a při vyhodnocování dotazů jsou z nich opět načtena. Tyto tabulky lze poté snadno zaplnit daty z již existujících tabulek v Impala.

Toto řešení jsem zprovoznil až po zprovoznění předchozího řešení, tedy nahrání dat do tabulek v Impala. V případě využití pouze tohoto řešení by bylo potřeba replikovat krok nahrání dat do HDFS z předchozí sekce 4.2.1. Následoval by krok s nahráním dat do HBase. Kroky s vytvořením tabulek v HBase a následné vytvoření na ně namapovaných tabulek Impala jsou již shodné s tímto řešením.

Práce na tomto řešení se skládala z těchto tří kroků:

1. vytvoření tabulek v HBase
2. vytvoření odpovídajících tabulek v Impala, které jsou mapované na HBase tabulky
3. nahrání dat do HBase tabulek

4.3.1 Vytvoření tabulek v HBase

Tabulky v HBase jsou tvořené z rodin sloupců – atributů. Rodiny sloupců shlukují jednotlivé sloupce, jejichž jména nejsou při vytváření tabulky definované. Při vytváření tabulek v HBase se definuje pouze název tabulky a názvy rodin sloupců. Při vytváření tabulek v HBase, které odpovídají tabulkám dimenzí ve schématu poskytnutých dat, jsem se řídil radami z dokumentace HBase. Zejména udržení malého počtu rodin sloupců a zavedení krátkých názvů rodin sloupců i konkrétních sloupců. [20]

Atributy tabulek dimenzí jsem rozdělil do dvou rodin sloupců. Každá tabulka dimenzí obsahuje část s vlastními daty popisující danou dimenzi a dále část s auditními sloupci. Na příkladu níže ukazuji založení tabulky časové dimenze `d_cas`.

```
create 'd_cas', 'd', 'a'
```

Rodinu sloupců pro část s vlastními popisnými daty dimenze jsem označil `d` a rodinu sloupců s auditními údaji jsem označil `a`. Zbývající tabulky dimenzí jsem vytvořil analogicky výše zmíněným příkazem se změněným názvem tabulky.

4.3.2 Tvorba tabulek v Impala mapovaných na tabulky v HBase

Po vytvoření tabulek v HBase bylo mým dalším krokem vytvoření odpovídajících tabulek v Impala, které jsou namapované na data v HBase tabulkách. Mapování sloupců tabulky v Impala na sloupce tabulek HBase je součástí příkazu pro vytvoření tabulek.

Pro tabulky namapované na HBase tabulky jsem v Impala vytvořil novou databázi.

```
create database hbase;
```

Tabulku pro data časové dimenze `D_CAS` jsem vytvořil s pomocí následujícího příkazu.

```
create external table hbase.d_cas
(
    CAS_KLIC decimal(38,0), CAS_ID string,
    CAS_DEN_KOD string, CAS_TYDEN_KOD string,
    CAS_DEKADA_KOD string, CAS_MESIC_KOD string,
    CAS_KVARTAL_KOD string, CAS_POLOLETI_KOD string,
    CAS_ROK_KOD string, CAS_DEN_TYDNE_KOD string,
    CAS_DEN_TYDNE_PORADI decimal(38,0),
    CAS_DEN_MESICE_PORADI decimal(38,0),
    CAS_DEN_ROKU_PORADI decimal(38,0),
    CAS_PRACOVNI_DEN_PRIZ string,
    CAS_PREDCHOZI_MESIC_KLIC decimal(38,0),
    CAS_PREDCHOZI_ROK_KLIC decimal(38,0),
    CAS_PREDCHOZI_PRACOVNI_DAT string,
    CAS_NASLEDUJICI_PRACOVNI_DAT string,
    CAS_PRAC_DEN_MESICE_PORADI decimal(38,0),
    CAS_PRAC_DNU_V_MESICI_POCET decimal(38,0),
    CAS_KONEC_ROKU_PRIZ string, CAS_KONEC_MESICE_PRIZ string,
    CAS_ZACATEK_KVARTALU_PRIZ string,
    CAS_KONEC_KVARTALU_PRIZ string,
    CAS_NACTENA_DATA_PRIZ string,
    CAS_POSLED_NACTENA_DATA_PRIZ string,
    CAS_MESIC_CZ_NAZEV string, CAS_MESIC_PORADI string,
    CAS_A_PLATNE_OD_DAT string, CAS_A_PLATNE_DO_DAT string,
    CAS_A_PLATNY_PRIZ string, CAS_A_SMAZANY_PRIZ string,
    CAS_A_UCINNE_OD_DAT string, CAS_A_VLOZENO_DC string,
    CAS_A_ZMENENO_DC string, CAS_A_SMAZANO_DC string,
    CAS_A_ZDROJ_SYSTEM string
)
stored by 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
with serdeproperties
('hbase.columns.mapping' =
':key, d:c1, d:c2, d:c3, d:c4, d:c5,
d:c6, d:c7, d:c8, d:c9, d:c10, d:c11, d:c12, d:c13, d:c14,
d:c15, d:c16, d:c17, d:c18, d:c19, d:c20, d:c21, d:c22,
d:c23, d:c24, d:c25, d:c26, d:c27, a:ca1, a:ca2, a:ca3,
a:ca4, a:ca5, a:ca6, a:ca7, a:ca8, a:ca9')
tblproperties('hbase.table.name' = 'd_cas');
```

Nejprve specifikuji názvy sloupců tabulky v Impala a jejich datové typy jako při tvorbě běžných tabulek. Dále je specifikovaný formát uložení dat, což je v tomto případě tabulka v HBase. Následuje mapování definovaných sloupců Impala tabulky na rodiny sloupců a samotné sloupce tabulky v HBase. Uvádí se pouze názvy sloupců z tabulky v HBase, jejich protějšky v Impala jsou určené pořadím, jak jsou zapsané v definici a následně v popisu mapování. Názvy sloupců v HBase tabulkách nejsou podstatné, protože v rámci této práce se k nim přistupuje pouze pomocí Impala a tam jsou namapované na korektní názvy sloupců. Zvolil jsem tedy takto krátké názvy na základě zmíněného doporučení pro pojmenování z dokumentace HBase. [20]

Na závěr příkazu je definovaný název tabulky v HBase, ze které má vytvořená Impala tabulka brát data. Zbytek tabulek byl vytvořen přizpůsobením uvedeného příkazu podle odpovídajících schémat a názvů tabulek ostatních dimenzí.

4.3.3 Nahrání dat do HBase tabulek

Po vytvoření odpovídajících Impala tabulek namapovaných na tabulky HBase jsem využil tohoto propojení. Dále jsem využil již nahraných dat do Impala tabulek vytvořených v rámci práce na prvním řešení. Propojení Impala a HBase tabulek umožňuje vložení dat do tabulky HBase provedením Hive Query Language příkazu pro vložení dat nad odpovídající Impala tabulkou.

Následujícím příkazem jsem vložil do HBase tabulky data z již existující tabulky v Impala z prvního řešení.

```
insert into table hbase.d_cas select *
from vzp_dw_l2_view_schema.d_cas;
```

Uvedený příkaz vybral data z existující tabulky a vložil je do HBase tabulky pomocí mapování mezi Impala a HBase tabulkou. Pro zbytek tabulek dimenzí jsem nahrání dat provedl analogicky.

