Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Porovnání možností databází SQL a NoSQL pro využití ve webových aplikacích

Bakalářská práce

Autor: Ondřej Krkoška
Studijní obor: Aplikovaná informatika

Vedoucí práce: prof. RNDr. Petra Poulová, Ph.D.

Hradec Králové 2023

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 18.10.2023 Ondřej Krkoška

Poděkování:

Rád bych poděkoval vedoucí bakalářské práce prof. RNDr. Petře Poulové, Ph.D. za odborné vedení, za pomoc a rady při zpracování této práce.

Anotace

Tato písemná práce se zaměřuje na srovnání dvou hlavních kategorií databázových systémů (SQL a NoSQL) a na jejich využití v moderních webových aplikací. Úvodní část práce představuje základní druhy databází, jako jsou databáze klíč-hodnota, relační, dokumentové, sloupcové a grafové. Druhá část se zabývá konkrétními zástupci těchto kategorií, těmi jsou MySQL, Redis, HBase, MongoDb, Neo4j, a složitostí návrhu schématu pro každou z nich. Dále je v práci provedeno porovnání hardwarové náročnosti těchto databází na základě času potřebného pro vykonání základních operací jako jsou insert a update dat. V závěrečné části jsou vyhodnoceny výsledky a zhodnocena vhodnost využití jednotlivých typů databází ve webových aplikacích. Práce nabízí pohled na SQL a NoSQL databáze se snahou rozšířit čtenáři povědomí o různých typech databází a pomoci mu při výběru správného databázového systému pro konkrétní aplikaci.

Annotation

Title: Comparison of SQL and NoSQL database for web applications

This thesis focuses on the comparison of two main categories of database systems (SQL and NoSQL) and their use in modern web applications. The introductory part of the thesis presents the basic types of databases, such as key-value, relational, document, column and graph databases. The second part deals with specific representatives of these categories, namely MySQL, Redis, HBase, MongoDb, Neo4j, and the complexity of schema design for each of them. Furthermore, the thesis compares the hardware requirements of these databases based on the time required to perform basic operations such as insert and update data. In the final part, the results are evaluated and the appropriateness of using individual types of databases in web applications is evaluated. The work offers an overview of SQL and NoSQL databases with an effort to broaden the reader's awareness of different types of databases and help him choose the right database system for a specific application.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Většina lidí používá databáze v podstatě denně. Ať se jedná o sociální sítě jako Facebook, Skype nebo interní firemní systémy, vše využívá databáze. V podstatě lze říci, že databáze tvoří základ moderního světa. Existuje mnoho různých databázových systémů a tato práce se věnuje dvěma kategoriím, SQL a NoSQL.

První část této práce poskytuje úvod do světa databází. Zabývá se historií, vývojem a úvodem do jednotlivých typů databází. Představuje pět různých modelů databází. Databáze klíč-hodnota dále pak relační, dokumentové, sloupcové a grafové databáze. Na tyto typy databází je práce zaměřena.

Ve druhé části je demonstrována složitost návrhu za využití konkrétních zástupců, kterými jsou MySQL, Redis, HBase, MongoDb a Neo4j. Jsou zde vysvětleny základní principy fungování jednotlivých databází. Složitost návrhu je demonstrována na základě jednotného zadání a následného vytvoření databázového schématu.

Následující část je zaměřena na porovnání hardwarové náročnosti. Ta je měřena na základě času potřebného pro vykonání základních operací pro insert a update dat.

Poslední část hodnotí výsledky a posuzuje vhodnost využití jednotlivých typů databází.

2 Cíl a metodika práce

Práce si klade za cíl seznámit čtenáře s fungováním a základními principy různých typů databází. Představuje dvě hlavní kategorie databází, SQL a NoSQL. Popisuje výhody i nevýhody a snaží se poskytnout základní přehled o využití databází, a to na základě obecných teoretických informací, ale i na základě příkladů na konkrétních databázových systémech. Záměrem práce je pomoci při výběru vhodné technologie pro konkrétní projekt.

2.1 Výzkumné otázky

Otázka 1: Jsou NoSQL databáze vhodnější pro velké množství nestrukturovaných dat ve webových aplikacích?

Otázka 2: Jsou SQL databáze lepší pro webové aplikace s komplexními vztahy mezi daty?

Otázka 3: Jsou SQL databáze lepší pro webové aplikace, které musí splňovat přísné požadavky na konzistenci dat?

2.2 Metodika

Studium odborné literatury a článků bylo prvním krokem v tvorbě této práce. Toto studium mělo za cíl získat základní informace o databázích a seznámit se s dostupnými technologiemi. V této fázi byla provedena rešerše odborné literatury a článků. Byly vyhledány informace o různých typech databází, konkrétně relačních databází a NoSQL databází.

Následujícím krokem bylo zpracování teoretické části práce. V této části byl vytvořen přehled o jednotlivých druzích databází, včetně jejich výhod a nevýhod. Byly vysvětleny základní pojmy, jako jsou například entity, relace, atributy a dotazovací jazyky.

Dalším krokem bylo zpracování praktické části práce. V této části byla porovnána složitost návrhu jednotlivých databází na základě vytvoření schématu pro společné zadání. Byli vybráni konkrétní zástupci jednotlivých druhů databází, těmi jsou relační databáze MySQL a jako NoSQL byly vybrány databáze Redis, HBase,

MongoDB a Neo4j. Pro každou z těchto dat bylo vytvořeno schéma a následně byla porovnána složitost návrhu schématu.

Posledním krokem bylo testování rychlosti databází MySQL, Redis a MongoDB pomocí jazyka Java. Byla vytvořena jednoduchá aplikace, která prováděla cyklický zápis dat.

3 Historie databází

Potřeba zaznamenávat a uchovávat informace je dovednost nezbytná k rozvoji společnosti. Pro tyto účely se používala různá média jako například hliněné tabulky, pergamen nebo stále ještě používané kartotéky, které by se daly považovat za předchůdce databází. Nicméně možnosti těchto metod brzy dosáhly svého maxima a pro potřebu zaznamenání stále většího a rozšiřujícího množství dat se stali naprosto nedostačujícími, protože veškeré úkony byly limitovány rychlostí lidí. To se však změnilo v polovině 20. století, kdy přišla digitální revoluce, při které došlo k hojnému rozšíření počítačů.

3.1 Co je to databáze

Nejprve by bylo vhodné vysvětlit, co to vlastně databáze je. Databázi lze definovat jako soubor vzájemně propojených a logicky souvisejících uložených dat, se kterými lze manipulovat pomocí systému řízení báze dat (zkráceně DBMS). DBMS umožňuje práci s databází, dále pak dohlíží na zachování integrity dat a řídí přístupy jednotlivých uživatelů.

3.2 Vývoj

Jako první databázové modely se uvádějí hierarchický a síťový model. Tyto dva modely byly vyvíjeny souběžně.

Hierarchický model organizuje data do stromové struktury. Jednotlivé záznamy jsou mezi sebou propojeny ukazateli. Záznamy jsou pak odvozeny z jednoho rodičovského kromě tzv. kořenového záznamu, který je na vrcholu. Hierarchická struktura umožňuje pouze vztahy 1:1 a 1:N. Další nevýhodou je velice nízká pružnost v případě změny struktury. (HEAVY.AI, c2022)

Síťový model je velice podobný tomu hierarchickému ovšem s tím rozdílem, že umožňuje vztahy N:N. Nízká pružnost však omezuje i tento model, přesto se ale stal používanějším až do příchodu relačního modelu. (MARIADB, c2023)

Relační model je v současnosti nejrozšířenější a bude rozebrán detailněji v pozdější kapitole. Ve stručnosti si lze představit tuto databázi jako tabulky, ve

kterých jsou související data vzájemně propojena pomocí klíčů, což umožňuje využití vztahu N:N. Výhodou zde je velmi vysoká integrita dat. (ORACLE, c2023)

Další důležitou skupinou jsou **NoSQL** (Not only SQL) databáze. Přestože tento typ databází byl vyvinut již v začátcích digitální revoluce, pozornost jim začala být věnována teprve kolem roku 2000. Jejich užití řeší některé nedostatky relačních databází, a to zejména u nestrukturovaných a rozsáhlých dat. (MEIER, 2019)

NoSQL se dělí na několik typů, které budou rozebrány v následujících kapitolách.

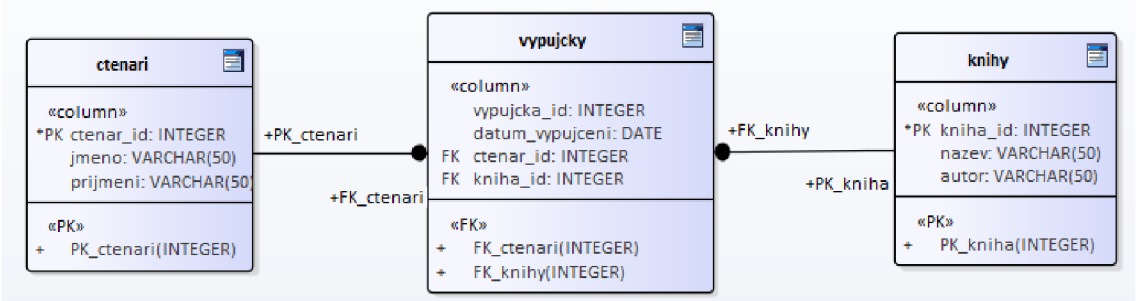
4 SQL databáze

SQL databáze, také nazývány relační databáze, využívají jazyk SQL. Jedná se o neprocedurální jazyk, to znamená, že definuje, co je potřeba provést, nikoliv jak je potřeba to provést. (Pokorný a Valenta, 2020, s. 67-69)

Pokorný a Valenta (2020, s. 67-69) dále uvádějí, že modelování databáze v SQL má následující rysy:

- Data jsou uložená v databázi ve formě tabulek
- SQL vrací data programu, který se nemusí starat o fyzickou strukturu či umístění dat
- SQL vrací data programu, který se nemusí starat o fyzickou strukturu či umístění dat
- Poloha tabulek v databázi není důležitá, jsou identifikovány jménem
- Pořadí sloupců v tabulkách není důležité, jsou identifikovány jménem
- Pořadí řádků v tabulkách není důležité, jsou identifikovány hodnotami ve sloupcích
- Data jsou prezentovány uživateli jako tabulky, bez ohledu na jejich vnitřní strukturu
- Tabulky a indexy mohou být v průběhu existence databáze definovány, modifikovány a opět rušeny

Dalším rysem SQL je možnost vytváření indexů, což jsou datové struktury umožňující rychlejší vykonání uživatelských požadavků. Indexy i tabulky mohou být vytvářeny, upravovány a rušeny dle požadavků uživatele. (Pokorný a Valenta, 2020, s. 67-69)



Obrázek 1: Příklad relační databáze (vlastní zpracování)

Na obrázku 1 je graficky znázorněna relační databáze o třech relacích (tabulkách). První relací jsou čtenáři. V této relaci jsou atributy *ctenar_id, jmeno* a *prijmeni*, přičemž *ctenar_id je* primárním klíčem. Atribut *ctenar_id* je také přiřazen jako cizí klíč relaci *vypujcky,* díky tomu lze propojit konkrétní záznam výpůjčky s konkrétním čtenářem. Relace *vypujčky* má dále další tři atributy, kterými jsou *vypujcka_id, datum_vypujceni* a *kniha_id*, kde *vypujcka_id* je primárním klíčem a *kniha_id* je dalším cizím klíčem odkazujícím na relaci *knihy*. Tedy z obrázku můžeme vyčíst základní fungování vztahů, které jsou tvořeny pomocí cizích klíčů. Primární klíče pak slouží pro jednoznačné identifikování záznamu. To znamená, že záznam relace *vypujcky* spojuje dva konkrétní záznamy ze zbylých relací a přidává jim další informační hodnotu díky vlastním atributům. Klíče jsou vytvářeny pomocí omezení, která nám zajišťují integritu. Na obrázku jsou omezení znázorněny ve spodní části relací.