Kapitola 5

Měření a vyhodnocení

5.1 Popis způsobu měření

Při měření pro porovnání jednotlivých řešení jsem jako hlavní metriku zvolil dobu odezvy při vyhodnocení analytických dotazů. Pro měření jsem využil poskytnuté typické dotazy ze strany VZP, které jsou popsané a analyzované v sekci 2.4.

Pro vyhodnocení výsledků jednotlivých řešení jsem se zaměřil pouze na měření odezvy příkazů, které data čtou a ne zapisují. Neměřit zapisující operace jsem se rozhodl na základě informací o kontextu tohoto zadání a požadavků na tuto práci v kapitole 2. Zápis dat a měření jeho odezvy je mimo rozsah této práce.

Z původní stovky poskytnutých vzorových dotazů jsem jich pro měření vybral 43. Vynechal jsem dotazy, které po spuštění nad poskytnutou sadou dat nevracely žádné výsledky. To bylo způsobeno tím, že tyto dotazy nebyly přizpůsobené po anonymizaci dat. Kvůli tomu u některých dotazů nebyla nalezena žádná data, která by splnila podmínky v dotazech.

Žádný z vybraných 43 dotazů nemá ve svých podmínkách takové hodnoty, které by vedly k eliminaci čtení některých dat s pomocí partitioning. Do všech vybraných dotazů tedy vstupují data ze všech partition.

Základní informace pro postup při provádění měření jsem čerpal z postupů popsaných ve zdrojích, které vyhodnocují některé z NoSQL technologií. [22][13][30]

Při měření odezvy dotazů v Impala jsem postupoval podle doporučení v dokumentaci k této technologii. [42]

Měření bylo provedeno na poskytnutém serveru, jehož základní specifikace jsou popsané v sekci 2.5.

5.2 Popis dat pro měření

Data pro měření jsem využil ta, která mi byla poskytnuta společností VZP. Jde o data popsaná v sekci 2.2. Základní informace o jednotlivých souborech, které obsahují data tabulek dimenzí a faktové tabulky, jsou v tabulce 5.1. V tabulce je pro každý datový soubor uvedená velikost souboru a počet záznamů.

Název tabulky	Velikost souboru	Počet záznamů	Počet atributů
d_cas	10 kB	32	37
d_druh_dokladu_zp	7 kB	29	19
d_druh_pojisteni_zp	1 kB	3	13
d_odbornost	148 kB	574	24
d_pojistenec	209 MB	453 797	30
d_pracoviste_zp	21 MB	37 007	34
d_uzemni_pracoviste	38 kB	109	36
d_vydajovy_fond	144 kB	604	24
d_zdrav_zarizeni	16 MB	23 982	41
fmt_radek_dokladu_zp	275 MB	554 940	66

Tabulka 5.1: Velikost souboru a počet záznamů pro jednotlivé tabulky

Abych zjistil, jak se jednotlivá řešení chovají s přibývajícím množstvím dat, prováděl jsem měření nad různě velkými faktovými tabulkami. Poskytnutou faktovou tabulku jsem osmkrát rozkopíroval a v každé kopii změnil hodnoty atributu `CAS_KLIC`, podle které je nastavený partitioning. Tak vznikla faktová tabulka skládající se z osmi partition. Tuto tabulku jsem určil jako základní datovou sadu pro měření. V dalších fázích měření jsem tuto základní tabulku duplikoval pomocí vložení dat do sebe samé. Tím jsem vždy získal faktovou tabulku dvojnásobné velikosti oproti té předchozí. Tabulka 5.2 obsahuje velikosti duplikovaných faktových tabulek, na kterých jsem provedl měření. Faktor značí, kolikrát byla tabulka znásobena oproti té základní s osmi partition.

Faktor	Velikost (ve formátu Parquet)	Počet záznamů
1	524 MB	4 439 520
2	1,5 GB	8 879 040
4	2,9 GB	17 758 080
8	5,7 GB	35 516 160
16	11,6 GB	71 032 320
32	23,5 GB	142 064 640

Tabulka 5.2: Velikosti škálované faktové tabulky

Data v tabulkách dimenzí jsem nechal v takovém množství, v jakém byla poskytnuta. U tabulek dimenzí nedávalo smysl škálovat jejich velikosti, jelikož je jejich velikost alespoň řádově stabilní v čase.

5.3 Realizace měření

Pro všechna řešení jsem dobu odezvy na dotazy měřil na 43 poskytnutých vzorových dotazech.

Následují popisy postupu při měření pro jednotlivá řešení.

5.3.1 Impala - obě navrhovaná řešení

Jelikož obě dvě navrhovaná řešení používají Impala jako rozhraní pro zadávání a vyhodnocení dotazů, postup při měření odezvy na dotazy je u těchto dvou řešení analogický.

Základní faktovou tabulku nahranou v sekci 4.2 jsem rozkopíroval tak, aby obsahovala osm partition podle atributu `cas_klic`.

```
CREATE TABLE vzp_dw_l2_view_schema.faktovka1
PARTITIONED BY (cas_klic)
STORED AS PARQUET
AS SELECT * FROM vzp_dw_l2_view_schema.fmt_radek_dokladu_zp;

-- opakovat s různými hodnotami cas_klic (za PARTITION)
-- aby vzniklo 8 partition
INSERT INTO vzp_dw_l2_view_schema.faktovka1
PARTITION (2459156)
SELECT * FROM vzp_dw_l2_view_schema.fmt_radek_dokladu_zp;
```

Prvním příkazem jsem vytvořil tabulku jako kopii poskytnuté tabulky. Tato tabulka obsahovala pouze jednu partition podle atributu `cas_klic`. Druhým příkazem jsem vložil ta samá data do nové partition, která je určena hodnotou v závorce za `PARTITION`.

Vytvoření dalších faktových tabulek, které jsou zdvojením té předchozí, jsem provedl následovně.

```
-- vytvoření tabulky jako kopie té původní
CREATE table vzp_dw_l2_view_schema.faktovka2
PARTITIONED BY (cas_klic)
STORED AS PARQUET
AS SELECT * FROM vzp_dw_l2_view_schema.faktovka1;

-- nahrání dat z tabulky do sebe sama vede ke zdvojení
INSERT INTO vzp_dw_l2_view_schema.faktovka2
PARTITION(cas_klic)
SELECT * FROM vzp_dw_l2_view_schema.faktovka2;
```

Stejným způsobem jsem vytvořil i zbytek faktových tabulek. Zaměnil jsem pouze názvy tabulek. Po tomto kroku jsem měl k dispozici všechny tabulky popsané v tabulce 5.2.

Po vytvoření rozšířených faktových tabulek pro měření jsem spustil příkazy, které shromáždily statistiky o datech v tabulce. Tyto údaje jsou použity optimalizátorem při vyhodnocování dotazů. Jako příklad níže uvádím příkaz pro výpočet statistik pro tabulku `faktovka1`. Pro ostatní tabulky jsem výpočet statistik provedl analogicky, pouze se změnou názvu tabulky.

```
compute stats faktofka1;
```

Dál jsem pokračoval spouštěním dotazů. Všechny dotazy jsem zapsal do souboru, který jsem uložil na jeden z kontejnerizovaných uzlů clusteru. Jedná se o jeden z uzlů, který funguje jako sekundární uzel.