4.1 Jazyk SQL

Jazyk SQL se dělí na několik částí. DDL (Data Definition Language), DML (Data Manipulation Language), DCL (Data Control Language) a TCL (Transaction Control Language).

Varma (2023) popisuje jednotlivé části jazyka SQL takto:

- **DDL** (Data Definition Language), slouží k vytváření a upravování databázové struktury pomocí příkazů CREATE, DROP, ALTER a TRUNCATE.
- **DML** slouží k vytváření, upravování, mazání a vyhledávání záznamů. Obsahuje příkazy INSER, DELETE, UPDATE A SELECT.
- **DCL** slouží k určování přístupových práv k tabulkám. K tomu využívá příkazy GRANT a REVOKE.
- **TCL** slouží ke správě transakcí za pomoci příkazů COMMIT, ROLLBACK a SAVE TRANSACTION.

4.2 Omezení

Omezení (constraints) zaručují zachování integrity dat. V případě pokusu o porušení některého z omezení je prováděný příkaz zamítnut a databáze je uchována v konzistentním stavu.

Omezení lze rozdělit do tří skupin:

- Entitní integrita
    - PRIMARY KEY
    - UNIQUE KEY
- Referenční integrita
    - FOREIGN KEY
- Doménová integrita
    - CHECK
    - DEFAULT

4.3 Entitní integrita

Pod tuto skupinu patří definice nenulových hodnot, primárních a unikátních klíčů. Omezení nenulových hodnot říká, že žádná hodnota v daném sloupci nesmí obsahovat hodnotu *null*. V případě omezení primárního klíče je nutné, aby sloupec nabýval jedinečných hodnot. Tento typ omezení může být v tabulce použit pouze jednou a zároveň nad daným sloupcem vytvoří index. Omezení na jedinečné hodnoty nařizuje, aby hodnoty ve sloupci nabývali jedinečných hodnot, na rozdíl od omezení primárního klíče není jeho použití omezeno.

4.4 Referenční integrita

Do této kategorie patří cizí klíče, tedy omezení zaručuje, že sloupec má hodnoty již existujících klíčů. Pokud je na nějaký záznam odkazováno cizím klíčem, není možné jej smazat dříve, než bude smazán cizí klíč.

4.5 Doménová integrita

Doménová integrita udává definiční obor všech přípustných hodnot pro daný atribut. Toho je docíleno podmínkou vytvořenou pomocí příkazu *CHECK.* Také sem patří příkaz *DEFAULT,* který slouží k nastavení defaultní hodnoty atributu.

4.6 Pohledy

Pohledy (views) jsou v podstatě virtuální tabulky založeny na výsledku SQL příkazu. Může se jednat o spojení atributů z vícera tabulek, nebo jen o zobrazení

vyfiltrovaných částí jedné tabulky. Díky pohledům je čtení v databázi jednodušší a intuitivnější. Další výhodou pohledů je, že neumožňují přístup k datům z originálních tabulek, ze kterých byly vytvořeny tedy lze je využít v případech, kdy uživateli nemají být udělena práva k manipulaci s daty. Také je možno na nich provádět dotazy jako na běžných tabulkách.

4.7 Spouštěče

Spouštěč (trigger) je uložený program, který čeká na konkrétní událost prováděnou nad konkrétní tabulkou. V případě spuštění pak provede definovanou akci. Například může být spuštěn při aktualizaci záznamu.

Dělí se na tři druhy:

- DML spouštěče – reagují na příkazy DML, těmi jsou INSERT, UPDATE a DELETE
- DDL spouštěče – jak již název napovídá, tyto spouštěče budou reagovat na příkazy DDL CREATE, ALTER a DROP
- Přístupové spouštěče – tento typ reaguje na události LOGON

(DRKUSIC, 2020)

4.8 Procedury

Procedura je stejně jako spouštěč uložený SQL kód. Na rozdíl od spouštěče však není volána automaticky při vyvolání určité události, ale je volána volitelně v případě potřeby.

Procedury mají několik výhod:

- Znovupoužití: proceduru může využívat vícero uživatelů či aplikací bez nutnosti opětovného zápisu kódu
- Jednoduchá úprava: proceduru je možné snadno upravit bez nutnosti restartování aplikace
- Snížení zatížení sítě: po síti je posílán pouze název procedury a případné parametry

4.9 Transakce

SQL databáze využívají ke svému zpracování transakce. Transakce je logická jednotka práce, která zachová databázi v konzistentním stavu. Každá transakce musí splňovat vlastnosti **ACID**. To znamená, že musí být atomická, konzistentní, izolovaná a trvalá. Atomicita (**a**tomicity) znamená, že transakce proběhne celá nebo vůbec, výsledkem nemůže být žádný mezistav. Konzistencí (**c**onsistency) je myšleno, že výsledek transakce nenaruší integritu databáze. V průběhu vykonávání transakce však může nastat dočasné narušení integrity. Izolovanost (**i**solation), tato vlastnost znamená, že se jednotlivé transakce nesmí vzájemně ovlivňovat. Trvalost (**d**urability) pak znamená, že výsledek transakce je uložen trvalým způsobem.

5 NoSQL

Na výraz NoSQL začal být kladen důraz v roce 2000, a to zejména kvůli rychlému rozvoji internetu a obrovskému množství dat generovaných různými webovými službami. Doposud nejrozšířenější relační databáze se začaly blížit v některých ohledech svým limitům a to z toho důvodu, že tyto databáze nejsou pouhé nástroje pro ukládání dat. Relační databáze využívají znatelnou část výpočetního výkonu pro zajištění datové integrity, indexování, uživatelských rolí atd. Tyto funkce mají bezesporu mnoho výhod, nicméně pro zpracování rozsáhlého množství dat, u kterých je kladen důraz hlavně na rychlost zpracování, nejsou pro svoji náročnost žádoucí. Dalším důvodem stojícím za rozvojem NoSQL databází je, že nemají předefinovanou strukturu a je tedy možné snadné zpracování nestrukturovaných dat. (MEIER, 2019)

NoSQL databáze mají následující vlastnosti:

- Předem nedefinovaný datový model
- Snadná škálovatelnost
- Transakce nevyužívají model ACID ale BASE

NoSQL má několik základních modelů:

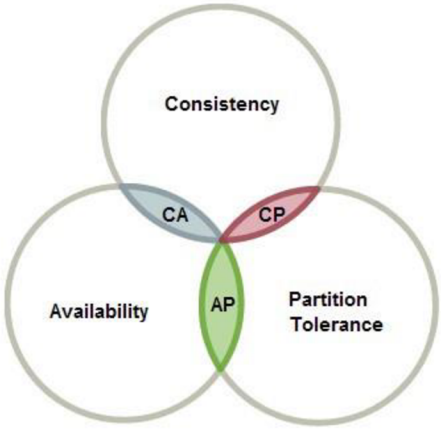
- Databáze klíč-hodnota
- Sloupcová databáze
- Dokumentová databáze
- Grafová databáze

5.1 CAP teorém

CAP teorém popisuje tři základní vlastnosti NoSQL databází, zároveň však říká, že lze v jeden okamžik dosáhnout pouze dvou.

- **Konzistence (Consistenci)** - Dotaz vrátí hodnotu posledního zápisu nebo chybové hlášení. (MEIER, 2019)

- **Dostupnost (Availability)** - Každý dotaz vrátí nechybovou odpověď, i v případě, že některé uzly nefungují, ovšem bez záruky vrácení nejnovějšího zápisu. (MEIER, 2019)
- **Tolerance oddílů (Partition Tolerance)** - Selhání jednotlivých uzlů nebo spojení mezi uzly v replikované počítačové síti nemají vliv na systém jako celek a uzly lze kdykoli přidat nebo odebrat, aniž by bylo nutné zastavit provoz. (MEIER, 2019)



Obrázek 2: CAP teorém. Zdroj: (NAZRUL, 2018)

5.2 Databáze klíč hodnota

V této databázi jsou uloženy vždy pouze dvojice klíč, který představuje jedinečný identifikátor a k němu přiřazená hodnota. Hodnotou může být libovolný objekt, nebo i primitivní datový typ. Hodnota tedy může být například řetězec znaků, pole, číslo, datum, XML atd. Klíč lze strukturovat pomocí speciálních znaků, jako jsou lomítka, dvojtečky apod. To umožňuje definici jmenného prostoru, který může představovat základní struktura dat. Kromě tohoto nelze využít žádnou strukturu ani odkazy. Databáze klíč-hodnota také nemají žádný dotazovaný jazyk, to znamená, že hodnoty nelze dotazovat ani prohledávat. Dotazovat se lze pouze na hodnotu klíče. (MEIER, 2019)

Příklad záznamu uživatele:

- uzivatel:U1234:jmeno David
- uzivatel:U1234:prijmeni Novák

- uzivatel:U1234:email david.novak@gmail.com
- uzivatel:U1234:heslo D75872C

Výhody key-value databáze:

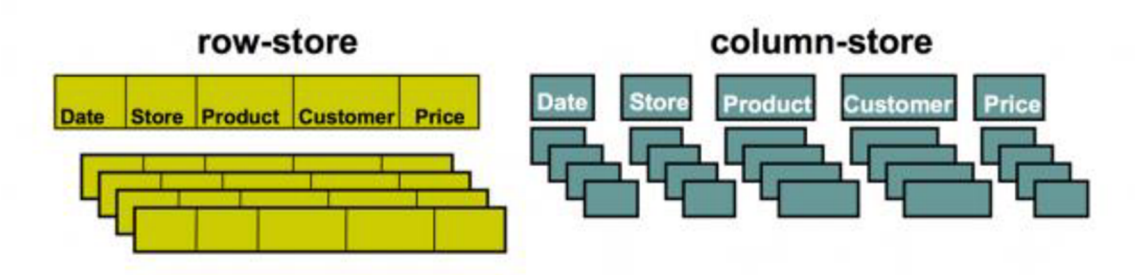
- **Jednoduchost** – využití je poměrně přímočaré. Jednoduché příkazy a absence datových typů zajišťuje snadné použití
- **Rychlost** – databáze klíč hodnota indexuje klíče, lze tak rychle vyhledat jakýkoliv záznam bez ohledu na velikost databáze
- **Škálovatelnost** – snadná horizontální i vertikální škálovatelnost

5.3 Sloupcové databáze

Sloupcové databáze na rozdíl od relačních ukládají data po sloupcích. To se ukázalo být efektivnější pro optimalizaci dotazovacích a agregačních funkcí. To je dáno tím, že informace v jednom řádku jsou zřídka kdy potřeba najednou a je tedy zbytečné načítat celý řádek. V mnoha případech je tedy vhodné strukturovat databázi po sloupcích, a to zejména v případech, kdy je potřeba přeskočit nerelevantní data v celém záznamu nebo je potřeba hojně využívat agregačních funkcí.

Sloupcové databáze excelují v:

- Dotazy, které zahrnují pouze několik sloupců
- Agregační dotazy na obrovské množství dat
- Sloupcová komprese



Obrázek 3: Porovnání sloupcové a relační databáze. Zdroj: (MPP & Columnar Databases, 2021)

5.4 Dokumentová databáze

Tato databáze kombinuje absenci schématu s možností strukturování uložených dat. Na rozdíl od toho, co vyplývá z názvu, úložiště dokumentů neukládá libovolné

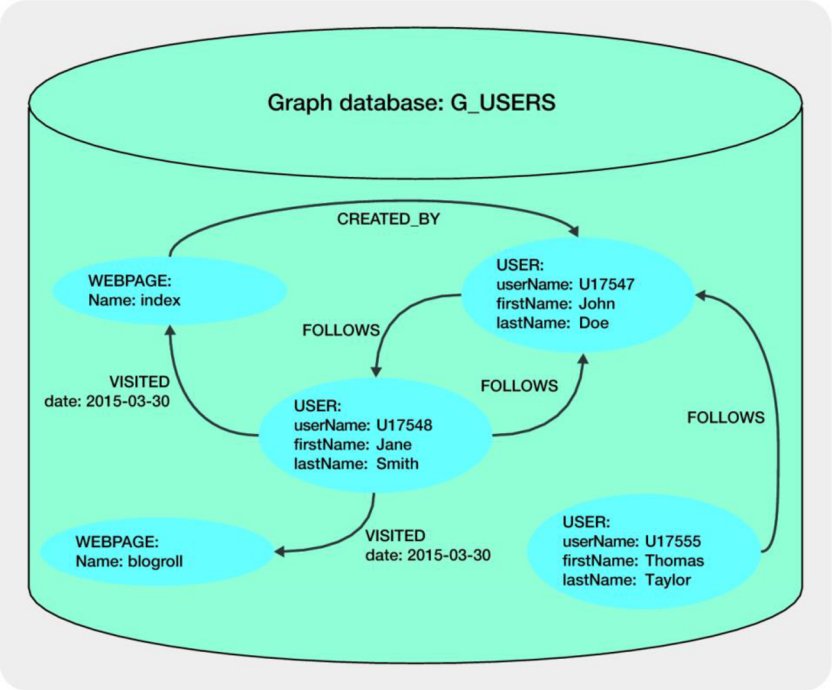
dokumenty, jako jsou videa nebo zvuková data, ale strukturovaná data v záznamech, které se nazývají dokumenty. Toto řešení bylo vyvinuto zejména pro použití ve webových službách. Lze je proto snadno integrovat s webovými technologiemi, jako je např. JavaScript. Důraz je zde kladen především na zpracování velkého množství různorodých dat, což se týká velkého množství webových dat, kde není třeba zajišťovat stálou konzistenci dat. Výjimkou jsou webové služby citlivé na zabezpečení, jako je online bankovnictví, které silně spoléhají na omezení schémat a zaručenou konzistenci. Dokumentová databáze, jak již bylo zmíněno, je zcela bez schémat, tj. není třeba definovat schéma před vložením datových struktur. Odpovědnost za schéma je tedy přenesena na uživatele nebo zpracovatelskou aplikaci. Nevýhodou vyplývající z neexistence schématu je chybějící referenční integrita. (MEIER, 2019)

Vlastnosti dokumentové databáze:

- Je to úložiště párů klíč-hodnota, kde klíč je id dokumentu a hodnota je zde dokument samotný.
- Dokumenty obsahují datové struktury ve formě rekurzivně vnořených párů atribut-hodnota bez referenční integrity.
- Tyto datové struktury jsou bez schémat, tj. v každém dokumentu lze použít libovolné atributy, aniž by bylo nutné nejprve definovat schéma.

5.5 Grafová databáze

Grafové databáze se používají, když jsou data organizována v sítích. V těchto případech nezáleží na jednotlivých záznamech, ale na propojení všech záznamů mezi sebou, například na sociálních sítích, ale také při analýze infrastrukturních sítí (např. vodovodní sítě nebo elektrické sítě), v internetovém směrování, nebo při analýze odkazů mezi weby. Záznamy se skládají z uzlů (objektů) a z orientovaných hran (vztahů). Uzly i hrany jsou označeny a mohou mít vlastnosti. Vlastnosti jsou uvedeny jako dvojice atribut-hodnota podle vzoru (atribut : hodnota) s názvy atributů a příslušnými hodnotami.



Obrázek 4: Příklad grafové databáze s uživatelskými daty. Zdroj: (MEIER, 2019)

Na obr. 5 Je znázorněna grafová databáze G_USERS, která představuje informace o webovém portálu s uživateli, webovými stránkami a vztahy mezi nimi. Lze zde pozorovat dva typy uzlů, USER a WEBPAGE. Dále jsou zde tři typy hran, FOLLOWS, VISITED a CREATED_BY. Typ uzlu USER má atributy *userName*, *firstName* a *lastName*. Typ uzlu WEBPAGE má pouze atribut Name, typ hrany VISITED má atribut *date*.

Databáze grafů má následující vlastnosti:

- Data jsou organizována jako grafům podobné struktury
- Manipulace s daty jsou vyjádřeny jako grafové transformace nebo operace, které přímo řeší typické vlastnosti grafů (např. cesty, sousedství, podgrafy, souvislosti atd.).
- Databáze podporuje kontrolu integritních omezení pro zajištění konzistence dat.

6 Složitost návrhu a implementace

Následující část se bude zabývat složitostí návrhu a implementace. Pro demonstrativní účely bude využito stejné zadání, které bude převedeno do zástupců jednotlivých typů DBMS. V případě implementace se předpokládá pouze jeden databázový uzel, není zde tedy popsána distribuce, ve které NoSQL databáze excelují oproti relačním.

Zadání je zasazeno do zjednodušeného prostření e-shopu, kde bude potřeba evidovat informace o zákazníkovy a o objednávkách. Tzn. pro každý typ databáze bude vytvořeno schéma, které bude uchovávat informace o zákazníkovi (jméno, příjmení, adresa apod.), objednávkách (položky, datum vytvoření, stav atd.) a vztazích mezi objednávkami a zákazníky tak, aby bylo možné dané schéma využít v reálných podmínkách.

6.1 SQL databáze

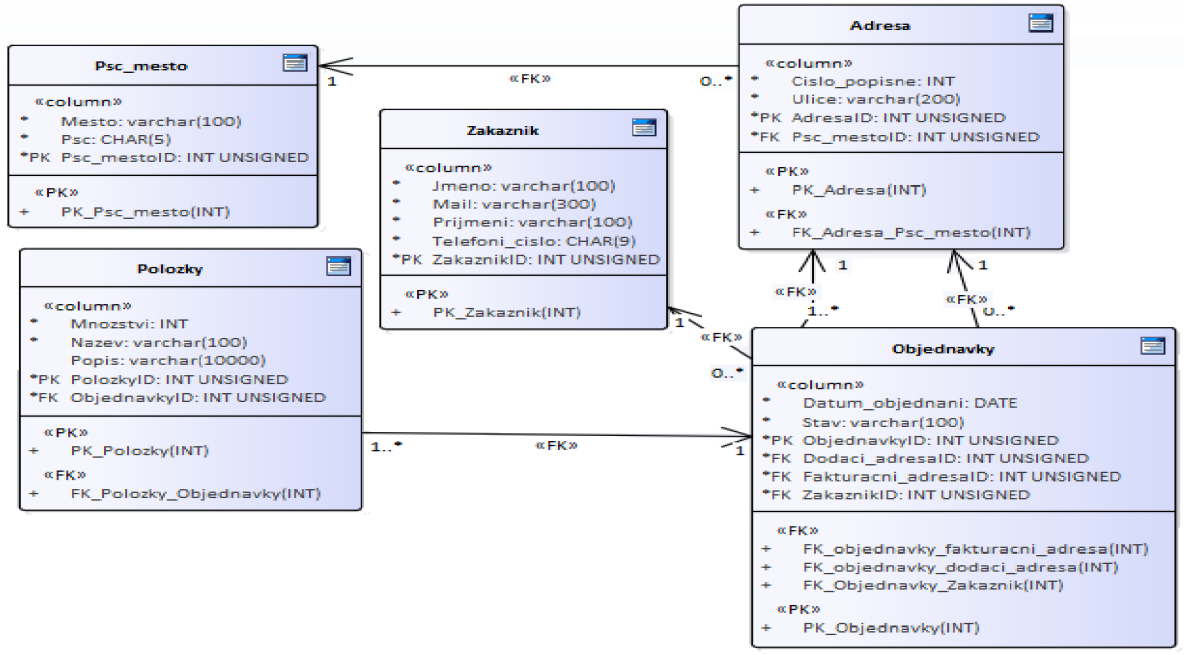
Jako zástupce SQL databází bude využit MySQL DBMS, a to z několika důvodů. Je k dispozici jako open source, tedy program šířený zdarma. Dále má solidní výkon u méně rozsáhlých databází (nevhodný pro rozsáhlé a složité projekty). Také je multiplatformní, snadno se učí a používá.

6.1.1 Návrh

Pro prvotní návrh schématu pro SQL databáze je nejjednodušším způsobem využití grafického editoru, ten umožnuje návrh relací, klíčů, indexů i některých integritních omezení, v tomto případě je využit Enterprise Architect. Návrh je poměrně intuitivní, je však důležité řídit se několika základními pravidly, která jsou označována jako normální formy.