Pro vyhodnocení dotazů jsem využil rozhraní Impala pro příkazovou řádku impala-shell. Na doporučení dokumentace [42] jsem použil přepínače, které zajistí, aby se vypsal pouze zjednodušený výstup, který se zapíše do souboru. Tím jsem předešel potenciálnímu zpoždění způsobenému vypisováním dlouhých podrobných výsledků přímo do konzole.

Celou sadu dotazů uloženou do souboru `query.sql` jsem spustil pomocí následujícího příkazu.

```
impala-shell -B -o output.txt -c -f ./query.sql
```

Přepínač `-B` slouží k tomu, aby se výsledky dotazu vypsaly v jednoduchém formátu. Přepínač `-o` umožňuje zapsat výsledky dotazů do souboru, který je specifikovaný za přepínačem. Možnost `-c` zajišťuje, že se Impala pokusí o vyhodnocení všech zadaných příkazů. To i v případě, kdy některý z příkazů selže. Pomocí `-f` lze aplikaci předložit soubor obsahující dotazy.

Postup pro řešení, které používá HBase pro uložení tabulek dimenzí, je analogický. Ve spouštěných dotazech pouze stačí změnit databázi u tabulek dimenzí z `vzp_dw_l2_view_schema` na `hbase`. Faktová tabulka je pro dotazy v HBase stejná.

Všechny dotazy použité při měření jsou uložené na přiloženém paměťovém médiu ve složce `/src/prakticka_cast/dotazy` (Příloha B).

5.3.2 Oracle - původní řešení

Pro práci spojenou s měřením prováděným nad databází Oracle jsem použil nástroj Oracle SQL Developer. Ten umožňuje vytvořit připojení k databázi a provádět nad ní příkazy pro dotazování nebo manipulaci s daty. [43]

V Oracle SQL Developer jsem si vytvořil připojení do databáze běžící na stejném serveru, kde jsou spuštěna i má další dvě řešení. Spolu s ukázkovými daty mi byly poskytnuty i SQL skripty, s jejichž pomocí lze vytvořit příslušné tabulky.

V Oracle databázi jsem spuštěním poskytnutých skriptů vytvořil všechny potřebné tabulky. S pomocí rohraní Oracle SQL Developer jsem do nich importoval poskytnutá data. Po nahrání dat jsem s pomocí poskytnutých skriptů v tabulkách vytvořil indexy a cizí klíče tak, aby odpovídaly těm skutečně používaným.

Tabulky dimenzí byly v tomto bodě hotové a připravené pro měření. Faktovou tabulku ještě zbývalo rozkopírovat do více partition a následně vytvořit škálované kopie faktové tabulky. U faktové tabulky je nastavený partitioning podle atributu `cas_klic`.

Faktovou tabulku jsem vytvořil tak, aby obsahovala potřebné partition, do kterých jsem následně nahrál data. Data do nových partition jsem vložil pomocí výběru dat z již existující partition, kterou jsem zaplnil importem poskytnutých dat v souboru CSV.

Po provedení předchozího kroku již faktová tabulka obsahem odpovídá té, která byla použita jako základ pro první měření v dalších řešeních. Pro další měření jsem opět škáloval velikost faktové tabulky vkládáním dat z ní do sebe samé.

Po vytvoření všech tabulek v Oracle databázi jsem spustil poskytnuté skripty, které vypočítají statistiky o datech v tabulkách.

Všechny poskytnuté skripty i SQL příkazy pro provedení výše popsaných operací jsou uložené na paměťovém médiu ve složce
`/src/prakticka_cast/oracle_vytvoreni` (Příloha B).

5.4 Výsledky

Sadu dotazů jsem spouštěl několikrát pro každou velikost faktové tabulky. Pro faktové tabulky s faktorem 1, 2 a 4 jsem prováděl měření pětkrát a pro faktové tabulky s faktorem 8, 16 a 32 třikrát. V tabulkách a grafech níže jsou zapsané průměrné časy z těchto měření.

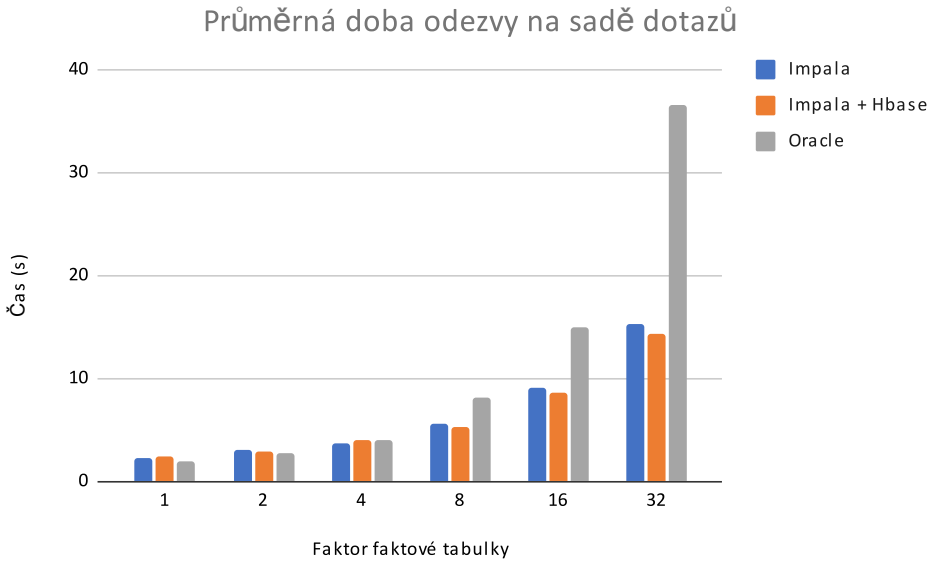
V tabulce 5.3 jsou průměrné časy odezvy na dotazy naměřené po několikanásobném spuštění sady 43 dotazů. Uvedené časy odpovídají průměrnému času na vyhodnocení jednoho dotazu ze sady. V řádcích tabulky jsou naměřené hodnoty pro jednotlivé velikosti faktové tabulky (velikosti jsou popsané v tabulce 5.2). První sloupec tabulky obsahuje faktor faktorvé tabulky, který určuje, kolikrát byla základní faktová tabulka znásobena. Další tři sloupce pak obsahují naměřené průměrné časy pro jednotlivé technologie.

Průměr je vypočítán z průměrů jednotlivých měření. Průměr jednoho měření je celková odezva na sadu dotazů vydělená počtem dotazů, tedy 43. Jednotlivé časy odezvy pro každý dotaz z každého měření jsou dostupné na přiloženém paměťovém médiu ve složce `/vysledky-mereni` (Příloha B).

Faktor faktové tabulky	Impala + Parquet	Impala + HBase	Oracle
1	2,3 s	2,5 s	1,9 s
2	3,0 s	2,9 s	2,7 s
4	3,7 s	4,0 s	4,0 s
8	5,6 s	5,2 s	8,2 s
16	9,1 s	8,6 s	15,0 s
32	15,3 s	14,3 s	36,5 s

Tabulka 5.3: Průměrná doba odezvy na sadě dotazů

Obrázek 5.1 je grafické znázornění tabulky 5.3. Ukazuje vývoj průměrné odezvy na sadu dotazů při rostoucí velikosti tabulky. S pomocí grafu také můžeme porovnat výsledky jednotlivých řešení, které jsou odlišené barevně.