Normální formy:

1. Všechny hodnoty jsou atomické
2. Data závisí na celém klíči
3. Neklíčová data jsou závislá pouze na klíči, nikoliv mezi sebou
4. Složený klíč nesmí být tvořen nezávislými atributy



Obrázek 5: Grafický návrh schématu SQL databáze. Zdroj: (vlastní zpracování)

Obr. 6 ukazuje příklad toho, jak může vypadat graficky řešený návrh. V tomto případě se jedná o řešení výše zmíněného zadání. Tedy o evidenci základních údajů o zákaznícovi, objednávkách a dalších souvisejících relací, tj. položky objednávky, adresa a města s poštovním směrovacím číslem. Z takto navrženého schématu pak lze vygenerovat DDL kód, který vytvoří schéma databáze.

6.1.2 Implementace

Nejprve je nutné nainstalovat samotný MySQL server, v tomto případě MySQL community edition, který je dostupný pod GPL licencí. Ten je dostupný v podobě instalátoru z oficiálních stránek (mysql.com). Po jeho spuštění se otevře standardní instalační okno. Zde lze navolit vše potřebné pro instalaci. V tomto případě to bude MySQL server 8.0.32 – X64 a klient MySQL Workbanch, který zajištuje grafické prostředí pro práci s relačními databázemi. Dále lze konfigurovat porty, uživatele apod.

Po dokončení instalace je možné se pomocí klienta k serveru připojit, vytvořit nové databázové schéma a použít vygenerované DDL příkazy. Tím jsou definovány relace společně s jejich vztahy.

6.1.3 Práce s daty

MySQL ukládá každou databázi jako podadresář do svého datového adresáře nastaveného v *my.ini* popřípadě *my.cnf* v proměnné datadir. Když vytvoříte relaci, MySQL uloží její definici do souboru nesoucí stejný název jako relace. Jelikož MySQL využívá souborový systém k ukládání názvů databází a definic tabulek, závisí rozlišování malých a velkých písmen na využité platformě. V případě využití operačního systému Windows se malá a velká písmena nerozlišují, kdežto u systému vycházejících z Unixu rozlišována jsou. Konkrétní zpracování a uložení dat a indexů závisí také na použitém enginu.

MySQL je ve výchozím nastavení orientováno na řádky, což znamená, že data každého řádku jsou uložena společně a DBMS při provádění dotazů pracuje v jednotkách řádků. Pro efektivní vyhledávání používá MySQL standardně B-strom nebo hashovací tabulky. Uživatel pak může s daty pracovat za pomoci DML jazyka. (SCHWARTZ, 2012)

6.1.3.1 Datové typy

SQL databáze podporují velkou škálu datových typů. Níže jsou zdůrazněny pouze ty nejčastěji využívané.

Varchar, řetězec s proměnnou délkou o rozsahu od 0 do 65 535 znaků. Lze nastavit maximální délku. V případě zadání kratšího řetězce se znaky nedoplňují.

Char, znakový řetězec o rozsahu 0 až 255znaků s pevnou délkou. V případě zadání kratšího řetězce oproti nastavené délce budou na místo chybějících znaků doplněny mezery.

Int, celočíselná hodnota s rozsah od -2 147 483 648 do +2 147 483 647 (unsigned Int od 0 do 4 294 967 295).

Float, datový typ s plovoucí řádovou čárkou, rozsahem od -3.402823466E+38 do +3.402823466E+38 a přesností na sedm desetinných míst.

Double, datový typ s plovoucí řádovou čárkou, rozsahem od -1.7976931348623157E+308 do +1.7976931348623157E+308 a přesností na patnáct desetinných míst.

Date, datum ve formátu „RRRR-MM-DD“ a to v rozsahu od 1000-01-01 do 9999-12-31.

DateTime, datum a čas ve formátu „RRRR-MM-DD HH:MM:SS“ s rozsahem od 1000-01-01 00:00:00 do 9999-12-31 23:59:59.

6.1.4 Využití v konkrétní aplikaci

GitHub používá MySQL jako hlavní datové úložiště pro všechny věci, které nejsou z gitového zdroje a jejichž dostupnost je pro fungování GitHubu zásadní.
Provozuje několik clusterů MySQL, které slouží různým službám a úkolům. Clustery používají klasické nastavení primárních replik, kde jeden uzel v clusteru (primární) je schopen přijímat zápisy. Zbytek uzlů clusteru (repliky) asynchronně přehrává změny z primárního a obsluhuje čtený provoz. (NOACH, 2018)

6.2 Databáze klíč-hodnota

Zástupcem pro tento typ databází je Redis. Jedná se o široce rozšířený open source DBMS. Mezi hlavní výhody Redisu patří výkon, všechna data jsou uložena v paměti, což umožňuje nízkou latenci a rychlý přístup k datům. Dále pak využívá flexibilních datových struktur a umožňuje replikaci na vícero serverů.

6.2.1 Perzistence

Jak již bylo zmíněno, Redis uchovává data v paměti, nicméně lze nastavit ukládání dat i na pevný disk. K tomuto slouží RDB a AOF, popřípadě kombinace obojího.

6.2.1.1RDB

RDB spočívá v uložení obrazu celé databáze v definovaných intervalech, který lze podmínit počtem změn klíčů. Nastavení RDB zálohování lze upravit za běhu databáze bez nutnosti restartu. Obraz databáze se ukládá jako soubor s příponou

.rdb. Jedná se tedy o vhodný způsob zálohování databáze. Nevýhodou je, že může dojít ke ztrátě dat zapsaných po posledním zálohování. (Redis, c2023)

6.2.1.2AOF

Ve chvíli, kdy je AOF zapnuté Redis připojí veškeré obdržené příkazy k AOF souboru. V případě restartování databáze, Redis provede veškeré příkazy znovu. Jelikož AOF soubor s každým dalším zápisem roste, je třeba při překročení velikosti vytvořit nový s komprimovanou sadou příkazů. Například se může stát, že hodnota jednoho klíče bude stokrát inkrementována tzn., že bude do AOF souboru přidáno 100 zápisů, které ovšem nejsou potřeba k obnově aktuálního stavu, a proto budou v novém souboru komprimovány. Přepis do nového souboru je zcela bezpečný. (Redis, c2023)

Často je používán samotný AOF, ale nedoporučuje se to, protože mít pravidelné zálohy RDB zvyšuje bezpečnost a umožňuje rychlejší restarty v případě chyb v enginu AOF.

6.2.2 Návrh

Jak již bylo zmíněno, databáze klíč-hodnota nevyužívá žádné předem definované schéma. Pro základní strukturu dat lze využít speciálních znaků, kterými lze strukturovat klíč, který bude dotazován. Toto je jediný způsob, jak lze definovat jmenný prostor, o ostatní se musí postarat aplikační logika. Samotná hodnota pak může nabývat jednoduchých hodnot ale i rozsáhlých řetězců, není proto nutné tvořit klíč pro každý atribut zvlášť, naopak je vhodné pod klíč ukládat serializované objekty, které mohou být ve většině moderních programovacích jazyků deserializovány. V případě Redisu lze namísto serializace využít pokročilých datových struktur, jako je například datový typ Hash.
Řešení e-shopu by mohlo být řešeno například takto:
Pro evidenci zákazníka použijeme datový typ Hash, jehož klíč je v podobě „customer:[id]“

```
HMSET customer:[id] name [jméno] surname [příjmení] mail [mail] tel [tel]
```

Jelikož se lze dotazovat pouze na klíče, je vhodné vytvořit Hash pro dohledání indexu zákazníka. Zde je to řešeno pomocí mailu.

```
HSET customers [customer_mail] [customer_id]
```

Adresa zákazníka bude opět typu Hash, pro spojení zákazníka a adresy využijeme id zákazníka, které přidáme do klíče adresy.

```
HMSET adress:[customer_id] city [město] PostalCode [PSČ] numberOfDescriptive [čp.] street [ulice]
```

Každá objednávka bude vytvořena samostatně a hodnota bude serializovaná instance objednávky.

```
SET order:[id] [objednávka]
```

Přiřazení objednávek k uživateli lze provést pomocí datového typu List. V tomto případě je každá nová objednávka přiřazena na začátek.

```
LPUSH customer:[customer_id]:orders [order_id]
```

Pro zpracování objednávek lze opět použít datový typ List, zde se ale budou objednávky přidávat nakonec seznamu, tak aby se nejprve zpracovali objednávky, které přišli dříve.

```
RPUSH orders [order_id]
```

Smazání prvního výskytu objednávky se zadaným id po jejím zpracování z listu „orders“ lze provést takto.

```
LREM orders 1 [id]
```

6.2.3 Implementace

Redis je primárně určen pro linuxovou distribuci, v případě systému Windows je tedy nutné nejprve nainstalovat Windows Subsystem for Linux (WSL2). Samotná instalace Redisu využívá APT repositář. Pro instalaci tedy stačí přidat repositář Redisu do APT indexu, updatovat APT a nakonec instalovat samotný Redis. Podrobný návod lze nalézt na oficiálních stránkách (redis.io). Změna konfigurace se provádí úpravou souboru redis.conf.

6.2.4 Práce s daty

Redis podporuje datové typy list, Set, Hash a Sorted Set. Všechny tyto typy ukládají hodnoty v podobě řetězců, tzn. String. Také zde existují atomické operace, které jsou bezpečné i při vícenásobných přístupech ke stejnému klíči.
Důležité je zmínit, že k datům můžeme přistupovat pouze přímým dotazem na klíč. Není možné vrátit klíč, který obsahuje konkrétní hodnotu.

6.2.5 Datové typy

String obsahuje libovolné řetězce znaků. Lze tedy použít pro mnoho věcí, jako je uložení binárních dat, serializovaných objektů apod. Hodnota však nemůže být větší než 512 MB.

List, jedná se o seznam hodnot, který je v Redisu řešený jako Linked List. To znamená, že novou hodnotu lze přidat pouze nakonec nebo na začátek seznamu. Výhodou tohoto řešení je rychlost zápisu dalších hodnot, která je nezávislá na celkovém počtu hodnot v seznamu. Nevýhodou je pomalý přístup k hodnotám nacházejících se daleko od začátku či konce.

Set, datový typ Set je kolekce neseřazených hodnot. Každá hodnota zde může být obsažena pouze jednou. Umožňuje operace jako např. průnik sad, testování existence hodnoty apod.

Sorted Set, podobně jako Set ani tento datový typ neumožňuje duplicitní uložení hodnot. Rozdíl je v tom, že ke každé hodnotě ukládá i tzv. skóre, tj. číselná hodnota označující pořadí. Stejné skóre se může vyskytnout u vícera hodnot, v tom případě jsou porovnány jako řetězce. Je možné se dotazovat na prvky podle rozsahu skóre, lexikograficky, v opačném pořadí apod.

Hash, tato datová struktura je v podstatě asociativní pole. Hash je vhodný pro reprezentaci objektů.

6.2.6 Využití v konkrétní aplikaci

StackExchange využíva Redis jako cachovací vrstvu pro celou síť. Do mezipaměti jsou ukládány prakticky všechny stránky, ke kterým přistupují anonymní uživatelé (a které jsou následně zobrazovány). (MONTROSE, 2016)

Montrose (2016) dále uvádí tyto statistiky:

- V každém okamžiku obsahuje mezipaměť Redis přibližně 1 300 000 klíčů
- Většina z těchto záznamů vyprší v řádu několika minut
- V době největšího zatížení Redis vyřizuje několik stovek čtení/zápisů za sekundu

6.3 Sloupcová databáze

HBase, distribuovaná databáze šířena jako open source software, byla vytvořena podle Bigtable od společnosti Google a napsána v jazyce Java. HBase je velmi efektivní pro práci s velkými a řídkými datovými sadami.

HBase vypadá podobně jako relační databáze, nicméně na rozdíl od relačních databází je orientována na sloupce a umožňuje práci s polostrukturovanými daty. Data tedy seskupuje po sloupcích, případně sloupcových rodinách. O samotný zápis dat se stará HDFS (Hadoop Distributed File System). Data jsou uložena jako pole bajtů, uložit lze tedy cokoliv, co lze na takovéto pole konvertovat. S daty pak lze manipulovat pomocí funkcí MapReduce Hadoop. Uložená data jsou neměnná a proto, pokud je změněna hodnota v záznamu nedojde k přepsání, nýbrž k vytvoření nové verze záznamu.

6.3.1 Komponenty datového modelu

Tabulka, v HBase je tabulka sloupcově orientovaná, tzn. data jsou uložena ve sloupcovém formátu.

Klíč řádku, podobně jako u relačních databází i zde se jedná o jedinečné id řádku. Usnadňuje a urychluje vyhledávání.

Rodina sloupců, logicky dělí tabulku a sdružuje množinu sloupců, které jsou uloženy společně.

Sloupec, každý sloupec nebo též kvalifikátor sloupce. Není předem definován, je vytvořen až při zápisu záznamu. Každý záznam má stejné rodiny sloupců, ale nemusí mít stejný počet sloupců.

Buňka, skládá se z klíče řádku, rodiny sloupců, sloupce a ukládá data. Buňka je neměnná, lze však vytvořit novou verzi, která je označena časovým razítkem.

Časové razítko, skládá se z data a času. Při každém zápisu je buňka označena datem a časem, kdy byl zápis proveden. Časové razítko usnadňuje orientaci mezi jednotlivými verzemi dat.

6.3.2 Návrh

Podobně jako u relačních databází i zde se vytvářejí tabulky, které seskupují data. Evidence zákazníků a jejich objednávek je řešitelná jednou tabulkou se třemi rodinami sloupců. Konkrétně s rodinami personal_info, adress a order.

customer											
row key	personal_info				adress				order		
	name	surname	mail	tel	city	psc	street	desc_num	{order1}	...	{orderX}

Tabulka 1:Struktura tabulky v HBase. Zdroj: (vlastní zpracování)

Tabulka výše zobrazuje příklad toho, jak by mohla vypadat základní struktura. Modré pole označuje název tabulky (customers), šedá pole představují sloupcové rodiny (personal_info, order) a bílá pole jsou potencionální sloupce. Potencionální proto, že nejsou předem definovány a v záznamu nemusí být uplatněny. Ve sloupcové rodině *orders* bude tolik sloupců, kolik je objednávek a bude v nich uložený celý objekt. V případě potřeby vykonávat nad objednávkami různé dotazy např. pro statistické účely by toto řešení nebylo vhodné. Vhodnější by bylo pro objednávky vytvořit novou tabulku. Zde je však snaha demonstrovat nestrukturovanost sloupcových rodin.

Vytvoření tabulky a záznamu pak může vypadat takto:

```
create ‘customer‘,‘personal_info‘, ‘order‘
```

```
put ‘customer‘ [‘id‘] ‘personal_info:name‘, [‘jméno‘]
put ‘customer‘ [‘id‘] ‘personal_info:second_name‘, [‘příjmení‘]
put ‘customer‘ [‘id‘] ‘personal_info:mail‘, [‘mail‘]
put ‘customer‘ [‘id‘] ‘personal_info:tel‘, [‘tel‘]
put ‘customer‘ [‘id‘] ‘adress:city‘, [‘město‘]
put ‘customer‘ [‘id‘] ‘adres:postal_code‘, [‘PSC‘]
 put ‘customer‘ [‘id‘] ‘adress:street‘, [‘ulice‘]
put ‘customer‘ [‘id‘] ‘adress:desc_num‘, [‘číslo popisné‘]
put ‘customer‘ [‘id‘] ‘order:order1 ‘, [‘{order} ‘]
```

6.3.3 Práce s daty

Práce s daty probíhá skrze *HBase Shell*, který lze spustit z příkazového řádku v adresáři HBase pomocí příkazu bin/hbase shell. V základu lze k dotazování využít pouze příkazy *scan* a *get*. Ani jeden z techto příkazů však neumožňují komplexní dotazy, pro ty je nutné využít například Apache Drill. Tento nástroj podporuje distribuované datově náročné databáze, a umožňuje nad nimi volat SQL.

6.3.4 Implementace

Předpokladem pro implementaci je nainstalovaný JDK, to stačí pro spuštění HBase v režimu standalone, tedy spuštění pouze jedné instance. Pro distribuovanou HBase je potřeba také Apache HADOOP, který obsahuje Hadoop Distributed File System. Poté je nutné stáhnout a extrahovat HBase (hbase.apache.org). Následuje spuštění databázového serveru. Ten lze spustit skrze příkazový řádek přejitím do extrahované složky a zadáním příkazu bin/start-hbase.sh. Dále je nutné spustit HBase shell příkazem bin/hbase shell v něm je poté možné manipulovat s daty.

6.3.5 Využití v konkrétní aplikaci

Shina (2016) uvádí, že Facebook původně využíval MySQL, nicméně to nedokázalo efektivně zpracovávat velké datové sady, indexy. Data neustále rostla a výkon klesal.

Dle Shina (2016) Facebook objevil dva obecné vzorce dat:

- Dočasná data s nestálou strukturou
- Stále rostoucí sada dat, která jsou zřídka kdy dotazována

Dále Shina (2016) představuje hlavní požadavky:

- Ukládání velkých objemů neustále rostoucích dat z různých služeb Facebooku.
- Vysoký výkon potřebný pro obsluhu milionů požadavků.
- Zachování konzistence při skladování a výkonu.

Pro všechny tyto problémy přišel Facebook s řešením, kterým je HBase. Facebook přijal HBase pro obsluhu Facebook messengeru, chatu atd. kvůli jeho různým funkcím. HDFS, což je základní souborový systém používaný HBase, jim také poskytl několik potřebných funkcí, jako jsou komplexní kontrolní součty, replikace a automatické vyvažování zátěže. (SINHA, 2016)

6.4 Dokumentová databáze

MongoDB community edition, multiplatformní open source databáze používající dokumenty uložené ve formátu BSON, což je binární forma JSON. Záznamy jsou zde ukládány do kolekcí. Kolekci si lze představit jako tabulku bez vnitřní struktury, je však dobré používat alespoň podobnou strukturu záznamů pro efektivní indexaci. Každý dokument je složen z vnořených párů klíč-hodnota. Data jsou pak seskupena na disku podle kolekcí. Pro každou kolekci jsou standardně dva soubory, jeden pro meta data kolekce a druhý pro samotná data. Souborů však může být i více v závislosti na velikosti kolekce.

6.4.1 Návrh

Evidenci zákazníků a objednávek lze řešit pomocí dvou kolekcí. První kolekce custommers bude obsahovat informace o zákaznících a druhá orders o objednávkách. Do těchto kolekcí budou zapisovány záznamy, jejichž strukturu řeší aplikační logika. Příklad toho, jak by to mohlo vypadat je níže.

Vytvoření kolekcí:

```
Db.createCollection(„customers“)
Db.createCollection(„orders“)
```

Přidání záznamu objednávky:

```
db.orders.insert(
        {
         id: 1259,
        state: 'done',
        dateOfAcceptance: 2023-03-14T18:40:31.631Z,
         items: [
                { name: 'item1', count: 3, priceForUnit: 159 },
                { name: 'item2', count: 1, priceForUnit: 852 }
                ]
        }
)
```

Přidání záznamu zákazníka:

```
db.customers.insert(
        {
        id: 123,
        name: 'Robert',
        secondName: 'Pacovsky',
         contacts:{
        tel: 702159365,
        mail: pacovsky@gmail.com
        }
        adress: {
                city: 'Hradec Králové',
                postalCode: '500 02',
                street: 'Čs. odboje',
                numberOfDesc: '123'
                 },
        orders: [ '1259', ... ]
        }
)
```

6.4.2 Implementace

Nejsnazším způsobem je využít instalátor, kde lze navolit základní nastavení. Pro další práci s MongoDB je dobré využít Compass, to je grafické rozhraní, které umožňuje dotazování, optimalizaci, analýzu dat a správu jednotlivých databází v MongoDB.

6.4.3 Práce s daty

V kolekci se lze na dokumenty dotazovat pomocí metod find(), pretty() a findOne. Metoda find() vrátí veškeré dokumenty, které se shodují se zadanými kritérii v neformátované podobě, pro zobrazení ve formátované podobě lze použít metoda pretty(). Metoda findOne() pak vrací pouze první nalezený dokument, který se shoduje s kritérii.

Pro vyhledávání lze využít tyto operátory:

- Rovno {<key>:{$eg;<value>}
- Méně než {<key>:{$lt:<value>}}
- Méně nebo rovno {<key>:{$lte:<value>}}
- Větší než {<key>:{$gt:<value>}}
- Větší nebo rovno {<key>:{$gte:<value>}}
- Nerovná se {<key>:{$ne:<value>}}

Příklad:

```
db.orders.pretty({"dateOfAcceptance":{$lte:“2023-03-14T18:40:31.631Z
“}}).pretty()
```

Tento příkaz vyhledá všechny dokumenty v kolekci orders, které byly vytvořeny 2023-03-14T18:40:31.631Z nebo dříve.

6.4.4 Datové typy

MongoDB umožňuje využití jakéhokoliv datového typu za předpokladu, že lze datový typ převést na formát JSON. Konkrétně lze tedy využít string, integer, boolean, double, array, timeStamp, object, null, date, binary data ad.