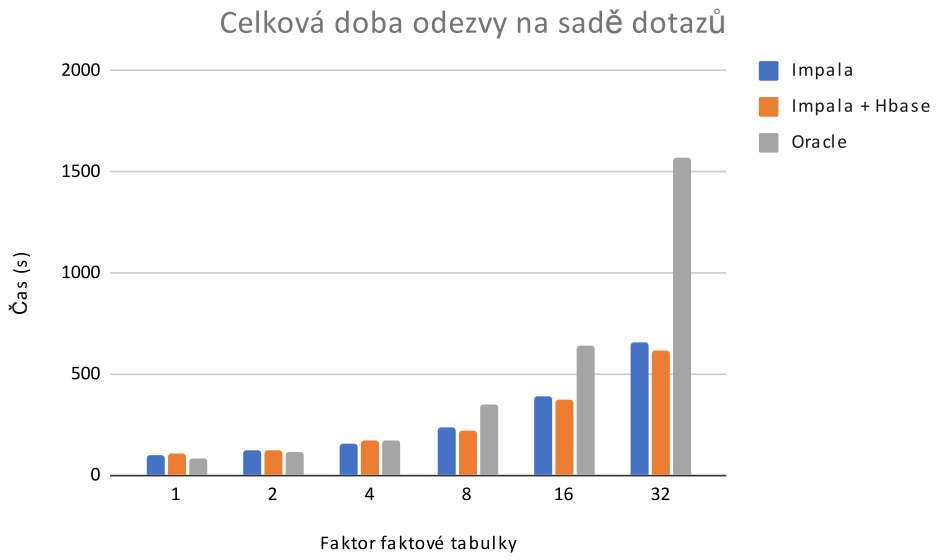
Obrázek 5.1: Průměrná doba odezvy na sadě dotazů

Tabulka 5.4 obsahuje průměrnou celkovou dobu odezvy pro vyhodnocení celé sady 43 dotazů. Průměr je počítán z více provedených měření. V řádcích tabulky jsou průměrné hodnoty z celkové doby měření na sadě dotazů. První sloupec obsahuje faktor faktové tabulky (viz tabulka 5.2). Zbylé tři sloupce obsahují naměřené hodnoty pro jednotlivé technologie.

Obrázek 5.2 graficky zobrazuje data z tabulky 5.4. V grafu lze vidět vývoj celkového času odezvy na sadu dotazů při rostoucí velikosti faktové tabulky. Lze také porovnat jednotlivá řešení.

Faktor faktové tabulky	Impala + Parquet	Impala + HBase	Oracle
1	100,1 s	106,0 s	81,7 s
2	128,9 s	124,9 s	117,0 s
4	158,7 s	174,1 s	170,5 s
8	241,4 s	225,6 s	353,4 s
16	390,6 s	371,9 s	643,5 s
32	659,0 s	614,5 s	1569,8 s

Tabulka 5.4: Celková doba odezvy na sadě dotazů



Obrázek 5.2: Celková doba odezvy na sadě dotazů

Z naměřených dat vychází v experimentech s menšími faktovými tabulkami (faktor 1, 2 a 4) lépe původní řešení v databázi Oracle. S roustoucí velikostí faktové tabulky (faktor 8, 16) si vedou lépe obě řešení v Impala. V experimentech s největší testovanou faktovou tabulkou pak obě řešení v Impala dosahují více než dvojnásobně lepších časů.

V detailních datech z průběhu měření jsem identifikoval jeden dotaz ze sady, jehož vyhodnocení v databázi Oracle trvá výrazně déle než v Impala. Vyhodnocení tohoto dotazu v Oracle trvá také výrazně déle v porovnání s odezvou na ostatní dotazy ze sady. (Detailní výsledky měření jsou dostupné na přiloženém paměťovém médiu ve složce `/vysledky-mereni` (Příloha B)).

Průměrné časy na vyhodnocení tohoto dotazu jsou v tabulce 5.5.

Faktor faktové tabulky	Impala + Parquet	Impala + HBase	Oracle
1	2,8 s	7,2 s	9,6 s
2	4,4 s	12,9 s	12,8 s
4	6,1 s	24,4 s	22,2 s
8	10,7 s	10,7 s	71,7 s
16	19,2 s	19,4 s	126,7 s
32	37,2 s	36,9 s	526,7 s

Tabulka 5.5: Doba odezvy na dotaz problémový pro Oracle

Z dat v tabulce 5.5 je patrné, že tento dotaz nad velkými faktovými tabulkami v databázi Oracle trvá výrazně déle než v Impala. V řešení Impala s HBase byla odezva také větší v případě menších faktových tabulek. To si vysvětluji špatným rozhodnutím optimalizátoru při vyhodnocení dotazu, kde nad velkými faktovými tabulkami následně zvolil lepší strategii.

Po porovnání s tabulkami 5.4 a 5.3 je zřejmé, že tento dotaz výrazně převyšuje průměrnou délku vyhodnocení dotazu v Oracle. Je také patrné, že doba odezvy na tento čas tvoří velkou část celkové doby odezvy na celou sadu dotazů. V případě největší faktové tabulky to je téměř třetina celkové odezvy na dotazy.

Provedl jsem základní šetření tohoto dotazu a nezjistil žádnou výraznou odlišnost z pohledu struktury SQL. Dále jsem si nechal vypsat exekuční plán tohoto dotazu v Oracle. Ani z toho jsem nezjistil žádnou příčinu, proč je odezva na tento dotaz v Oracle tak velká. Tento dotaz je dostupný v příloze C.1.

Dotaz analyzovaný výše svou dobou odezvy výrazně ovlivňuje souhrnné výsledky řešení v databázi Oracle. Doba odezvy na jiné dotazy ze sady se od ostatních takto výrazně neliší. Proto jsem se rozhodl prezentovat navíc alternativní výsledky, ze kterých jsem tento dotaz vynechal.

Tabulky 5.6 a 5.7 a obrázky 5.3 a 5.4 obsahují souhrnná data z měření po vynechání zmíněného dotazu.

Faktor faktové tabulky	Impala + Parquet	Impala + HBase	Oracle
1	2,3 s	2,4 s	1,7 s
2	3,0 s	2,7 s	2,4 s
4	3,6 s	3,6 s	3,4 s
8	5,5 s	5,1 s	6,6 s
16	8,8 s	8,4 s	12,0 s
32	14,8 s	13,8 s	24,3 s