6.4.5 Využití v konkrétní aplikaci

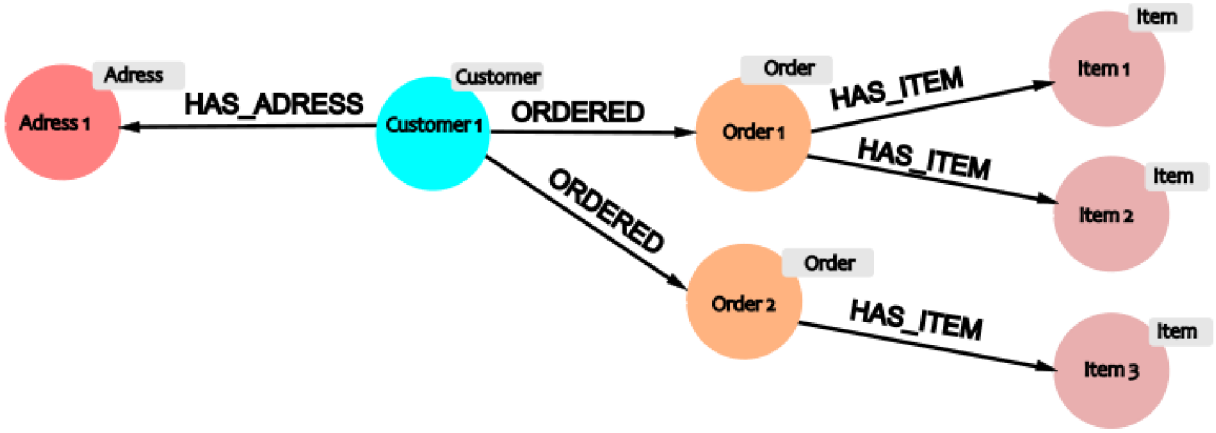
„Shutterfly je populární platforma pro sdílení fotografií online a používá MongoDB ke správě a ukládání více než 6 miliard obrázků, které mají rychlost transakcí až 10 000 operací za sekundu. Shutterfly dříve používal Oracle, ale později přešel na MongoDB a to pro to, že přechod na nerelační databázi jim pomohl zvýšit jejich škálovatelnost, výkon a produktivitu. Společnost se rozhodla pro přístup k databázi dokumentů z důvodu velkého objemu dat.

*Tento přechod měl dva hlavní důvody – MongoDB umožňuje flexibilitu se schématem a usnadňuje horizontální škálovatelnost. Shutterfly používá MongoDB Atlas jako svou databázovou aplikaci. Podnik provozuje několik clusterů Atlas a efektivně spravuje tisíce transakcí každou minutu.“ (*Vlastní překlad); (upGrad, 2022)

6.5 Grafová databáze

Neo4J je transakční systém pro správu grafových databází podporující ACID vlastnosti, který je implementován v jazyce Java. Je k dispozici v komunitní edici pod GNU licencí.

6.5.1 Návrh



Obrázek 6: Reprezentace zákazníka a jeho objednávek v grafové databázi

Obrázek výše popisuje strukturu grafové databáze pro řešení evidence zákazníků a objednávek. Zobrazuje uzly a jejich vztahy (hrany). Lze zde naleznout uzly označené štítky „Adress“, „Customer“, „Order“ a „Item“. Šipky znázorňují směr hran, říkají tedy, že „Customer 1“ „HAS_ADRESS“ „Adres 1“ atd.

Příklad toho, jak může vypadat zápis zákazníka, jeho adresy, objednávky, která obsahuje jednu položku a vztahů (hran) všech uzlů:

```
CREATE (customer1:Customer { name: 'Alfréd', secondName: 'Novák', tel: '702 654 366', mail: 'novak@gmail.com'})
CREATE (adress1:Adress {city: 'Hradec Králové', postalCode: '504 01', street: 'Šimkova', numberOfDesc: '123'})
CREATE (customer1)-[:HAS_ADRESS]->(adress1)
CREATE (order1:Order {id: '159357', dateOfAcceptance: '2023-03-14T18:40:31.631Z ', state: 'accepted'})
CREATE (customer1)-[:ORDERED]->(order1)
CREATE (item1:Item {itemName: 'notebook', count: '1', costForUnit: '10000'})
CREATE (order1)-[:ORDERED]->(item1)
```

6.5.2 Práce s daty

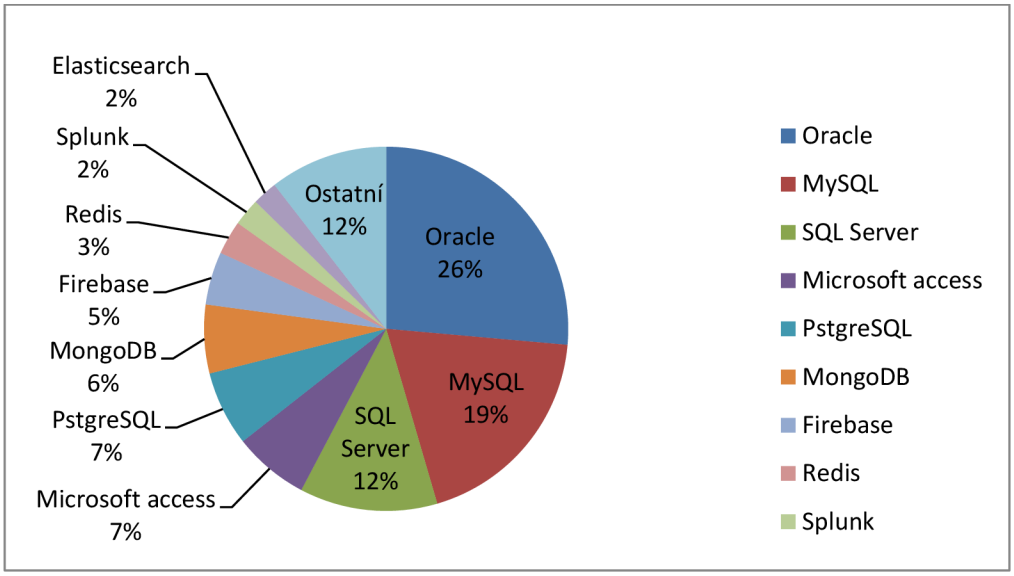
Data jsou zde uložena ve formě hran a uzlů. Hrana i uzel mohou mít libovolný počet atributů. Uzly mohou být označeny štítky a indexovány podle jednoho či více atributů. Štítky uzlů fungují jako sekundární indexování usnadňující práci s daty. Například uzel představující osobu může mít štítek „Person“. Vytváření záznamů a dotazování nad databází se provádí pomocí jazyka Cypher.

6.5.3 Využití v konkrétní aplikaci

Die Bayerische je německá společnost nabízející služby v oblasti pojištění. Tato společnost využívá Neo4J. Grafová databáze může perfektně reprezentovat vztahy mezi lidmi a produkty pojišťovny. Smlouvy jsou propojeny s lidmi a tito lidé jsou propojeni s dalšími lidmi s dalšími smlouvami. To vede ke kompaktní datové síti, ve které se lze díky grafovému přístupu snadno orientovat. Všechny relevantní informace tak lze získat velmi rychle. Vývojáři se rozhodli pro Neo4j kvůli snadné naučitelnosti a velkou flexibilitu datového modelu. (Neo4J, c2023)

7 Porovnání databází

TOPDB Top Database index popisuje oblíbenost a využití databází. Mezi prvních deset řadí Oracle, MySQL, SQL Server, Microsoft Access, PostgresSQL, MongoDB, Firebase, Redis, Splunk, Elasticsearch. (PYPL, 2023)



Graf 1: Využití databází v praxi. Zdroj: (vlastní zpracování na základě dat dostupných z: (PYPL, 2023))

Tato část se věnuje třem již dříve zmíněným databázím, které jsou v žebříčku oblíbenosti na druhé, šesté a osmé příčce. Těmi jsou MySQL, MongoDB a Redis. Konkrétně je tato část zaměřena na hardwarovou náročnost.

7.1 Hardwarová náročnost

Hardwarová náročnost bude měřena za předpokladu existence pouze jediného databázového uzlu, a to na základě času potřebného k vykonání základních operací. Pro účely testování byl vygenerován csv soubor se smyšlenými a náhodnými daty pomocí jazyka Java viz obr. níže. Soubor obsahuje 1 milion řádků, přičemž každý řádek představuje jeden záznam. Vkládání dat bude probíhat pomocí cyklu, a to pomocí jazyka Java.

Veškeré testy probíhaly na stejném zařízení s operačním systémem Windows 10 Home a následující hardwarovou konfigurací. Procesor Intel Core i5-9300H (2,40 - 4,10GHz), RAM 8GB DDR4 a SSD disk.

```
private static String chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ" + "0123456789" + "abcdefghijklmnopqrstuvxyz";
4 usages
private static String randomString(int lenghtOfString) {
    StringBuilder sb = new StringBuilder(lenghtOfString);
    for (int i = 0; i < lenghtOfString; i++) {
        int index = (int) (chars.length() * Math.random());
        sb.append(chars.charAt(index));
    }
    return sb.toString();
}
1 usage
public static void generateDataFile(int n) {
    String file = "";
    for (int i = 0; i < n; i++) {
        file = "";
        file = file + (i + 1) + "; ";                          //id
        file = file + randomString( lenghtOfString: 10) + "; ";     //field1
        file = file + randomString( lenghtOfString: 8) + "; ";      //field2
        file = file + randomString( lenghtOfString: 5) + "; ";      //field4
        file = file + randomString( lenghtOfString: 15) + "; ";     //field5
        file = file + (int) (Math.random() * (100 - 0) + 0) + "\n";
        System.out.println(i);
        try {
            Files.writeString(FILE_PATH, file, StandardCharsets.UTF_8, StandardOpenOption.APPEND);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    System.out.println("File was generate :)");
}
```

Obrázek 7: Kód pro generování souboru s testovacími daty. Zdroj: (vlastní zpracování)

Struktura vygenerovaného souboru:

1; SDPerF0vhc; vW3S1uaU; Vbpj7; QtAqXbc11WI202x; 15

2; bhe9so76hM; qVsqQbd9; EkhNI; 6EqlgOKsauSLDtE; 61

3; tcG6A59YKf; a9qGrpLD; PHEMk; Qdb7YeL5jZ7ep8J; 71

První sloupec představuje id záznamu, následující čtyři sloupce jsou zcela náhodně generované řetězce a poslední sloupec nabývá hodnot od 0 do 99.

7.1.1 MySQL

Pro účel testování byla vytvořena jedna tabulka o šesti sloupcích, přičemž každý sloupec v tabulce odpovídá jednomu sloupci ve vygenerovaném souboru.

7.1.1.1 Vkládání záznamů

```
public static void sqlInsert() {
    int i = 0;
    try (Connection mySqlCon = DriverManager.
            getConnection( url: "jdbc:mysql://localhost/test?user=root&password=password")) {
        while (scanner.hasNextLine()) {
            i++;
            tokenizer = new StringTokenizer(scanner.nextLine(), delim: "; ");
            PreparedStatement query = mySqlCon.prepareStatement( sql: "INSERT INTO test VALUES (?, ?, ?, ?, ?, ?)");
            query.setInt( parameterIndex: 1, (Integer.parseInt(tokenizer.nextToken())));
            query.setString( parameterIndex: 2, tokenizer.nextToken());
            query.setString( parameterIndex: 3, tokenizer.nextToken());
            query.setString( parameterIndex: 4, tokenizer.nextToken());
            query.setString( parameterIndex: 5, tokenizer.nextToken());
            query.setInt( parameterIndex: 6, (Integer.parseInt(tokenizer.nextToken())));
            query.executeUpdate();
            System.out.println(i);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Obrázek 8: Kód pro cyklické zapisování do MySQL. Zdroj: (vlastní zpracování)

Na obr. výše je ukázka kódu, který v cyklu zapisuje záznamy do databáze. Kód byl spuštěn v několika paralelních instancích.

Před prvním testem byla navýšena vyrovnávací paměť **innodb_buffer_pool_size** na 2G. Následný test ukázal, že jeden milion zápisů se uskutečnil během 815 sekund, v průměru tedy docházelo k 1227 zápisům za sekundu.

7.1.1.2 Úprava záznamů

V minulé části byla vytvořena tabulka *test* se šesti sloupci. *Id, field1, field2* až *field5,* přičemž *field5* obsahuje hodnotu 0 až 99. Toho bude využito pro následující test.

```
UPDATE test SET field2 = 'updated', field3 = '0<value< 80' WHERE field5 > 0 AND field5 < 80;
```

Podmínka „WHERE field5 > 0 AND field5 < 80“ vybere 790 098 řádků. Celý výše uvedený příkaz byl vykonán za 23 vteřin, což znamená, že probíhalo 34 352 updatů za vteřinu.

7.1.2 MongoDB

Záznam je zde v podobě objektu o pěti atributech. Každý atribut představuje hodnotu sloupce ze souboru.

7.1.2.1 Vkládání záznamů

Vkládání jednoho milionu záznamů do MongoDb trvalo oproti MySQL pouhých 156 sekund, probíhalo tak 6 410 zápisů za sekundu, což je cca pětinásobný počet s porovnáním s Mysql. Pro práci s MongoDb byl využit driver *mongodb-driver-sync*. Data byla vkládána pomocí kódu, viz obr. níže.

```
public static void mongoDbInsert() {
    int i = 0;
    try (MongoClient mongoClient = MongoClients.create("mongodb://localhost:27017")) {
        MongoDatabase database = mongoClient.getDatabase( s: "testdb");
        while (scanner.hasNextLine()) {
            i++;
            tokenizer = new StringTokenizer(scanner.nextLine(),  delim: "; ");
            tokenizer.nextToken();
            MongoCollection<Document> collection = database.getCollection( s: "test");
            Document document = new Document();
            document.put("field1", tokenizer.nextToken());
            document.put("filed2", tokenizer.nextToken());
            document.put("field3", tokenizer.nextToken());
            document.put("filed4", tokenizer.nextToken());
            document.put("field5", Integer.parseInt(tokenizer.nextToken()));
            collection.insertOne(document);
            System.out.println(i);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Obrázek 9:Kód pro cyklický zápis do MongoDB. Zdroj: (vlastní zpracování)

7.1.2.2 Úprava záznamů

```
db.testdb.updateMany({$and:[{field5: {$gt: 0}}, {field5: {$lt: 80}}]}, {$set:{field2: "updated", field3: "0<value<80"}})
```

Příkaz výše je alternativou příkazu, který byl použit při testování Mysql. Vybere stejné záznamy a ovlivní stejná pole. Provedení tohoto příkazu trvalo 13,63 sekund a probíhalo tak cca 57 967 updatů za sekundu. Oproti zápisu je tak v případě updatu rychlejší pouze o cca 69%.

7.1.3 Redis

V tomto případě byly provedeny dva různé testy zápisu. První využíval datovou strukturu hash, což umožňuje přístup ke každému hodnotě zvlášť, ale značně navyšuje náročnost zápisu. Druhý způsob využíval zápisu objektu v podobě json.

7.1.3.1Vkládání záznamů

V případě Redisu byly testovány dva způsoby. První způsob využíval strukturu hash a druhý ukládal objekt ve formátu json. Datový typ hash má tu výhodu, že lze přistupovat ke každému atributu zvlášť. Tento způsob zápisu trval 190 sekund.

Zápis v objektu v json formátu dopadl oproti hash formátu o něco lépe, trval pouhých 142 sekund, což je nerychlejší získaný výsledek.

Při využití datového typu hash tak probíhalo cca 5 263 záznamů za sekundu, při využití formátu json tato hodnota vzrostla na cca 7 042 záznamů.