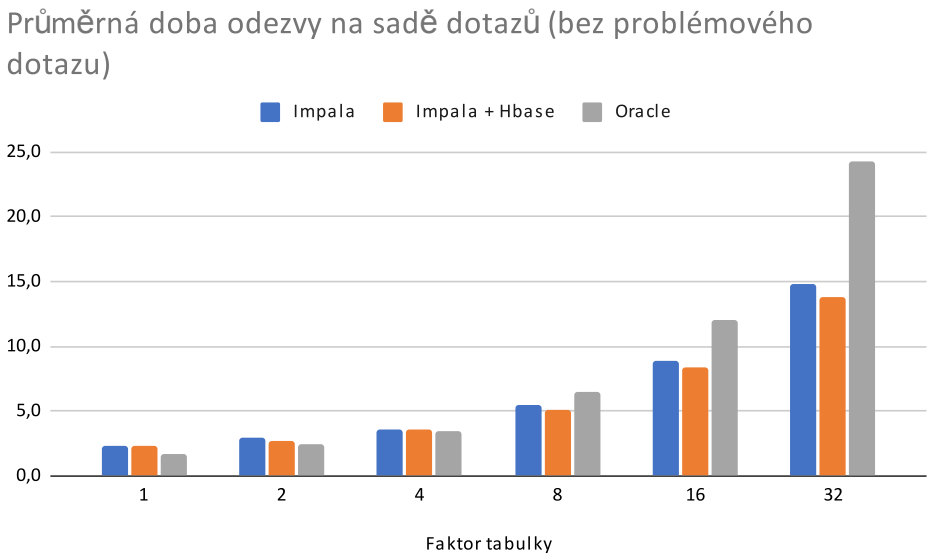
Tabulka 5.6: Průměrná doba odezvy na sadě dotazů (bez „problémového“ dotazu)

Faktor faktové tabulky	Impala + Parquet	Impala + HBase	Oracle
1	97,3 s	98,8 s	72,2 s
2	124,5 s	112,0 s	104,3 s
4	152,6 s	149,7 s	148,2 s
8	230,8 s	214,9 s	281,7 s
16	371,4 s	352,5 s	516,8 s
32	621,7 s	577,5 s	1043,1 s

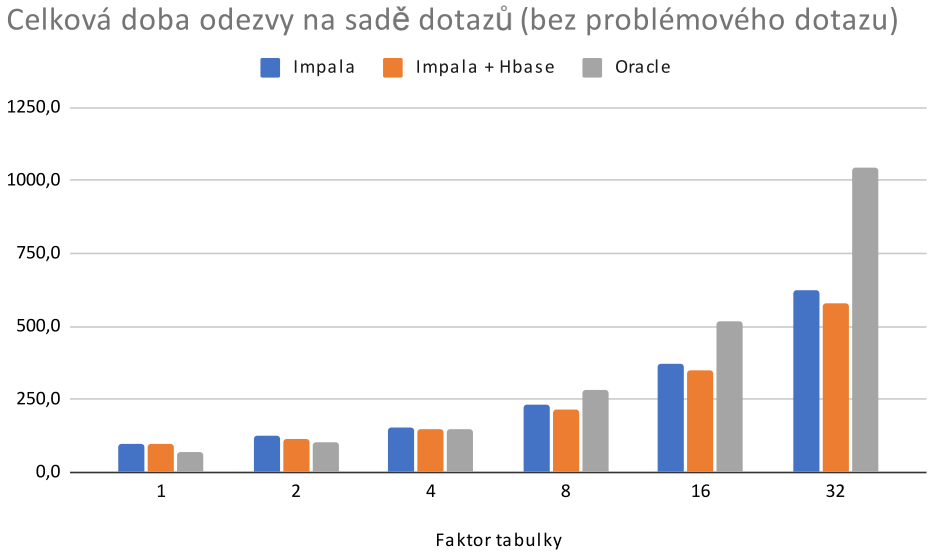
Tabulka 5.7: Celková doba odezvy na sadě dotazů (bez „problémového“ dotazu)

Vynechání zmiňovaného dotazu mírně posiluje lepší výkon databáze Oracle pro první tři velikosti faktové tabulky. V následujících dvou větších velikostech si stále vedou lépe řešení v Impala, ale doba odezvy již v porovnání s Oracle není výrazně menší. Na největší faktové tabulce si stále výrazně lépe vedou řešení s Impala.

Po vynechání dotazu se také mírně zvětšil rozdíl mezi řešením Impala (s daty v Parquet) a Impala s HBase.



Obrázek 5.3: Průměrná doba odezvy na sadě dotazů (bez „problémového“ dotazu)



Obrázek 5.4: Celková doba odezvy na sadě dotazů (bez „problémového“ dotazu)

5.5 Vyhodnocení výsledků

Z naměřených dat v tabulkách 5.4 a 5.3 (i 5.6 a 5.7 po vynechání „problémového“ dotazu) lze vyčíst, že aktuální řešení v databázi Oracle si vede lépe při menších velikostech faktové tabulky. S rostoucí velikostí faktové tabulky se řešení v Impala vyrovnávají řešení v Oracle. S dotazy nad největšími faktovými tabulkami už si řešení v Impala poradí výrazně lépe než to v Oracle.

Pokud by velikost dat ve faktové tabulce (zejména jejích partition, které vstupují do dotazu) odpovídala velikosti největší faktové tabulky použité při měření, tak vychází Impala jako lepší řešení pro tuto část datového skladu. Nutno ale podotknout, že řešení pomocí Impala přináší nové nároky na transformaci a načítání dat. Další přidané nároky souvisí s instalací a administrací nových nástrojů. Řešení v Impala bych tedy volil až v případě velikosti dat přesahující velikost největší faktové tabulky využité při měření.

Co se týká porovnání řešení Impala (s daty v Parquet na HDFS) a Impala s HBase, tak si o trochu lépe vedlo řešení s HBase. Ovšem s přihlédnutím na nároky spojené se správou další technologie a transformacemi dat do HBase bych doporučil spíše první řešení v Impala s daty ve formátu Parquet uloženými na HDFS.

Za zmínku stojí také hardware infrastruktura, na které bylo měření prováděné. Při použití jednoho vysoce výkonného stroje bych radil zůstat u řešení v Oracle, které je pro využití takového hardware vyvinuté. Pro řešení v Impala by bylo vhodné využití clusteru sestaveného z více propojených výpočetních strojů.

Závěr

Cílem této práce bylo seznámit se s prostředím datového skladu ve VZP, zejména s jeho částí, která slouží jako zdroj dat pro BI aplikaci. Další cíl byl provedení analýzy typických dotazů, které uživatelé pomocí BI aplikace tvoří. Třetí požadavek na tuto práci byl výběr NoSQL technologií, které je možné propojit s BI aplikací a dále provést jejich rešerši. Posledním cílem bylo na základě provedené rešerše zvolit dvě technologie, zprovoznit je v testovacím prostředí a provést na nich měření pro porovnání s původním řešením.

S prostředím datového skladu VZP jsem se seznámil v průběhu konzultací se zadavatelem této práce. Získal jsem základní informace o stávajícím řešení a o případech užití části datového skladu, která poskytuje data BI aplikaci. Dále mi zadavatel práce poskytl sadu ukázkových dotazů, které jsou uživateli běžně zadávané v BI aplikaci. Z analýzy těchto dotazů jsem zjistil společné vlastnosti těchto dotazů. Po získání základních informací o chodu datového skladu a analýze dotazů jsem měl k dispozici informace, ze kterých vyplynuly požadavky na zvažované technologie.

Se zjištěními z předchozích kroků jsem vybral pět NoSQL databázových technologií, které jsou pod open-source licencí a je možné je propojit s Oracle Analytics Server. Dále jsem analyzoval základní vlastnosti těchto pěti technologií a studoval další zdroje, ve kterých byly tyto technologie testované při použití v datovém skladu.

Na základě rešerše jsem zvolil technologii Impala s daty tabulek uloženými ve formátu Parquet v HDFS. Další technologii jsem zvolil Impala s daty tabulek dimenzí uloženými v databázi HBase. Tyto dvě technologie jsem zprovoznil na poskytnutém testovacím serveru a nahrál do nich data pro měření. Následně jsem provedl měření, které zkoumá dobu odezvy na typické dotazy a jak se tato odezva mění s rostoucím objemem dat ve faktové tabulce. Všechny stanovené cíle práce, které jsou shrnuté v prvním odstavci, byly splněny.

Z výslekdů měření jsem došel k závěru, že při velkých objemech dat si na testovacím serveru vybraná NoSQL řešení vedou lépe než řešení v Oracle. Řešení s pomocí Impala a HBase si vedlo jen mírně lépe než řešení v Impala se všemi daty ve formátu Parquet. Pokud vezmu v potaz nároky na správu další technologie, tak se řešení s HBase nevyplatí v porovnání s druhým navrhovaným řešením. S ohledem na nároky na zavedení nové technologie a migrování dat do ní bych se přiklonil k současnému řešení v Oracle. Tomu nahrává také současný stav hardware infrastruktury pro datový sklad, která je přizpůsobená pro databázi Oracle. Z měření lze ale vypozorovat trend, že s přibývajícím objemem dat si vybrané technologie vedou lépe než původní řešení. V případě přibývajícího množství dat bych vybrané technologie dále zvažoval.

Na tuto práci je možné navázat dalšími experimenty na vybraných technologiích, které jsou nyní zprovozněné v testovacím prostředí VZP. V rozsahu této práce nebyla optimalizace vybraných technologií pro vybrané dotazy. Dalo by se tedy navázat například návrhem datových schémat pro použité technologie, které jsou optimalizované pro typické dotazy.

Bibliografie

1. NEGI, M. *Fundamental of Database Management System: Learn essential concepts of database systems*. BPB PUBN, 2019. ISBN 9789388176620.
2. ORACLE. Database Concepts [online]. 2017 [cit. 2021-09-15]. Dostupné z: `https://docs.oracle.com/database/121/CNCPT/toc.html`.
3. HARRINGTON, J.L. *Relational Database Design and Implementation*. Elsevier Science, 2016. ISBN 9780128499023.
4. LIGHTSTONE, S.S.; TEOREY, T.J.; NADEAU, T. *Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More*. Elsevier Science, 2010. The Morgan Kaufmann Series in Data Management Systems. ISBN 9780080552316.
5. GROSSMANN, W.; RINDERLE-MA, S. *Fundamentals of Business Intelligence*. Springer Berlin Heidelberg, 2015. Data-Centric Systems and Applications. ISBN 9783662465318.
6. KIMBALL, R.; ROSS, M. *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*. Wiley, 2013. ISBN 9781118530801.
7. HARRISON, G. *Next Generation Databases NoSQL, NewSQL, and Big Data*. Apress, 2015. ISBN 9781484213308.
8. TIWARI, S. *Professional Nosql*. Wiley India Pvt. Limited. ISBN 9788126533268.
9. FRIESEN, J. *Java XML and JSON: Document Processing for Java SE*. Apress, 2019. ISBN 9781484243299.
10. ORACLE. Oracle Database 19c [online]. 2021 [cit. 2021-11-14]. Dostupné z: `https://docs.oracle.com/en/database/oracle/oracle-database/19/index.html`.
11. ORACLE. About Oracle Analytics Server [online]. 2021 [cit. 2021-11-14]. Dostupné z: `https://docs.oracle.com/en/middleware/bi/analytics-server/user-oas/oracle-analytics-server.html`.

12. THE APACHE SOFTWARE FOUNDATION. Cassandra Documentation. *Apache Cassandra* [online]. 2021 [cit. 2021-10-13]. Dostupné z: https://cassandra.apache.org/doc/latest/.

13. CHEBOTKO, Artem; KASHLEV, Andrey; LU, Shiyong. A Big Data Modeling Methodology for Apache Cassandra. *2015 IEEE International Congress on Big Data*. 2015, s. 238–245.

14. THE APACHE SOFTWARE FOUNDATION. Apache Hive [online]. 2020 [cit. 2021-10-15]. Dostupné z: cwiki.apache.org/confluence/display/Hive/Home.

15. KUMAR, N. *Big Data Using Hadoop and Hive*. Mercury Learning & Information, 2021. ISBN 9781683926450.

16. THE APACHE SOFTWARE FOUNDATION. HDFS Architecture [online]. 2021 [cit. 2021-10-15]. Dostupné z: https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html.

17. THE APACHE SOFTWARE FOUNDATION. MapReduce Tutorial [online]. 2021 [cit. 2021-10-15]. Dostupné z: https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

18. THE APACHE SOFTWARE FOUNDATION. Apache Hadoop YARN [online]. 2021 [cit. 2021-10-15]. Dostupné z: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html.

19. THE APACHE SOFTWARE FOUNDATION. Spark Overview [online]. 2021 [cit. 2021-10-15]. Dostupné z: https://spark.apache.org/docs/latest/.

20. APACHE HBASE TEAM. Apache HBase Reference Guide [online]. 2021 [cit. 2021-10-20]. Dostupné z: hbase.apache.org/book.html.

21. VOHRA, D. *Apache HBase Primer*. Apress, 2016. ISBN 9781484224243.

22. CARVALHO SCABORA, Lucas de; BRITO, Jaqueline Joice; CIFERRI, Ricardo Rodrigues; AGUIAR CIFERRI, Cristina Dutra de. Physical Data Warehouse Design on NoSQL Databases - OLAP Query Processing over HBase. In: *ICEIS*. 2016.

23. CLOUDERA, INC. Apache Impala - Interactive SQL [online]. 2021 [cit. 2021-09-20]. Dostupné z: https://docs.cloudera.com/documentation/enterprise/6/6.3/topics/impala.html.

24. CLOUDERA, INC. Using the Parquet File Format with Impala Tables [online]. 2021 [cit. 2021-09-20]. Dostupné z: https://docs.cloudera.com/documentation/enterprise/6/6.3/topics/impala_parquet.html.

25. CLOUDERA, INC. Components of the Impala Server [online]. 2021 [cit. 2021-09-20]. Dostupné z: https://docs.cloudera.com/documentation/enterprise/6/6.3/topics/impala_components.html.

26. CLOUDERA, INC. Impala Performance - Tuning SQL queries [online]. 2020 [cit. 2021-09-22]. Dostupné z: https://docs.cloudera.com/best-practices/latest/impala-performance/topics/bp-impala-tuning-sql-queries.html.

27. CLOUDERA, INC. Using Impala to Query HBase Tables [online]. 2021 [cit. 2021-09-22]. Dostupné z: https://docs.cloudera.com/documentation/enterprise/5-8-x/topics/impala_hbase.html.

28. MONGODB, INC. Introduction to MongoDB [online]. 2021 [cit. 2021-09-30]. Dostupné z: https://docs.mongodb.com/manual/introduction/.

29. MEMBREY, P.; HOWS, D.; PLUGGE, E. *MongoDB Basics*. Apress, 2014. Expert's voice in open source. ISBN 9781484208953.

30. CHAVALIER, Max; EL MALKI, Mohammed; KOPLIKU, Arlind; TESTE, Olivier; TOURNIER, Ronan. Document-oriented data warehouses: Models and extended cuboids, extended cuboids in oriented document. In: *2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS)*. 2016, s. 1–11.

31. CLOUDERA, INC. CDH Overview [online]. 2021 [cit. 2021-10-19]. Dostupné z: https://docs.cloudera.com/documentation/enterprise/5-8-x/topics/cdh_intro.html.

32. CLOUDERA, INC. Cloudera Manager 5 Overview [online]. 2021 [cit. 2021-10-19]. Dostupné z: https://docs.cloudera.com/documentation/enterprise/5-8-x/topics/cm_intro_primer.html.

33. CLOUDERA, INC. Get Started with Hue [online]. 2021 [cit. 2021-10-19]. Dostupné z: https://docs.cloudera.com/documentation/enterprise/5-8-x/topics/hue.html.

34. CITRIX SYSTEMS, INC. What is containerization and how does it work? [online]. 2021 [cit. 2021-10-23]. Dostupné z: https://www.citrix.com/solutions/app-delivery-and-security/what-is-containerization.html.

35. DOCKER, INC. Docker overview [online]. 2021 [cit. 2021-10-23]. Dostupné z: https://docs.docker.com/get-started/overview/.

36. ORACLE. Welcome to VirtualBox.org! [online]. 2021 [cit. 2021-10-24]. Dostupné z: https://www.virtualbox.org.

37. WINSCP.NET. Introducing WinSCP [online]. 2017 [cit. 2021-10-23]. Dostupné z: https://winscp.net/eng/docs/introduction.

38. cloudera/clusterdock [online]. 2016 [cit. 2021-10-20]. Dostupné z: https://hub.docker.com/r/cloudera/clusterdock.

39. THE APACHE SOFTWARE FOUNDATION. PySpark Documentation [online] [cit. 2021-10-26]. Dostupné z: https://spark.apache.org/docs/latest/api/python/.

40. DATABRICKS. CSV Data Source for Apache Spark 1.x [online]. 2017 [cit. 2021-11-03]. Dostupné z: https://github.com/databricks/spark-csv.

41. THE APACHE SOFTWARE FOUNDATION. Using Apache Commons CSV [online]. 2021 [cit. 2021-10-26]. Dostupné z: https://commons.apache.org/proper/commons-csv//.

42. CLOUDERA, INC. Benchmarking Impala Queries [online]. 2021 [cit. 2021-11-13]. Dostupné z: https://docs.cloudera.com/documentation/enterprise/6/6.3/topics/impala_perf_benchmarking.html.

43. ORACLE. Oracle SQL Developer [online]. 2021 [cit. 2021-11-25]. Dostupné z: https://www.oracle.com/database/technologies/appdev/sqldeveloper-landing.html.

Příloha A

Seznam použitých zkratek

ACID Atomicity, Consistency, Isolation, Durability

BI Business Intelligence

BSON Binary JSON

CDH Cloudera Distribution Hadoop

CSV Comma Separated Values

CQL Cassandra Query Language

DBMS Database Management System

ETL Extract, Transform, Load

HDFS Hadoop Distributed File System

HQL Hive Query Language

JSON JavaScript Object Notation

OLAP Online Analytical Processing

SQL Structured Query Language

VZP Všeobecná zdravotní pojišťovna

XML Extensible Markup Language

Příloha **B**

Obsah přiloženého paměťového nosiče

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `prakticka_cast` ..... zdrojové kódy implementace
    - `dotazy` ..... sada dotazů použitá při měření
      - `hbase` ..... sada dotazů pro HBase
      - `impala` ..... sada dotazů pro Impala
      - `oracle` ..... sada dotazů pro Oracle
    - `nacteni_dat_nosql`
      - `hbase` ..... vytvoření tabulek pro řešení s HBase
      - `impala` ..... vytvoření tabulek v Impala
      - `transformace_pyspark` ..... transformace dat v PySpark
    - `oracle_vytvoreni` ... vytvoření tabulek a dalších objektů v Oracle
  - `prace` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `dp-nemecst4.pdf` ..... text práce ve formátu PDF
- `vysledky-mereni`
  - `vysledky-mereni.xlsx` ..... detailní výsledky měření

Příloha C

Ukázkové dotazy

```
WITH SAWITH0 AS
  (
    select
      sum(T19805.RADDOK_UHR_OSTATNI_CASTKA)              as c1
    , sum(T19805.RADDOK_UHR_MATERIAL_CASTKA)             as c2
    , sum(T19805.RADDOK_UHR_CELKEM_CASTKA)               as c3
    , sum(T19805.RADDOK_UHR_BODY_POCET)                  as c4
    , sum(T19805.RADDOK_UHR_BODY_CASTKA)                 as c5
    , 1                                                  as c6
    , T17608.CAS_MESIC_KOD                               as c7
    , concat(concat(T3442.ZZ_KOD, ' - '), T3442.ZZ_NAZEV) as c8
    from
      FIT_DW_L2_SCHEMA.D_UZEMNI_PRACOVISTE_NEW T17992
      /* D_UZEMNI_PRACOVISTE_NEW_POSKYT */
    , FIT_DW_L2_SCHEMA.D_CAS_NEW T17686
      /* D_CAS_NEW_PROVEDENI */
    , FIT_DW_L2_SCHEMA.D_CAS_NEW       T17608
    , FIT_DW_L2_SCHEMA.D_ODBORNOST_NEW T3840
      /* D_ODBORNOST_NEW_POSKYT */
    , FIT_DW_L2_SCHEMA.D_ZDRAV_ZARIZENI_NEW T3442
      /* D_ZDRAV_ZARIZENI_NEW_POSKYT */
    , FIT_DW_L2_SCHEMA.D_DRUH_POJISTENI_ZP_NEW  T3310
    , FIT_DW_L2_SCHEMA.FMT_RADEK_DOKLADU_ZP_TEST T19805
    where
      (
        T3310.DRPOJ_KLIC = T19805.DRPOJ_KLIC
        and T3310.DRPOJ_KOD = '1'
        and T3442.ZZ_KLIC = T19805.ZZ_KLIC_POSKYT
        and T3840.O_KLIC = T19805.O_KLIC_POSKYT
        and T17608.CAS_KLIC = T19805.CAS_KLIC
```

```
        and T17686.CAS_KLIC = T19805.CAS_KLIC_PROVEDENI
        and T17992.UP_KLIC = T19805.UP_KLIC_POSKYT
        and T19805.RADDOK_SMAZANO_PRIZ = 'N'
        and concat(concat(T3840.O_KOD, ' - '), T3840.O_NAZEV)
        = '001 - všeobecné praktické lékařství'
        and concat(concat(T17992.UP_RP_KOD, ' - '), T17992.UP_RP_NAZEV)
        = '2 - RP Plzeň pro Jihočeský, Karlovarský a Plzeňský kraj'
        and T3442.ZZ_A_ZDROJ_SYSTEM <> 'ZDROJ_C'
        and T19805.RADDOK_A_ZDROJ_SYSTEM <> 'PECE_EU'
        and T19805.RADDOK_TYP_DAVKY_INT_KOD <> 'F'
        and
        (
          T17686.CAS_ROK_KOD in ('2017'
                               , '2018'
                               , '2020')
        )
        and T17608.CAS_MESIC_KOD between '2020/11' and '2020/11'
      )
    group by
      T17608.CAS_MESIC_KOD
    , concat(concat(T3442.ZZ_KOD, ' - '), T3442.ZZ_NAZEV)
  )
select
  D1.c1 as c1, D1.c2 as c2, D1.c3 as c3, D1.c4 as c4,
  D1.c5 as c5, D1.c6 as c6, D1.c7 as c7, D1.c8 as c8,
  D1.c9 as c9
from
  (
    select distinct
      0     as c1, D1.c6 as c2, D1.c7 as c3, D1.c8 as c4,
      D1.c5 as c5, D1.c4 as c6, D1.c3 as c7, D1.c2 as c8,
      D1.c1 as c9
    from
      SAWITH0 D1
    order by
      c4, c2, c3
  )
  D1
;
```

```
WITH SAWITH0 AS
  (
    select
      sum(T19805.RADDOK_UHR_OSTATNI_CASTKA)                          as c1
    , sum(T19805.RADDOK_UHR_MATERIAL_CASTKA)                         as c2
    , sum(T19805.RADDOK_UHR_CELKEM_CASTKA)                           as c3
    , sum(T19805.RADDOK_UHR_BODY_POCET)                              as c4
    , sum(T19805.RADDOK_UHR_BODY_CASTKA)                             as c5
    , 1                                                              as c6
    , T17686.CAS_ROK_KOD                                             as c7
    , concat(concat(T3669.VYDFOND_ID, ' - '), T3669.VYDFOND_NAZEV)   as c8
    , concat(concat(T3442.ZZ_KOD, ' - '), T3442.ZZ_NAZEV)            as c9
    from
      FIT_DW_L2_SCHEMA.D_CAS_NEW T17686
      /* D_CAS_NEW_PROVEDENI */
    , FIT_DW_L2_SCHEMA.D_CAS_NEW       T17608
    , FIT_DW_L2_SCHEMA.D_ODBORNOST_NEW T3840
      /* D_ODBORNOST_NEW_POSKYT */
    , FIT_DW_L2_SCHEMA.D_VYDAJOVY_FOND_NEW  T3669
    , FIT_DW_L2_SCHEMA.D_ZDRAV_ZARIZENI_NEW T3442
      /* D_ZDRAV_ZARIZENI_NEW_POSKYT */
    , FIT_DW_L2_SCHEMA.D_DRUH_POJISTENI_ZP_NEW  T3310
    , FIT_DW_L2_SCHEMA.FMT_RADEK_DOKLADU_ZP_TEST T19805
    where
      (
        T3310.DRPOJ_KLIC = T19805.DRPOJ_KLIC
        and T3310.DRPOJ_KOD = '1'
        and T3442.ZZ_KLIC = T19805.ZZ_KLIC_POSKYT
        and T3669.VYDFOND_KLIC = T19805.VYDFOND_KLIC
        and T3840.O_KLIC = T19805.O_KLIC_POSKYT
        and T17608.CAS_KLIC = T19805.CAS_KLIC
        and T17686.CAS_KLIC = T19805.CAS_KLIC_PROVEDENI
        and T17686.CAS_ROK_KOD = '2020'
        and T19805.RADDOK_SMAZANO_PRIZ = 'N'
        and concat(concat(T3840.O_KOD, ' - '), T3840.O_NAZEV)
        = '720 - paliativní medicína'
        and T3442.ZZ_A_ZDROJ_SYSTEM <> 'ZDROJ_C'
        and T19805.RADDOK_A_ZDROJ_SYSTEM <> 'PECE_EU'
        and T19805.RADDOK_TYP_DAVKY_INT_KOD <> 'F'
        and T17608.CAS_MESIC_KOD between '2020/11' and '2020/11'
      )
    group by
      T17686.CAS_ROK_KOD
    , concat(concat(T3442.ZZ_KOD, ' - '), T3442.ZZ_NAZEV)
```

```
    , concat(concat(T3669.VYDFOND_ID, ' - '), T3669.VYDFOND_NAZEV)
  )
select
  D1.c1  as c1, D1.c2  as c2, D1.c3  as c3, D1.c4  as c4,
  D1.c5  as c5, D1.c6  as c6, D1.c7  as c7, D1.c8  as c8,
  D1.c9  as c9, D1.c10 as c10
from
  (
    select distinct
      0     as c1, D1.c6 as c2, D1.c7 as c3, D1.c8 as c4,
      D1.c9 as c5, D1.c5 as c6, D1.c4 as c7, D1.c3 as c8,
      D1.c2 as c9, D1.c1 as c10
    from
      SAWITH0 D1
    order by
      c5, c4, c2, c3
  )
  D1


;
```

C.1 Dotaz problémový pro Oracle

```
WITH SAWITH0 AS
  (
    select
      sum(T19805.RADDOK_UHR_OSTATNI_CASTKA)               as c1
    , sum(T19805.RADDOK_UHR_MATERIAL_CASTKA)              as c2
    , sum(T19805.RADDOK_UHR_CELKEM_CASTKA)                as c3
    , sum(T19805.RADDOK_UHR_BODY_POCET)                   as c4
    , sum(T19805.RADDOK_UHR_BODY_CASTKA)                  as c5
    , 1                                                   as c6
    , T17686.CAS_ROK_KOD                                  as c7
    , concat(concat(T3442.ZZ_KOD, ' - '), T3442.ZZ_NAZEV) as c8
    from
      FIT_DW_L2_SCHEMA.D_CAS_NEW T17686
      /* D_CAS_NEW_PROVEDENI */
    , FIT_DW_L2_SCHEMA.D_CAS_NEW            T17608
    , FIT_DW_L2_SCHEMA.D_ZDRAV_ZARIZENI_NEW T3442
      /* D_ZDRAV_ZARIZENI_NEW_POSKYT */
    , FIT_DW_L2_SCHEMA.D_DRUH_POJISTENI_ZP_NEW  T3310
    , FIT_DW_L2_SCHEMA.FMT_RADEK_DOKLADU_ZP_TEST T19805
    where
      (
        T3310.DRPOJ_KLIC = T19805.DRPOJ_KLIC
        and T3310.DRPOJ_KOD = '1'
        and T3442.ZZ_KLIC = T19805.ZZ_KLIC_POSKYT
        and T17608.CAS_KLIC = T19805.CAS_KLIC
        and T17686.CAS_KLIC = T19805.CAS_KLIC_PROVEDENI
        and T19805.RADDOK_SMAZANO_PRIZ = 'N'
        and T3442.ZZ_A_ZDROJ_SYSTEM <> 'ZDROJ_C'
        and T19805.RADDOK_A_ZDROJ_SYSTEM <> 'PECE_EU'
        and T19805.RADDOK_TYP_DAVKY_INT_KOD <> 'F'
        and T17608.CAS_MESIC_KOD between '2020/11' and '2020/11'
      )
    group by
      T17686.CAS_ROK_KOD
    , concat(concat(T3442.ZZ_KOD, ' - '), T3442.ZZ_NAZEV)
  )
select
  D1.c1 as c1, D1.c2 as c2, D1.c3 as c3, D1.c4 as c4,
  D1.c5 as c5, D1.c6 as c6, D1.c7 as c7, D1.c8 as c8,
  D1.c9 as c9
from
  (
```

```
    select distinct
      0     as c1, D1.c6 as c2, D1.c7 as c3, D1.c8 as c4,
      D1.c5 as c5, D1.c4 as c6, D1.c3 as c7, D1.c2 as c8,
      D1.c1 as c9
    from
      SAWITH0 D1
    order by
      c4, c2, c3
)
D1;
```