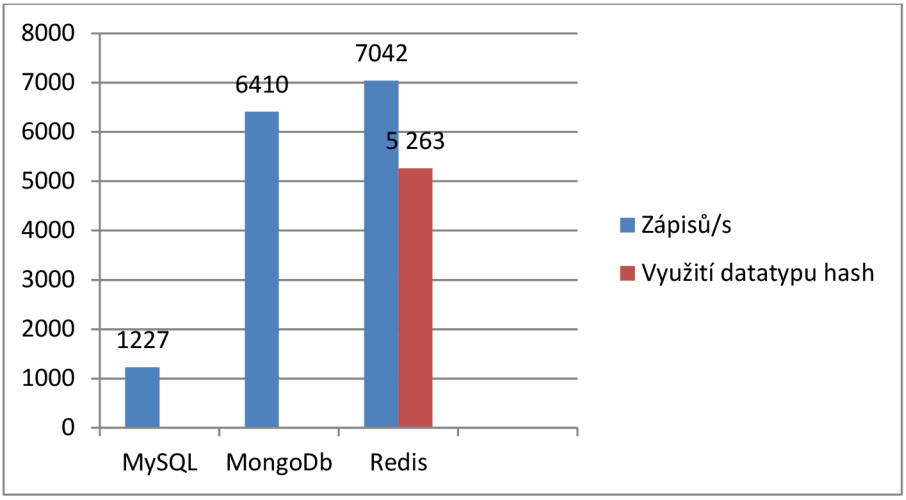
```
public static void redisInsert(String format) {
    int i = 0;
    try (Jedis jedis = new Jedis( url: "redis://localhost:6379")) {
        while (scanner.hasNextLine()) {
            tokenizer = new StringTokenizer(scanner.nextLine(), delim: "; ");
            int id = Integer.parseInt(tokenizer.nextToken());

            switch (format) {
                case ("hash"):
                    Map<String, String> map = new HashMap<String, String>();
                    map.put("field1" ,tokenizer.nextToken());
                    map.put("field2" ,tokenizer.nextToken());
                    map.put("field3" ,tokenizer.nextToken());
                    map.put("field4" ,tokenizer.nextToken());
                    map.put("field5" ,tokenizer.nextToken());
                    jedis.hmset( key: "test:" + id, map);
                    break;
                case ("json"):
                    jedis.set("test:" + id, mapper.writeValueAsString(new InputClass(tokenizer.nextToken(),
                            tokenizer.nextToken(),
                            tokenizer.nextToken(), tokenizer.nextToken(),
                            Integer.parseInt(tokenizer.nextToken()))));
                    break;
            }
            i++;
            System.out.println(i);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Obrázek 10: Kód pro cyklické zapisování do Redisu. Zdroj: (vlastní zpracování)

7.1.3.2 Úprava záznamů

Jak již bylo zmíněno v předchozích kapitolách, v případě databází klíč-hodnota se lze dotazovat pouze na celé klíče. To znamená, že nelze updatovat hromadně záznamy na základě podmínky. Nelze tedy vhodně porovnat s předchozími databázemi.

8 Shrnutí výsledků



Graf 2: Porovnání rychlosti zápisů za sekundu. Zdroj (vlastní zpracování)

Graf výše zobrazuje rychlosti zápisů jednotlivých databází. Lze z něj vyčíst, že v tomto ohledu je na tom nejlépe Redis, v těsném závěsu pak MongoDb a o nakonec o poznání hůře MySQL. Nicméně je nutno vzít v úvahu i ostatní faktory. MySQL jako jediná z těchto tří umožnuje využití integritních omezení a zaručuje tak stálou integritu dat. Redis a MongoDb nabízejí horizontální škálování, snadnou práci s nestrukturovanými daty a vysokou rychlost. Nehlídají však integritu dat. V případě Redisu nelze oproti předchozím dvěma dotazovat hodnoty, ale pouze celé klíče. Je proto nutné zvážit veškeré faktory.

8.1 Vhodnost využití ve webové aplikaci

Před volbou databázového systému je nutné nejprve provést analýzu dat. Je nutné si uvědomit, zdali se data mohou v průběhu rapidně změnit, budou mít často redundantní obsah, jaké velikosti může nabýt dataset, zdali budou některá data pouze dočasná a v neposlední řadě je nutné si uvědomit požadavky na integritu.

8.1.1 Relační databáze

V případě, že jsou vyžadovány vlastnosti ACID, tedy data musí být v každém případě konzistentní, jsou relační databáze nejlepší volbou. Nicméně použití relačních databází vyžaduje znalost povahy dat, jejichž struktura musí být stálá. Pokud není povaha dat jednoznačná, je na povážení, zdali nevyužít alternativy v podobě

NoSQL databáze. To sebou ale přináší určitá rizika, protože NoSQL databáze se nevyznačují vlastnostmi ACID ale BASE, to může znamenat potíže v případě, že je potřeba naprostá konzistence dat. Výhodou relačních databází je také atomicita dat, která minimalizuje redundanci a možnost modelování vztahů mezi daty.

Relační databáze jsou tedy nejvhodnějším kandidátem pro systémy zabývající se věcmi, jako jsou např. finanční transakce, kde jsou data stélé struktury, mají mezi sebou vztahy a musejí zachovat svojí konzistenci i v případě jakéhokoli selhání.

8.1.2 Dokumentové databáze

Databáze dokumentů jsou praktickým řešením např. katalogů nebo online profilů, ve kterých různí uživatelé poskytují různé typy informací. Díky absenci schématu lze uložit pouze atributy, které jsou specifické pro každého uživatele. Nestálá povaha dat zde není problém. Jednotlivým dokumentům lze libovolně přidávat, odebírat a aktualizovat atributy. Ačkoli je struktura dat zcela volitelná, je žádoucí, aby data ve stejné kolekci měla alespoň podobnou strukturu pro snadnou práci s nimi. K dokumentům lze přistupovat na základě párů klíč-hodnota.

Dokumentové databáze jsou také horizontálně škálovatelné a jsou tak vhodné pro správu velkého množství dat.

8.1.3 Databáze klíč hodnota

„Webové aplikace mohou ukládat podrobnosti o uživatelské relaci a preference v úložišti párů klíč–hodnota. Všechny informace jsou přístupné pomocí uživatelského klíče a databáze s páry klíč-hodnota se hodí pro rychlé čtení a zápis.

Tento druh databází se často využívá pro různá doporučení a zobrazování reklam v reálném čase, když se návštěvník pohybuje po webu.

*Pokud jde o technickou stránku, úložiště klíč-hodnota se běžně používá pro ukládání dat do mezipaměti, čímž lze urychlit aplikaci díky minimalizaci čtení a zápisu z disku. Redis je příkladem technologie, která poskytuje úložiště klíč-hodnota v paměti pro rychlé načítání dat.“ (*Vlastní překlad); (Hazelcast, c2023)

8.2 Výzkumné otázky

Tato práce zmiňuje tři výzkumné otázky, ze kterých lze vyvodit stejný počet hypotéz, jejichž potvrzením či vyvrácením lze zodpovědět výše zmíněné otázky. První hypotéza tvrdí, že databáze NoSQL jsou vhodnější pro velké množství nestrukturovaných dat ve webových aplikacích. Druhá předpokládá, že SQL databáze jsou lepší pro webové aplikace s komplexními vztahy mezi daty. Třetí hypotéza tvrdí, že SQL databáze jsou lepší pro webové aplikace, které musí splňovat přísné požadavky na konkrétní data.

První hypotéza tvrdí, že databáze NoSQL jsou vhodnější pro velké množství nestrukturovaných dat ve webových aplikacích. Je pravda, že databáze NoSQL často poskytují flexibilnější schéma nebo jsou dokonce bez schématu, což umožňuje ukládat a zpracovávat nestrukturovaná data efektivněji. NoSQL databáze tak mohou být výhodné pro aplikace, které potřebují rychle a efektivně zpracovávat data s nestálou strukturou. Absence schématu umožňuje snadnější rozšiřitelnost a agilitu v případě změn struktury dat. První hypotéza je tedy pravdivá.

Druhá hypotéza tvrdí, že SQL databáze jsou lepší pro webové aplikace s komplexními vztahy mezi daty. Je sice pravda, že SQL databáze jsou výborné pro modelování a správu vztahů mezi daty pomocí relačního schématu. Nicméně je důležité zmínit, že existují také grafové databáze, které se řadí do kategorie NoSQL a vynikají právě v modelování a dotazování na komplexní vztahy mezi daty. Grafové databáze jsou zvláště užitečné v sociálních sítích, dopravních sítích nebo analýze doporučení. Z tohoto pohledu lze říci, že druhá hypotéza je částečně pravdivá, ale je třeba brát v úvahu i grafové databáze.

Třetí hypotéza tvrdí, že SQL databáze jsou lepší pro webové aplikace, které musí splňovat přísné požadavky na konkrétní data. SQL databáze nabízejí vysokou konzistenci dat díky transakčnímu zpracování a integritním omezením. Jsou vhodné pro systémy pracující s daty stálé struktury, která musejí být vždy konzistentní

(vyjma samotného průběhu transakce), například v bankovnictví nebo rezervačních systémech. Třetí hypotéza lze tedy považovat za pravdivou.

9 Závěry a doporučení

Předmětem této práce bylo porovnání SQL a NoSQL databází v kontextu použití ve webových aplikacích. Z práce vyplynulo, že při výběru vhodné databázové technologie je důležité provést pečlivou analýzu dat a zvážit jejich požadavky na konzistenci a rychlost zpracování.

NoSQL databáze se vyznačují rychlostí zpracování dat a jsou tak vhodné pro aplikace, které potřebují zpracovat velké množství dat v reálném čase. Na druhou stranu, tato rychlost je dosažena na úkor záruky konzistence dat, což může být pro některé aplikace problematické.

Naopak SQL databáze jsou vhodné pro aplikace, které vyžadují naprostou konzistenci dat. Výhodou SQL databází je také jejich schopnost efektivně pracovat s vazbami mezi různými tabulkami a umožňovat složitější dotazy nad daty.

NoSQL databáze na druhé straně umožňují snadný přechod mezi různými formáty dat díky absenci schématu. To může být užitečné pro aplikace, které pracují s nestrukturovanými daty nebo pro aplikace, které se pravděpodobně budou vyvíjet a měnit v budoucnosti.

Doporučením z této práce pro vývojáře webových aplikací by bylo, aby při výběru databázové technologie zvážili požadavky aplikace na rychlost a konzistenci dat a na základě toho vybrali vhodnou technologii. Pokud aplikace potřebuje rychlost a zpracování velkého množství dat, je NoSQL databáze lepší volbou. Pokud aplikace vyžaduje naprostou konzistenci dat, SQL databáze jsou vhodnější.

V případě, že aplikace pracuje s nestrukturovanými daty nebo se bude pravděpodobně vyvíjet a měnit v budoucnosti, může být výhodné zvolit NoSQL databázi. Je také vhodné zvážit, jak bude databáze používána v aplikaci a zda je potřeba přistupovat k datům z více míst. Pokud tomu tak je, může být výhodné použít distribuovanou NoSQL databázi.

V každém případě je důležité provést pečlivou analýzu dat a zvážit požadavky aplikace.

10 Seznam použité literatury

Die Bayerische Case Study. *Neo4j* [online]. Neo4j, c2023 [cit. 2023-04-26]. Dostupné z: https://neo4j.com/case-studies/die-bayerische/?ref=who-uses

Documentation. *Redis* [online]. c2023 [cit. 2023-04-26]. Dostupné z: https://redis.io/docs/

DRKUSIC, Emil. Learn SQL: SQL Triggers. *SQLShack* [online]. 2020 [cit. 2022-09-02]. Dostupné z: www.sqlshack.com/learn-sql-sql-triggers/

Hierarchical Database. *Heavy.Ai* [online]. c2022 [cit. 2023-08-02]. Dostupné z: https://www.heavy.ai/technical-glossary/hierarchical-database

MEIER, Andreas a Michael KAUFMANN. *SQL & NoSQL databases: models, languages, consistency options and architectures for Big data management*. Wiesbaden: Springer, [2019]. ISBN 978-3-658-24548-1

MongoDB Real World Use Cases: Advantages & Top Companies [2023]. *UpGrad* [online]. 2022 [cit. 2023-07-02]. Dostupné z: https://www.upgrad.com/blog/mongodb-real-world-use-cases/

MONTROSE, Kevin. Does Stack Exchange use caching and if so, how?. *Meta* [online]. 2016 [cit. 2023-04-26]. Dostupné z: https://meta.stackexchange.com/questions/69164/does-stack-exchange-use-caching-and-if-so-how/69172#69172

MPP & Columnar Databases. *Insightsoftware* [online]. 2021 [cit. 2023-04-26]. Dostupné z: https://insightsoftware.com/blog/mpp-columnar-databases/

NAZRUL, Syed Sadat. CAP Theorem and Distributed Database Management Systems. *Towards Data Science* [online]. 2018 [cit. 2023-04-26]. Dostupné z: https://towardsdatascience.com/cap-theorem-and-distributed-database-management-systems-5c2be977950e

NOACH, Shlomi. MySQL High Availability at GitHub. *Github Blog* [online]. 2018 [cit. 2023-04-26]. Dostupné z: https://github.blog/2018-06-20-mysql-high-availability-at-github/

POKORNÝ, Jaroslav a Michal VALENTA. *Databázové systémy*. 2. přepracované vydání. Praha: Česká technika - nakladatelství ČVUT, 2020. ISBN 978-80-01-06696-6.

SCHWARTZ, Baron, Peter ZAITSEV a Vadim TKACHENKO. High Performance MySQL. *Springer Link* [online]. O'Reilly Media, 2012 [cit. 2023-04-26]. Dostupné z: https://www.oreilly.com/library/view/high-performance-mysql/9781449332471/ch01.html

SINHA, Shubham. How Was HBase Used To Solve Storage Issues In Facebook Messenger?. *Edureka* [online]. 2016 [cit. 2023-04-26]. Dostupné z: https://medium.com/edureka/hbase-tutorial-bdc36ab32dc0

TOPDB Top Database index. *PYPL Github* [online]. 2023 [cit. 2023-04-26]. Dostupné z: https://pypl.github.io/DB.html

Understanding the Network Database Model. *MariaDB* [online]. c2023 [cit. 2023-08-02]. Dostupné z: https://mariadb.com/kb/en/understanding-the-network-database-model/

VARMA, Srikanth. SQL Commands: DDL, DML, DCL, TCL, DQL. *Scaler* [online]. 2023 [cit. 2023-08-02]. Dostupné z: https://www.scaler.com/topics/dbms/sql-commands/

What is a key-value store?. *Hazelcast* [online]. Hazelcast, c2023 [cit. 2023-04-26]. Dostupné z: https://hazelcast.com/glossary/key-value-store/

What is a Relational Database (RDBMS)? *Oracle* [online]. c2023 [cit. 2023-08-02]. Dostupné z: https://www.oracle.com/database/what-is-a-relational-database/

UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Akademický rok: 2020/2021

Studijní program: Aplikovaná informatika
Forma studia: Prezenční
Obor/kombinace: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

Jméno a příjmení:	**Ondřej Krkoška**
Osobní číslo:	**I1900213**
Adresa:	Křovická 276, Dobruška, 51801 Dobruška, Česká republika
Téma práce:	Porovnání možností databází SQL a NoSQL pro využití ve webových aplikacích
Téma práce anglicky:	Comparison of SQL and NoSQL database for web applications
Jazyk práce:	Čeština
Vedoucí práce:	doc. RNDr. Petra Poulová, Ph.D. Katedra informatiky a kvantitativních metod

Zásady pro vypracování:

Cílem bakalářské práce je porovnání výhod a nevýhod databází SQL a NoSQL pro využití ve webových aplikacích. Důležitými kritérii pro porovnání budou především: složitost návrhu a implementace, způsob ukládaní informací a jejich použitelnost, náročnost na hardwarové prostředky. Dále autor porovná vhodnost využití dané databáze v konkrétní webové aplikaci.

Seznam doporučené literatury:

Podpis studenta: Datum:

Podpis vedoucího práce: Datum: