Analýza výkonu vybraných databázových technologií

Filip Sedlář

Bakalářská práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Filip Sedlář**
Osobní číslo:	**A21355**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Analýza výkonu vybraných databázových technológií**
Téma práce anglicky:	**Performance Analysis of Selected Database Technologies**

Zásady pro vypracování

1. Popište problematiku rozhodovacího procesu při výběru databázových technologií.
2. Stanovte hlavní rozdíly mezi relačními a NoSQL databázovými technologiemi formou komparativní analýzy.
3. Navrhněte a implementujte benchmarkovou aplikaci pro demonstraci kvalitativních rozdílů zkoumaných databázových technologií.
4. Popište postup měření výkonnosti relačních a NoSQL databází prováděný pomocí vytvořené benchmarkové aplikace.
5. Zhodnoťte a prezentujte naměřené výsledky, okomentujte zjištěné datové trendy.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. HOLUBOVÁ, Irena, Jiří KOSEK, Karel MINAŘÍK a David NOVÁK. Big Data a NoSQL databáze. Praha: Grada, 2015. Profesionál. ISBN 978-80-247-5466-6.
2. LAURENČÍK, Marek. SQL: podrobný průvodce uživatele. Praha: Grada Publishing, 2018. Průvodce (Grada). ISBN 978-80--271-0774-2.
3. Angular Docs – Oficiální Dokumentace Frameworku Angular. Online. 2016, 2023. Dostupné z: https://angular.io/docs. [cit. 2023-11-09].
4. Microsoft Docs – Microsoft SQL Documentation: Oficiální Dokumentace Společnosti Microsoft Corporation. Online. 2023. Dostupné z https:
learn.microsoft.com/en-us/sql. [cit. 2023-11-09].

Vedoucí bakalářské práce: **Ing. Jozef Kováč**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **5. listopadu 2023**
Termín odevzdání bakalářské práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.
- že při tvorbě této práce jsem použil nástroj generativního modelu AI ChatGPT; https://chatgpt.com za účelem úpravy textu, parafrázování a překladu. Po použití tohoto nástroje jsem provedl kontrolu obsahu a přebírám za něj plnou zodpovědnost.

Ve Zlíně, dne 13. 05. 2024 Filip Sedlář, v.r.

ABSTRAKT

Bakalářská práce se zabývá porovnáním relačních a NoSQL databází, s cílem identifikovat klíčové aspekty ovlivňující výběr databázové technologie pro různé typy aplikací. V teoretické části je poskytnut přehled o charakteristikách a rozdílech mezi těmito systémy. Praktická část práce je zaměřena na vývoj a použití benchmarkové aplikace pro testování výkonnosti databází. Výsledky demonstrují vliv výběru databáze na výkon aplikací a jejich výkonnost při testování na různých operacích.

Klíčová slova: databázové systémy, SQL, NoSQL, výkonnost databází, komparace

ABSTRACT

The bachelor thesis focuses on comparing relational and NoSQL databases, with the objective of identifying key aspects that influence the choice of database technology for various types of applications. The theoretical part provides an overview of the characteristics and differences between these systems. The practical part is devoted to the development and use of a benchmarking application for testing database performance. The results demonstrate the impact of database selection on application performance and their efficiency in testing across some operations.

Keywords: database systems, NoSQL, SQL, database performace, comparison

PODĚKOVÁNÍ

Chci poděkovat mému vedoucímu Ing. Jozefu Kováčovi za jeho neustálé rady, návrhy či poznatky při tvoření téhle bakalářské práce a jsem mu hluboce vděčný za jeho trpělivost a podporu.

Jako další chci poděkovat své rodině za neustálou podporu i v těžkých časech studia.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

V dnešní době, kdy technologie jdou stále dopředu, se velké objemy dat neustále generují a zpracovávají a z tohohle důvodu je důležité mít efektivní a výkonné databázové technologie, které umožní spolehlivé a rychlé operace. Proto je analýza výkonu těchto technologií považována za zásadní nástroj pro zajištění jejich maximální efektivity a pro optimalizaci jejich použití v různých aplikacích. Databázové systémy se používají skoro všude, a proto je velmi důležité vědět jaké jsou nejlepší a nejvhodnější k použití v různých oblastech informačních technologií.

Motivací pro vypracování téhle bakalářské práce, je potřeba porozumět a také vyhodnotit výkonnostní charakteristiky různých databázových technologií ve spojitosti se současnými informačními požadavky a potřeby. S neustálým nárustem komplexitou dat, nároky na rychlost a spolehlivostí zpracování dat je důležité provést důkladnou analýzu a porovnání těchto technologií.

Cílem této bakalářské práce je provést danou analýzu výkonu těchto vybraných databázových technologií a porovnat jejich schopnosti a omezení v různých případech a za různých podmínek. Konkrétně je tato práce zaměřena na zhodnocení jejich schopností zpracovávat data v reálném čase.

Teoretická část bakalářské práce je zaměřena na seznámení s kritérii a požadavky při výběru takových databázových technologií. Následně zde bude vysvětlena problematika výběru databázových technologií a v další kapitole bude komparativní analýza databázových technologií, a to relačních a NoSQL.

Praktická část bude zaměřena na návrh benchmarkové aplikace, která nám umožní systematické testování výkonu a porovnání jednotlivých databázových technologií. Za účelem analýzy výkonu databázových technologií budou prováděny testy, během nichž budou měřeny a vyhodnocovány klíčové výkonnostní ukazatele.

I. TEORETICKÁ ČÁST

1 ROZHODOVACÍ PROCES PŘI VÝBĚRU DATABÁZOVÝCH TECHNOLOGIÍ

Při rozhodování o tom, jaká databázová technologie bude nejlepší pro určitý projekt se musí brát v potaz potřeby projektu, tak aby daná databázová technologie splňovala dané potřeby pro daný projekt. Potřeb je několik, a proto se musí programátor rozhodnout, jaká databázová technologie bude pro něho nejvhodnější. Databázové technologie mají každý vlastní databázový systém, který umožňuje spravovat a komunikovat s daty dané databázového technologie.

1.1 Kritéria výběru databázových systémů

Mezi kritéria pro vybrání nejvhodnější databáze spadá několik požadavků, které ovlivňují funkcionalitu a výkonnost.

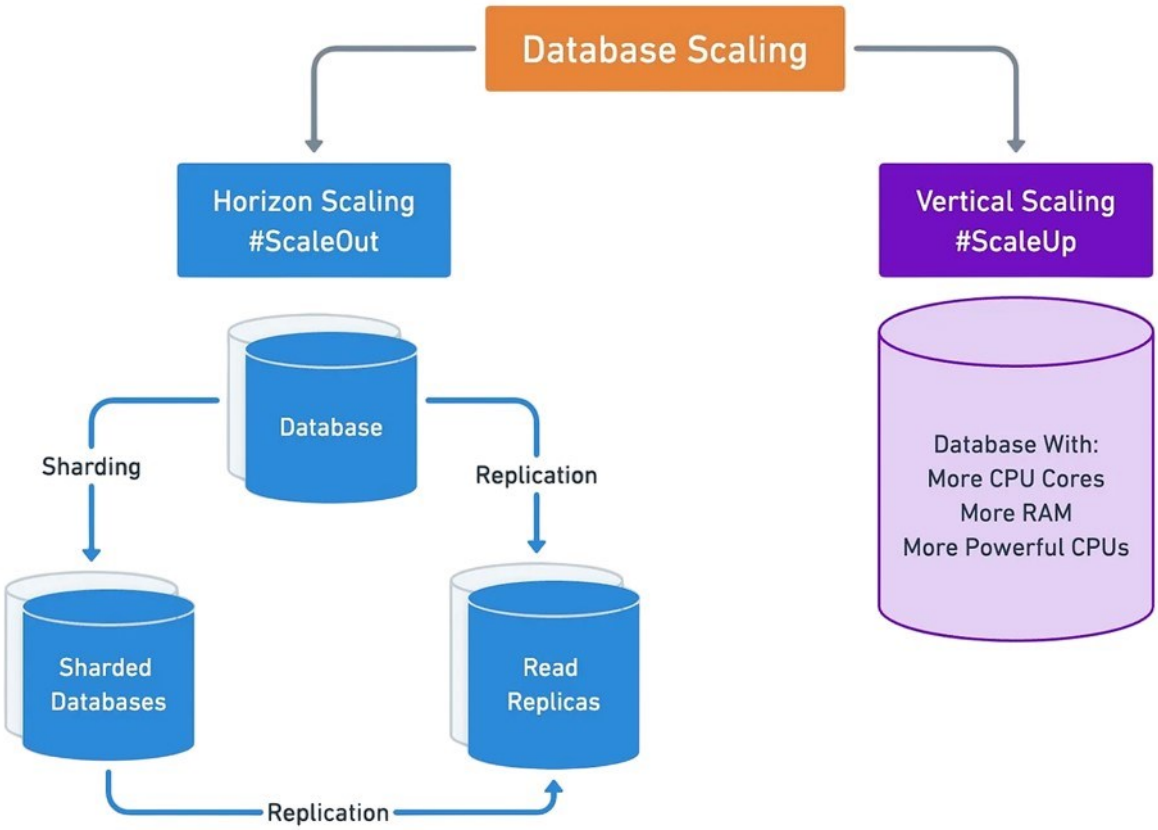
1.1.1 Škálovatelnost

Škálovatelnost databáze je klíčová pro aplikace, které očekávají nárůst objemu dat, uživatelů a typů dotazů, aniž by došlo k poklesu výkonu nebo dostupnosti. Na tuto schopnost databáze reagovat na rostoucí požadavky se dosahuje buď rozšířením zdrojů – hardware nebo software, nebo optimalizací její struktury a nastavení, případně kombinací obou přístupů.

Rozlišujeme dva základní směry škálovatelnosti: vertikální a horizontální. Vertikální škálovatelnost znamená zvyšování výpočetního výkonu a paměti jednoho serveru, zatímco horizontální škálovatelnost se týká přidávání dalších serverů pro rozložení zátěže mezi více uzlů.

Při horizontálním škálování hrají důležitou roli metody jako sharding nebo replication, které umožňují rozdělení a distribuci dat mezi více serverů, což vede ke zvýšení výkonu a spolehlivosti databáze.

Škálovatelnost databáze je nezbytná pro systémy, které potřebují zpracovávat velké množství dat, jakými jsou například e-commerce platformy, sociální sítě nebo finanční služby. Návrh škálovatelné databáze pomáhá předcházet výkonnostním problémům, snižovat riziko výpadků a zajišťuje dobré uživatelské zážitky i při rostoucím zatížení [1].



Obrázek 1. Škálování databázového systému [3]

1.1.2 Výkonnost

Výběr databáze je zásadní pro zajištění rychlé a efektivní práce aplikace, protože výkon databáze má bezprostřední vliv na její reaktivitu. Při posuzování vhodnosti databáze je důležité zvážit, jak rychle dokáže databáze zpracovávat čtení a zápis dat, její schopnost efektivně řešit náročné dotazy a zvládat rozsáhlé množství informací. Klíčové vlastnosti, které by měly být přítomny, zahrnují indexování, caching mechanismus a optimalizaci dotazů, jež přispívají k lepšímu výkonu. Je také podstatné, aby databáze mohla být efektivně horizontálně škálována, což umožňuje zachování vysoké rychlosti i při zvyšujícím se objemu dat a počtu uživatelů. Provádění testů výkonu a benchmarků nabízí klíčový pohled na to, jak databáze funguje v podmínkách blízkých reálnému nasazení [2].

1.1.3 Datový model

Struktura dat aplikace a jejich uspořádání jsou definovány pomocí datového modelu. Na výběr je mnoho typů databází, každá s vlastním přístupem k ukládání dat – od relačních modelů přes dokumentově orientované až po modely založené na klíči a hodnotě, grafy nebo sloupcové struktury.

Při rozhodování, který model je pro vaše data nejvhodnější, zohledněte jejich vlastnosti a metody, jakými budou data vyhledávána a zpracovávána. Pro data s pevnou strukturou a potřebou komplexních dotazů může být ideální volbou relační databáze. Pro méně strukturovaná data bez předem definovaného schématu by naopak mohla lépe posloužit databáze typu dokumentově orientované nebo NoSQL [viz. kapitola 2. – „Komparativní analýza
relačních a NoSQL technologií“].

Výběrem vhodného datového modelu se zefektivňuje ukládání a získávání dat, což přináší větší flexibilitu a usnadňuje proces vývoje [2].

1.1.4 Parametry databáze

Pro výběr dobrého databázové systému se musí dbát také na parametry, tak aby vyhovoval potřebám aplikace.

Příklad daných parametrů:

- **Rychlost** – Rychlost zpracování dat je důležitá pro efektivní vyhodnocení informací a k rychlému rozhodování založeném na datech [4].
- **Velikost** – Správný návrh velikosti databáze může napomoct k výběru hardwaru, který pomůže dosáhnout požadované rychlosti pro danou aplikaci [5].

1.1.5 Bezpečnost

Bezpečnost dat je klíčovým aspektem, zejména když aplikace manipuluje s citlivými nebo osobními údaji. Je důležité zhodnotit, jaké bezpečnostní prvky databáze poskytuje, včetně mechanismů řízení přístupu, šifrování uložených dat a dat přenášených mezi servery, schopností provádět audit a shodu s normami pro ochranu dat. Musí se zajistit, aby databáze nabízela spolehlivé metody pro ověřování a udělování oprávnění uživatelům přistupujícím k datům. Indikátory bezpečné databáze zahrnují také pravidelné aktualizace zabezpečení a prokázanou schopnost adresovat potenciální bezpečnostní hrozby [2].

1.1.6 Cena

Náklady jsou klíčovým faktorem v procesu výběru databáze. Je důležité zvážit jak počáteční, tak i průběžné výdaje, které databáze přináší. Přemýšlejte o různých aspektech, jako jsou náklady na licence, vyžadovaný hardware, výdaje na údržbu, podporu a možnou potřebu zaškolení databázových specialistů [2].

Také nezapomeňte na náklady spojené se škálováním a rozvojem aplikace v budoucnu. Databáze s otevřeným zdrojovým kódem mohou představovat finanční ušetření, avšak je důležité ověřit, zda odpovídají vašim potřebám a zda existuje spolehlivá podpora pro případ potřeby [2].

1.2 Současné trendy v databázových technologií

V posledních letech byly pozorovány významné změny ve vývoji a využívání databázových technologií. Tyto změny jsou poháněny rostoucí potřebou zpracování velkého množství dat a zajištění jejich rychlé dostupnosti a bezpečnosti. Důraz je kladen na vývoj a implementaci technologií, které umožňují efektivnější správu a analýzu dat, a to i v reálném čase.

1.2.1 Cloudové databáze

V nedávné době bylo uznáno, že cloudové systémy pro správu databází se staly nepostradatelnou součástí IT infrastruktur díky řadě klíčových předností. Možností flexibilního škálování databázových zdrojů bez nutnosti investovat do hardwaru bylo dosaženo velkého ocenění. Je poskytována úleva IT oddělením díky automatizovaným službám aktualizací a údržby, které zajišťují stále aktualizace a bezpečnost databázových systémů [6]

Flexibilita práce a zvýšení produktivity spolupráce jsou podporovány umožněním přístupu zaměstnanců k datům odkudkoli. Riziko ztráty dat je minimalizováno prostřednictvím pokročilých řešení pro zálohování a ochranu dat, která jsou poskytovatelé cloudových služeb schopni nabídnout. Další rozšiřování schopností organizací v oblastech správy a analýzy dat je usnadněno díky snadné integraci cloudových databázových platforem s dalšími cloudovými nástroji a službami [6].

V současné době Microsoft vede v oblasti cloudových technologií, následovaný blízko za sebou Amazon Web Services, Google Cloud Platform a Alibaba Cloud. I po komplikovaném přechodu na cloudové systémy si Oracle, IBM a SAP zachovávají důležitou pozici na trhu,

což jasně ukazuje, že cloudové technologie jsou nezpochybnitelně klíčovým prvkem v dnešním technologickém světě [7].

1.2.2 Umělá inteligence

Integrace umělé inteligence do řízení databází je vnímána jako účinný způsob, jakým lze podstatně snížit nároky na údržbu. Díky začlenění umělé inteligence do databází a jejich infrastruktury je možné, aby byly správcům databází umožněny identifikace nedostatků v úložišti a paměti, společně s dalšími problémy, které mohou narušovat provoz databází [6].

1.2.3 In – memory databáze

In – memory databáze se vyznačují ukládáním informací přímo do hlavní paměti počítače, což jim umožňuje poskytovat odpovědi s výrazně nižší latencí ve srovnání s tradičními diskovými systémy. Rychlost reakce je zlepšena, jelikož neexistuje potřeba datové transformace nebo jejich ukládání do mezipaměti – data jsou okamžitě přístupná v rámci systému. Sektory jako herní průmysl, telekomunikace, bankovnictví a turistický sektor zaznamenávají značné výhody z používání in – memory databáze [8].

Stávající trend využívání in – memory databází je hnán jejich schopností nabízet rychlejší přístup k datům, než je možné u disků či datových jezer [8].

1.2.4 Propojení SQL a NoSQL

Dříve byly databáze klasifikovány primárně jako SQL nebo NoSQL. Nicméně, nejnovější vývoj v technologiích nyní umožňuje vytváření propojení mezi těmito dvěma typy databází. Tyto propojení, známé jako data warehouses a lakehouses, nabízí uživatelům kombinaci předností obou systémů. Díky nim je možné přistupovat k NoSQL databázím tím samým způsobem, jak se přistupuje k databázím SQL [8].

- Data warehouse představuje klíčový systém řízení dat, který je zaměřen na podporu a usnadnění činností v rámci business intelligence, hlavně v analýze dat. Jeho primárním účelem je agregace a integrace rozsáhlých datových sad z rozličných zdrojů, což organizacím poskytuje podstatné náhledy pro zlepšení procesu rozhodování. Skladování historických dat činí z data warehouse nepostradatelný nástroj pro datové specialisty a business analytiky, čímž se stává základním „zdrojem pravdivých informací“ pro firmu [9].

- Data lakehouse propojuje principy data lakehouse a data warehouse, čímž umožňuje využívání strojového učení, business intelligence a prediktivních analýz. Tento model nabízí efektivní řešení pro ukládání různorodých typů dat – ať už strukturovaných, nestrukturovaných nebo polostrukturovaných – a zároveň zachovává možnost strukturovaného dotazování a správy dat. Výsledkem je systém, který umožňuje organizacím efektivně využívat data pro podporu rozhodovacích procesů a získávání obchodních vhledů [10].

1.2.5 Edge Computing

V nedávné době se edge computing stal klíčovou technologií, která zásadně mění procesy zpracování a analýzy dat. Základem tohoto přístupu je přibližování zpracování dat k jejich zdrojům. Tato metoda zefektivňuje celý proces tím, že omezuje potřebu přenosu dat do vzdálených datových center. Ve spojení se systémy pro správu databází edge computing zrychluje přístup k datům a umožňuje jejich ukládání v místě vzniku, čímž se snižuje latence a zvyšuje efektivita. Tento přístup nachází především uplatnění u IoT, kde je schopnost okamžitého zpracování dat nezbytná pro jejich správné fungování [8] [11].

2 KOMPARATIVNÍ ANALÝZA RELAČNÍCH A NOSQL TECHNOLOGIÍ

Komparativní analýza relačních a NoSQL technologií představuje klíčovou část v širším kontextu pochopení současných trendů a výzev v oblasti správy databází. Relační databázové systémy, které byly dominantním modelem po desetiletí, se zaměřují na pevně strukturovaná data a jejich integritu, zatímco NoSQL technologie nabízí flexibilitu, škálovatelnost a efektivitu při práci s různorodými datovými strukturami. Tato kapitola má za cíl prozkoumat, porovnat a vyhodnotit klíčové charakteristiky, výhody a omezení obou přístupů.

Vývoj digitálních technologií a exponenciální nárůst objemu dat vedl k nutnosti adaptace a inovace v databázových systémech. Zatímco relační databáze nabízejí robustní řešení pro transakční systémy a komplexní dotazování, stávají se NoSQL databáze odpovědí na potřebu rychle zpracovávat velké objemy nestrukturovaných dat a poskytovat vysokou dostupnost v distribuovaných systémech. Tato analýza má za úkol objasnit, v jakých situacích je vhodnější upřednostnit jednu technologii před druhou, a jak mohou společnosti efektivně využít oba typy databází k dosažení svých obchodních cílů.

2.1 Základní principy relačních databází

V podkapitole bude představena podstata relačních databází, jež jsou považovány za jeden z nejfundamentálnějších a nejdůležitějších konceptů v doméně správy databází. Založené na relačním modelu dat, který byl navržen Edgar F. Coddem v roce 1970, se relační databázové systémy staly esenciálním pilířem pro ukládání, manipulaci a retrieval strukturovaných informací. Charakteristickým rysem těchto systémů je reprezentace dat a jejich vzájemných vztahů prostřednictvím tabulek, což umožňuje efektivní organizaci a přístup k informacím.

Mezi hlavní principy, kterými jsou relační databáze definovány, patří integrita dat, normalizace, transakce a použití jazyka SQL pro dotazování a aktualizaci dat. Díky těmto principům je zajištěno, že relační databáze poskytují konzistentní, spolehlivé a bezpečné prostředí pro správu dat, což je nezbytné pro širokou škálu aplikací, od podnikových informačních systémů až po webové služby.

2.1.1 Integrita dat

Integrita dat se týká zachování jejich přesnosti, konzistence a úplnosti po celou dobu jejich existence. Tento aspekt je zásadní pro systémy, které manipulují s daty nebo je archivují, jelikož zabraňuje jejich ztrátě a nechtěným únikům. Proces zajištění, že data zůstanou integrována během času a přes různé formáty, vyžaduje stálou pozornost a zahrnuje řadu metod, pravidel a norem [12].

Pro zajištění integrity v hierarchických i relačních databázích jsou rozlišovány dva základní typy – fyzická a logická integrita. Tyto sady pravidel a procedur jsou využívány aplikačními programátory, systémovými programátory, manažery datového zpracování a vnitřními auditory k zajištění správnosti dat [12].

2.1.1.1 Fyzická integrita

Ochrana dat před vnějšími hrozbami, jako jsou přírodní události, výpadky proudu nebo hackerské útoky, patří do oblasti fyzické integrity. Lidské chyby nebo opotřebení úložiště mohou také zabránit přístupu k informacím v databázi [13].

2.1.1.2 Logická integrita

Je zajištěno, že data zůstávají konstantní během jejich aplikace rozličnými postupy v kontextu relačních databází. Tento proces také směřuje k ochraně informací před neoprávněnými zásahy hackerů nebo omyly lidského faktoru, avšak používá odlišný přístup než v případě zajištění fyzické integrity [14]. Logická integrita se dělí na čtyři podskupiny:

- Entitní integrita – využívá primární klíče pro unikátní rozpoznání záznamů v tabulce relační databáze, čímž dochází k eliminaci možnosti jejich duplikace. Z tohoto důvodu nemůže být hodnota primárního klíče nastavena na NULL, jelikož by pak nebylo možné řádek unikátně identifikovat, pokud by se ostatní data v řádcích shodovala.

Primary Key	Customer ID	Customer name	Age
	44922945	Oliver Twist	34
	30091920	James Gatz	42
Value cannot be NULL		James Gatz	42

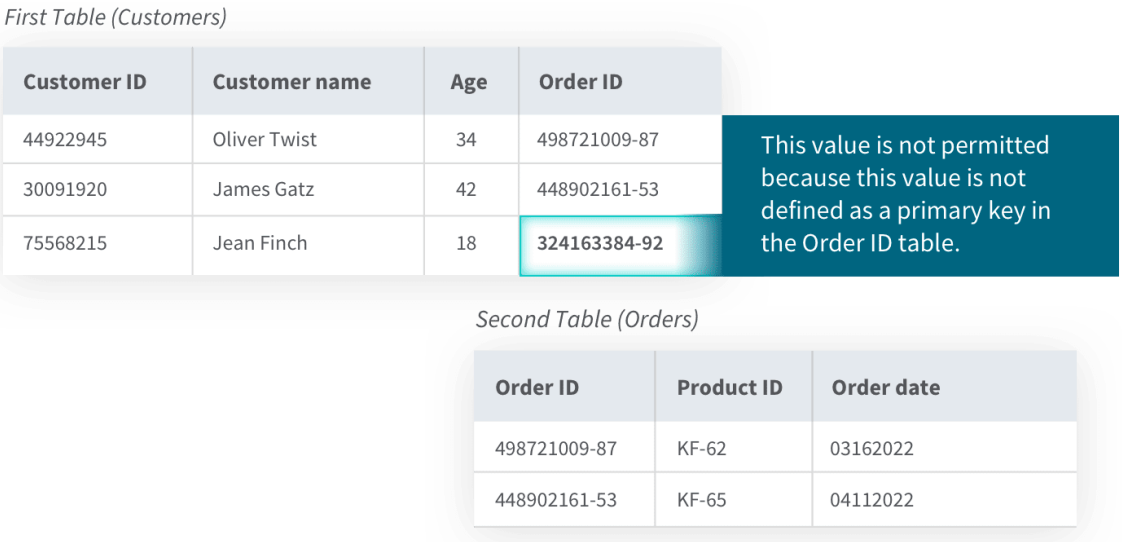
Obrázek 2. Příklad Entitní integrity [12]

- Doménová integrita – obsahuje sady pravidel a procesů, které omezují formát, typ a množství dat zapsaných do databáze. To zaručuje, že všechny sloupce v relační databázi odpovídají stanovené doméně.

Customer ID	Customer name	Age	Orders
44922945	Oliver Twist	34	21
30091920	James Gatz	42	9
75568215	Jean Finch	18	w#2@jk_1
			Value is “out of domain” because it is not an integer.

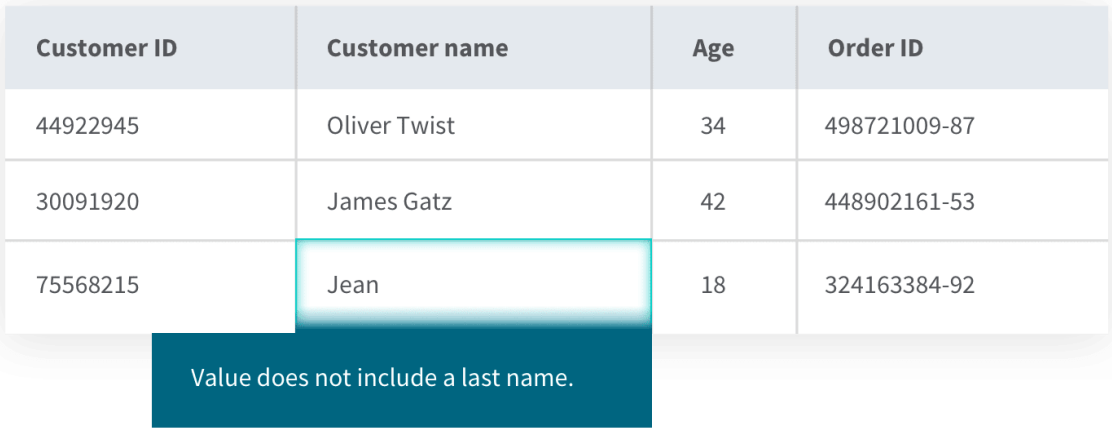
Obrázek 3. Příklad Doménové integrity [12]

- Referenční integrita – je to soubor pravidel a procedur, které slouží k zachování konzistence dat mezi dvěma tabulkami. Tato pravidla jsou integrována do struktury databáze a specifikují využití cizích klíčů k zajištění správnosti vkládaných dat a prevenci jejich duplikace.



Obrázek 4. Příklad Referenční identity [12]

- Uživatelem definovaná integrita – představuje nastavení pravidel a omezení pro data podle specifických potřeb uživatelů. Tento přístup se uplatňuje v případech, kdy standardní postupy zajištění integrity nedostačují k ochraně dat organizace, což umožňuje vytvoření pravidel reflektujících opatření pro zachování integrity dat dané organizace [14].



Obrázek 5. Příklad Uživatelem definované integrity [12]

2.1.2 Normalizace

Jedná se o metodu, jejímž cílem je snížení opakování dat v jedné nebo více relacích. Přítomnost nadbytečných dat v relaci může vést k problémům při vkládání, odstraňování a aktualizaci dat. Tento proces tedy usiluje o omezení opakovaných dat v relacích. K odstranění nebo minimalizaci nadbytečnosti v tabulkách databáze se využívají normální formy [15].

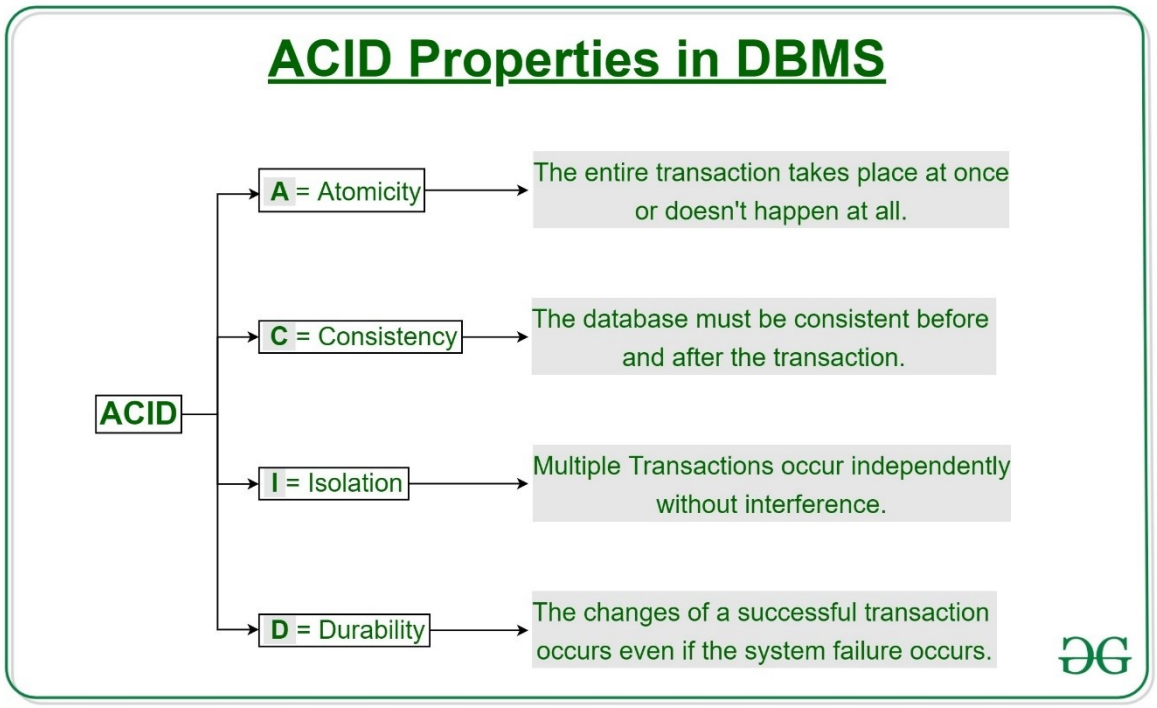
2.1.2.1 Normální formy

Normální formy jsou v databázovém designu uznávány jako fundamentální nástroje sloužící k dosažení optimalizované struktury databáze. Tyto formy, rozdělené do šesti hlavních kategorií, stanovují pravidla pro eliminaci redundance dat.

- **1NF** – Aby tabulka odpovídala první normální formě, musí obsahovat atomické hodnoty v buňkách, unikátní primární klíč pro identifikaci řádků bez duplicit a zajistit, že každý sloupec má pro každý řádek jednu hodnotu. Názvy sloupců by měli být unikátní [15] [16].
- **2NF** – Prvním krokem k dosažení 2NF je, aby tabulka splňovala všechny požadavky 1NF. Odstraňuje opakující se data tím, že požaduje, aby všechny atributy, které nejsou klíčové, závisely přímo na primárním klíči. Jinými slovy, každý sloupec by měl mít přímou vazbu k primárnímu klíči, nikoli k ostatním sloupcům [15][17].
- **3NF** – Musí splňovat pravidla 2NF. Následně vyžaduje úplnou funkční závislost neklíčových atributů výhradně na primárním klíči a zároveň požaduje, aby tyto atributy byly vzájemně nezávislé. Díky tomu dochází k odstranění tranzitivních závislostí mezi atributy [18].
- **BCNF** – Představuje vylepšenou verzi 3NF, kde se požaduje, aby všechny determinanty v tabulce byly kandidátní klíče. To znamená, že BCNF požaduje, aby všechny atributy, které nejsou částí klíče, závisely výhradně na kandidátním klíči [15].
- **4NF** – Posouvá principy BCNF ještě dále tím, že eliminuje přítomnost vícehodnotových závislostí v tabulce [15].
- **5NF** – Představuje vrcholný stupeň normalizačního procesu, který se zaměřuje na dělení tabulek na drobnější struktury s cílem odstranit duplicitní data a posílit integritu dat [15].

2.1.3 Transakce a ACID

Databázová transakce představuje soubor operací, který tvoří logický celek, často skládající se z více kroků na nižší úrovni. V případě, že jakýkoliv krok v rámci transakce neproběhne úspěšně, celá transakce je považována za neúspěšnou. Tento mechanismus se využívá pro vytváření, aktualizaci nebo získávání dat v databázi. [19] V kontextu relačních databází je nezbytné, aby transakce splňovaly čtyři základní atributy ACID. Tyto atributy jsou klíčové pro zajištění spolehlivého zpracování databázových transakcí [20].



Obrázek 6. ACID rozdělení [22]

2.1.3.1 Atomicita (Atomicity)

Garantuje, že všechny části transakce buď proběhnou celé do úspěšného konce, nebo se neprovedou vůbec. Tato vlastnost zajistí nedělitelnost transakce, brání přerušení běhu transakce a zamezí neúplným změnám v databázi [20].

2.1.3.2 Konzistence (Consitency)

Bez ohledu na výsledek jakékoliv transakce musí databáze zůstat v konzistentním stavu. To znamená, že pokud databáze byla před zahájením transakce v konzistentním stavu, musí tento stav udržet i po jejím dokončení [20].

2.1.3.3 Izolace (Isolation)

Všechny transakce pracují s izolovaným prostředím databáze, což znamená, že data využívaná v probíhajících transakcích nejsou přístupná pro úpravy prostřednictvím jiných transakcí, dokud nejsou tyto transakce plně dokončeny [20].

2.1.3.4 Trvalost (Durability)

Po úspěšném dokončení transakce se změny stávají trvalými a zůstanou v databázi zachovány, i v případě selhání databázového systému. To znamená, že pokud systém vykazuje chybu nebo dojde k jeho pádu předtím, než jsou data fyzicky zapsána na disk, budou po obnovení systému data aktualizována dle posledně dokončené transakce [20].

2.1.4 SQL

Structured Query Language je standardizovaný programovací jazyk určený pro extrakci, uspořádání, správu a manipulaci s daty, která jsou uložena v systémech relačních databází [21].

2.1.4.1 SQL dotazy

V této sekci bude pozornost věnována základním stavebním blokům pro tvorbu SQL dotazů. Pro manipulaci a správu dat uložených v relačních databázích jsou klíčové příkazy, jako jsou CREATE, SELECT, INSERT, UPDATE, DELETE, FROM, WHERE považovány za základ.

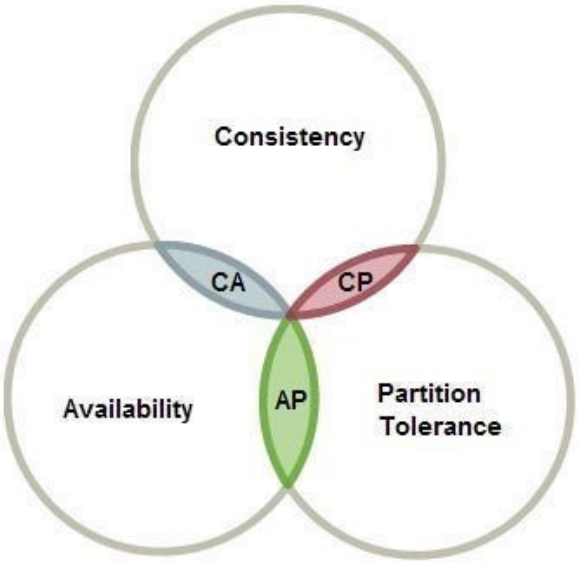
- **CREATE** – Vytvoření celé databáze, nové tabulky, views.
- **SELECT** – Vybírá data buď z databáze nebo přímo z tabulek.
- **INSERT** – Vkládání dat do tabulek.
- **UPDATE** – Aktualizuje existující tabulky.
- **DELETE** – Odstranění dat z existujících tabulek.
- **FROM** – Ukázání na tabulku odkud se mají vzít data.
- **WHERE** – Podobné jako FROM ale můžeme si zvolit požadavek podle kterého se daná data vezmou.

2.2 Základní principy NoSQL databází

NoSQL databáze jsou navrženy pro efektivní správu a zpracování diverzifikovaných datových struktur. Speciální pozornost bude věnována čtyřem hlavním typům NoSQL databází: klíč-hodnota, grafové, sloupcové a dokumentové. Každý typ nabízí unikátní vlastnosti a optimalizace pro různé druhy datových modelů a aplikačních požadavků. Flexibilita a škálovatelnost těchto systémů umožňuje aplikacím efektivně zvládat velké objemy dat a různorodé dotazovací požadavky. Odolnost vůči chybám je zajištěna distribuovanou architekturou a robustní replikací dat, což zvyšuje dostupnost a odolnost systému. Jejich neschematičnost a rozšiřitelnost NoSQL databází podporují dynamický vývoj aplikací a umožňují organizacím růst bez nutnosti přestavby celkové databázové infrastruktury. Detailněji se zaměříme na charakteristiky a použití každého typu databáze, aby bylo možné porozumět jejich specifickým přednostem a omezením.

2.2.1 CAP Teorém

CAP teorém, který byl formulován Carlem E. Brewerem v roce 2000, je považován za základní princip pro návrh distribuovaných systémů. Podle tohoto teorému nelze v distribuovaném systému současně zajistit všechny tři klíčové vlastnosti: konzistenci, dostupnost a odolnost k přerušení. Zatímco konzistence zajišťuje, že všechna data jsou v každém okamžiku stejná napříč všemi uzly, dostupnost zaručuje odpověď na každý dotaz, ať už úspěšný nebo s chybou. Odolnost vůči dělení sítě je zase zajištěna tak, že systém zůstává funkční i v případě výpadků části sítě. Tento teorém slouží jako důležitý rámec pro pochopení kompromisů při návrhu a provozu distribuovaných databází a systémů.



Obrázek 7. CAP Teorém [23]

- **Konzistence (Consistency)** - V distribuovaném systému konzistence znamená, že všechny uzly nebo repliky v systému mají ve stejnou dobu stejná data. Když klient čte data, obdrží nejnovější zápis nebo chybu. Jinými slovy, neexistuje žádná divergence v datech pozorovaných různými uzly [24].
- **Dostupnost (Avaibility)** - Dostupnost se týká schopnosti systému reagovat na požadavky klientů, a to i v případě selhání uzlů nebo síťových oddílů. Dostupný systém zajišťuje, že každý požadavek nakonec obdrží odpověď, i když nezaručuje, že odpověď obsahuje nejnovější data [24].
- **Odolnost k přerušení (Partition Tolerance)** - Tolerance oddílů se zabývá schopností systému pokračovat v činnosti, i když se vyskytnou síťové oddíly. Síťové oddíly mohou způsobit, že uzly ztratí vzájemný kontakt, což znesnadní komunikaci a synchronizaci [24].

CAP teorém jasně stanoví, že distribuovaný systém nemůže nikdy zaručit současné splnění všech tří klíčových vlastností: konzistence, dostupnosti a odolnosti k přerušení. Místo toho může systém vždy splnit pouze dvě z těchto vlastností současně. Tento princip nutí architekty a vývojáře distribuovaných systémů k zásadnímu rozhodování, které z vlastností upřednostnit v závislosti na specifických požadavcích a očekáváních aplikace. Výběr mezi konzistencí, dostupností a odolností vůči dělení sítě má zásadní vliv na celkový design a chování systému. Toto rozhodnutí ovlivňuje, jak systém zvládá výpadky a jak efektivně může zpracovávat dotazy a transakce v reálném čase.

Význam tohoto teorému spočívá v poskytování jasného rámce, který pomáhá pochopit omezení a možné kompromisy v provozu distribuovaných databázových systémů.

2.2.1.1 Konzistence + Dostupnost

Zajišťuje konzistenci a dostupnost napříč všemi uzly. Nemůže to však udělat, pokud existuje oddíl mezi libovolnými dvěma uzly v systému, a proto nemůže zajistit odolnost proti chybám [25].

2.2.1.2 Dostupnost + Odolnost k přerušení

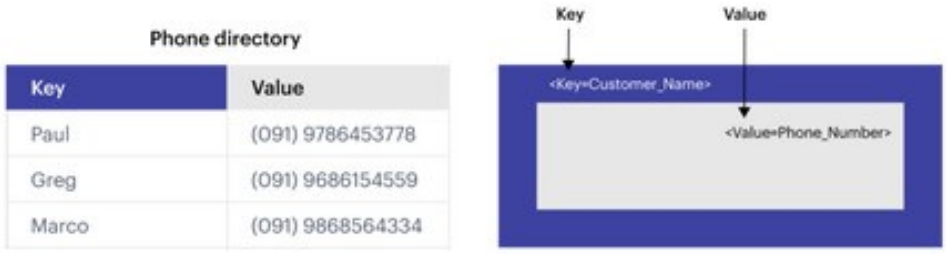
Poskytuje dostupnost a toleranci oddílů na úkor konzistence. Když dojde k oddělení, všechny uzly zůstanou dostupné, ale ty na špatném konci oddílu mohou vrátit starší verzi dat než ostatní (Když je oddíl vyřešen, databáze AP obvykle znovu synchronizují uzly, aby se opravily všechny nekonzistence v systému.) [25].

2.2.1.3 Konzistence + Odolnost k přerušení

Zajišťuje konzistenci a toleranci oddílů na úkor dostupnosti. Když dojde k rozdělení mezi libovolnými dvěma uzly, systém musí vypnout nekonzistentní uzel (tj. znepřístupnit jej), dokud nebude oddíl vyřešen [25].

2.2.2 Klíč – hodnota databáze

Databáze klíč-hodnota je typ softwarového programu pro ukládání dat, který ukládá data jako sadu jedinečných identifikátorů, z nichž každý má přidruženou hodnotu. Toto párování dat je známé jako „pár klíč–hodnota“ [26].



Obrázek 8. Příklad Klíč – hodnota databáze [27]

2.2.2.1 Pár klíč – hodnota

Pár klíč–hodnota je datový typ, který zahrnuje dva kusy dat, které mají sadu přidružených hodnot a skupinu identifikátorů klíčů. V rámci páru klíč–hodnota existují dva související datové prvky. První je konstanta používaná k definování datové sady. Druhým je hodnota, což je proměnná patřící do souboru dat. Klíčem může být například auto, zatímco hodnotami může být barva vozu, model nebo značka [28].

2.2.2.2 Výhody

Databáze založené na párech klíč–hodnota se vyznačují významnými přednostmi v oblastech škálovatelnosti, rychlosti a flexibility. Škálovatelnost je podporována snadností, s jakou se páry dat zpracovávají a ukládají, umožňující rychlé skenování a efektivní manipulaci s velkými objemy dat bez ztráty výkonu. Rychlost je zajištěna jednoduchostí struktury párů klíč-hodnota, což umožňuje rychlé čtení a zápis dat. Flexibilita těchto databází spočívá v jejich adaptabilitě na změny struktury databáze, což je neocenitelné ve scénářích, kde je potřeba dynamicky reagovat na požadavky aplikace. Navíc absence požadavku na speciální zástupné symboly pro volitelné hodnoty, jako je "null", zvyšuje efektivitu úložiště a snižuje složitost [28].

2.2.2.3 Nevýhody

Databáze založené na párech klíč–hodnota mají několik omezení, která mohou komplikovat pokročilé využití v některých aplikacích. Mezi hlavní nevýhody patří omezené možnosti dotazování, které ztěžují provádění složitých dotazů a spojení. Jednoduchý datový model těchto databází může být nedostatečný pro složité datové struktury a vztahy, což může komplikovat datové modelování. Dalším významným omezením je často omezená podpora transakcí ACID, což může vést k problémům s konzistencí a integritou dat. Navíc absence pokročilých funkcí, jako je fulltextové vyhledávání nebo geoprostorové indexování, může omezit širší použití těchto databází v různorodých aplikačních scénářích [29].

2.2.3 Grafové databáze

Databáze grafů je specializovaná, jednoúčelová platforma používaná k vytváření a manipulaci s daty asociativní a kontextové povahy. Hlavním konceptem grafového databázového systému je vztah. Vztahy jsou definovány jako prvotřídní občané – to znamená, že vše, co můžete dělat se všemi ostatními prvky, lze udělat se vztahem. Data jsou spolu spojena v grafu pro uložení kolekce uzlů a hran, kde hrany představují vztah mezi uzly [29].

2.2.3.1 Základní komponenty

Základními komponentami grafových databází jsou uzly, hrany a vlastnosti. Tyto elementy slouží k reprezentaci objektů a jejich vzájemných vztahů, umožňující detailní mapování a analýzu komplexních sítí interakcí.

- **Uzel** – Objekty nebo instance jsou reprezentovány pomocí uzlu. Koncepčně jsou uzly ekvivalentem řádku v relační databázi a fungují jako vrchol v grafu. Seskupení uzlu se jednoduše provede použitím štítku na každý člen [30].
- **Hrany** – Dalším názvem pro hrany v grafu jsou vztahy. Vztahy se vždy skládají z počátečního uzlu, koncového uzlu, typu a směru. Vytvářejí datové vzorce popisem vztahů mezi rodiči a dětmi, akcí, vlastnictví a podobně [30].
- **Vlastnosti** – Vlastnosti jsou informace spojené s uzly [30].

2.2.3.2 Výhody

Grafové databáze nabízejí mnoho výhod, díky kterým jsou vhodné pro aplikace vyžadující komplexní analýzu vztahů. Jejich struktury jsou agilní a flexibilní, což umožňuje snadné přizpůsobení datových modelů podle měnících se požadavků. Reprezentace vztahů mezi entitami je v těchto databázích explicitně vyjádřena, což zvyšuje přehlednost a srozumitelnost dat. Dále grafové databáze umožňují provádění dotazů, které vydávají výsledky v reálném čase, přičemž rychlost zpracování závisí přímo na počtu vztahů mezi uzly, což je ideální pro dynamické a náročné datové aplikace [31].

2.2.3.3 Nevýhody

Grafové databáze mají několik nevýhod, které mohou ovlivnit jejich využití v určitých situacích. Jednou z klíčových omezení je absence standardizovaného dotazovacího jazyka, což znamená, že použitý jazyk závisí na konkrétní platformě, což může komplikovat přenositelnost aplikací a integraci s jinými systémy. Dále nejsou grafy vhodné pro transakční systémy, jelikož mohou mít omezenou schopnost zvládat velké množství transakčních operací s vysokými nároky na konzistenci a rychlost. Navíc malá uživatelská základna může ztížit hledání podpory a sdílení řešení při technických problémech, což může být výzvou pro nové uživatele nebo v situacích, kdy je potřeba rychle řešit vzniklé problémy [31].

2.2.4 Sloupcové databáze

Sloupcová databáze, známá také jako sloupcově orientovaná databáze, je typem systému správy databází (DBMS), který ukládá data ve sloupcích společně na disk. To umožňuje rychlejší dotazy pro analýzu dat, která obecně zahrnuje filtrování a agregaci sloupců tabulky [31].

2.2.4.1 Výhody

Sloupcové databáze nabízejí řadu výhod, které je činí vhodnými pro specifické typy dotazů a zpracování dat. Jsou ideální pro situace, kdy dotazy zahrnují pouze několik sloupců, což umožňuje efektivní načítání a zpracování pouze potřebných dat místo celých řádků. Tato specializace je obzvláště užitečná při provádění agregací na obrovském množství dat, kde sloupcové databáze mohou výrazně zvýšit rychlost a snížit zatížení systému. Dále sloupcové databáze umožňují efektivní kompresi dat, což vede k lepší optimalizaci úložiště a rychlejšímu přístupu k datům, čímž se zvyšuje celková výkonnost systému [32].

2.2.4.2 Nevýhody

Sloupcové databáze, ačkoli jsou efektivní v některých scénářích, mají také několik nevýhod, které omezují jejich univerzální použití. Postupné načítání dat může být neefektivní, zejména když je potřeba zpracovat dotazy, které se týkají pouze několika řádků, protože celé sloupce musí být načteny i když jsou potřeba jen malé části dat. Tato vlastnost dělá sloupcové databáze méně vhodné pro aplikace zaměřené na Online Transaction Processing, kde je důležitá rychlost a efektivita při zpracování velkého množství malých transakcí. Tyto faktory mohou významně ovlivnit výkon systému v některých operativních prostředích, kde je prioritou rychlá reakce na dotazy zaměřené na specifické řádky dat [33].

2.2.5 Dokumentové databáze

Dokumentové databáze jsou typem NoSQL databázového systému, který je navržen pro efektivní správu polostrukturovaných dat. Data jsou ukládána ve formě dokumentů, což zajišťuje flexibilitu a dynamiku ve struktuře dat. Tento typ databáze usnadňuje integraci a škálování aplikací díky schopnosti zpracovávat různorodá data bez předdefinovaného schématu. Vzhledem k jejich agilnosti a jednoduchosti jsou dokumentové databáze vhodné pro rychlý vývoj moderních webových a mobilních aplikací. Podrobnější informace o dokumentech a jejich využití budou poskytnuty v následujících částech.

Obrázek 9. Příklad dokumentové databáze [34]

2.2.5.1 Dokumenty

Záznam v databázi dokumentů je označován jako dokument. Obvykle dokument obsahuje informace o jednom objektu a veškerá jeho přidružená metadata. Data v dokumentech jsou ukládána ve formě pole-hodnota. Hodnoty mohou být různých typů a struktur, včetně řetězců, čísel, dat, polí nebo objektů. Formáty pro ukládání dokumentů zahrnují JSON, BSON a XML [34].

2.2.5.2 Kolekce

Kolekce jsou skupinou dokumentů, kde dokumenty mají podobný obsah. Vzhledem k flexibilnímu schématu databází dokumentů nemusí všechny dokumenty v kolekci obsahovat stejná pole. Je třeba poznamenat, že některé databáze dokumentů umožňují validaci schématu a umožňují uzamknutí schématu podle potřeby [34].

2.2.5.3 CRUD operace

Vývojáři obvykle mohou v dokumentových databázích provádět operace CRUD (vytváření, čtení, aktualizace a odstraňování) díky API nebo dotazovacímu jazyku, které jsou k dispozici [34].

- **Vytvoření (Create)** - V databázi lze vytvářet dokumenty. Každý dokument má jedinečný identifikátor.
- **Čtení (Read)** - Dokumenty jsou schopny být čteny z databáze. Vývojáři mají možnost dotazovat se na dokumenty pomocí jejich jedinečných identifikátorů nebo hodnot polí pomocí API nebo dotazovacího jazyka. K zvýšení výkonu čtení je možné do databáze přidat indexy [34].
- **Aktualizace (Update)** - Stávající dokumenty lze aktualizovat – buď celé, nebo jejich části [34].
- **Smazat (Delete)** – Smazání dokumentů z databáze.

2.2.5.4 Výhody

Výrazná flexibilita je nabízena dokumentovými databázemi, což umožňuje rychlé přizpůsobení struktury dat bez nutnosti synchronizace nebo složitých migračních procesů, což je považováno za ideální pro aplikace s rychle se vyvíjející logikou. Horizontální škálování je podporováno těmito systémy, což má za následek snížení nákladů a zjednodušení správy dat, jelikož operace na dokumentech obvykle vyžadují méně koordinace. Navíc, agilní přístup bez pevného schématu umožňuje rychlé nasazení a změny databázových modelů v reálném čase [35].

2.2.5.5 Nevýhody

Omezení kontroly konzistence jsou vnímána jako výzva při udržování datové konzistence napříč distribuovanými systémy dokumentových databází. Nedostatečná izolace slabých míst v atomičnosti, které by měly být nedělitelné, může vést k vyššímu riziku datových nesrovnalostí při selhání transakce. Co se týče bezpečnosti, flexibilita a otevřená struktura dokumentových databází jsou vnímány jako možnost pro vznik bezpečnostních slabostí, pokud nejsou správně zabezpečeny, což vyžaduje zvýšenou pozornost při implementaci ochranných opatření [36].

2.3 Komparace SQL a NoSQL databází

Výběr SQL a NoSQL je klíčový pro zvolení vhodné technologie v závislosti na potřebách projektu. Databáze jsou podrobněji probrány v předchozí kapitolách [viz. kapitola 2.1 – „Základní principy relačních databázích“, viz. kapitola 2.2 – „Základní principy NoSQL databází“].

1. **Flexibilita a Schéma**

SQL databáze vyžadují předem definované schéma pro ukládání dat, což zajišťuje konzistenci, ale může omezit rychlost adaptace na změny. NoSQL databáze s dynamickým schématem umožňují rychlé přizpůsobení datových modelů, což je výhodné pro projekty s nestrukturovanými daty.

2. **Škálovatelnost**

SQL systémy jsou uzpůsobeny pro vertikální škálování, zatímco NoSQL databáze jsou navrženy pro snadné horizontální škálování pomocí clusterů, což je efektivní pro zpracování velkého objemu dat.

3. **Výkon a Optimalizace Dotazů**

SQL databáze podporují složité dotazy s vysokou přesností díky jazyku SQL, který umožňuje komplexní analýzy. NoSQL databáze poskytují různé modely dotazování, které jsou optimalizované pro rychlé operace specifické pro typ ukládaných dat.

4. **Transakční Integrita**

SQL databáze poskytují robustní transakční podporu s vlastnostmi ACID, zásadní pro aplikace vyžadující vysokou úroveň integrity dat. U NoSQL databází může být transakční podpora omezenější, což může být vyváženo jejich vyšším výkonem a lepší škálovatelností.

II. PRAKTICKÁ ČÁST

3 NÁVRCH A IMPLEMENTACE BENCHMARKOVÉ APLIKACE

V této kapitole je představen návrh a implementace benchmarkové aplikace, která je zaměřena na hodnocení výkonnosti relačních a NoSQL databází. Aplikací je poskytován robustní a flexibilní nástroj pro systematické testování a analýzu databázových systémů, umožňující identifikaci klíčových výkonnostních metrik a jejich kvantitativní hodnocení.

Aplikace byla navržena tak, aby byla schopna efektivně generovat, spravovat a analyzovat uživatelská data, což je klíčové pro posouzení škálovatelnosti, rychlosti a efektivity databázových operací. Díky modulární architektuře a intuitivnímu uživatelskému rozhraní je umožněno rychlé nastavení a spouštění různých testovacích scénářů, což zahrnuje jak komplexní zátěžové testy, tak specifické dotazy na výkon.

Další klíčové funkce, jako je možnost hromadného generování a mazání uživatelů nebo selektivní aktualizace jejich stavů podle geografické lokace, podporují vysokou úroveň přizpůsobitelnosti, která je nezbytná pro validní a přesné měření výkonnosti v dynamickém prostředí moderních databázových technologií.

3.1 Výběr technologií pro benchmarkovou aplikaci

Pro vývoj benchmarkové aplikace bylo rozhodnuto o využití technologií, které podporují rychlý vývoj a efektivní údržbu, a zároveň umožňují efektivní testování a analýzu databází. V této části jsou popsány technologie, které byly vybrány pro realizaci projektu, a důvody, proč bylo rozhodnuto o jejich použití.

3.1.1 PyCharm

PyCharm Professional, IDE od společnosti JetBrains, bylo vybráno pro jeho pokročilé funkce a podporu programovacího jazyka Python, který byl zvolen jako hlavní jazyk pro aplikaci. Profesionální verze PyCharmu nabízí komplexní nástroje pro ladění a testování kódu, které značně zjednodušují vývoj složitých aplikací. Jednou z klíčových výhod je možnost přímého napojení na různé typy databází, což umožňuje vývojářům snadno se připojit, spravovat a interagovat s databázemi přímo z IDE. Tato integrace je neocenitelná pro aplikace vyžadující intenzivní práci s daty.

Důvody výběru PyCharm Professional:

- **Efektivní správa projektu:** Nástroje pro správu kódu a projektů zvyšují produktivitu vývoje.

- **Podpora databází:** Přímé napojení na databáze jako MySQL, PostgreSQL, Oracle a další usnadňuje testování a manipulaci s daty.
- **Rozšířené ladění a testování:** Rozsáhlé možnosti ladění a testování kódu podporují vysokou kvalitu vývoje.

3.1.2 Flask

Flask, mikro web framework pro Python, byl vybrán pro svou jednoduchost a flexibilitu. Jeho minimalistická povaha a možnost rozšíření jsou ideální pro rychlý vývoj lehkých web aplikací. Flask umožňuje snadné nastavení web serveru a efektivní routování požadavků, což je klíčové pro testovací aplikace vyžadující dynamickou interakci s uživateli. Dobře dokumentované vlastnosti a rozsáhlá komunitní podpora usnadňují řešení potenciálních vývojových problémů.

Důvody výběru Flask:

- **Rychlé prototypování:** Umožňuje rychlé sestavování a iteraci web aplikací.
- **Flexibilita a rozšiřitelnost:** Minimalistický základ frameworku lze rozšířit dle potřeb aplikace.
- **Silná komunitní podpora:** Aktivní komunita a bohatá dokumentace poskytují důležité zdroje pro vývojáře.

3.1.3 Databázové systémy

Pro testování výkonnosti různých typů databázových systémů byly vybrány databáze MySQL, MongoDB, Redis a Neo4j, s tím, že kromě MySQL byla pro všechny ostatní databáze využita cloudová uložiště. Toto řešení umožňuje lepší škálovatelnost a flexibilitu při testování v různých prostředích a zátěžových scénářích.

3.1.3.1 MySQL

MySQL, oblíbený relační databázový systém, byl vybrán z důvodu jeho širokého nasazení v průmyslu a vynikající podpory pro komplexní transakční systémy. Tato databáze poskytuje pevnou strukturu a silné konzistenční záruky, což je klíčové pro aplikace, kde je důležitá přesnost a integrita dat.

Důvody výběru MySQL:

- **Podpora složitých dotazů:** Výborná podpora SQL jazyka umožňuje provádění komplexních dotazů, což je nezbytné pro důkladné testování výkonnosti v relačních databázích [38].
- **Transakční integrita:** Podpora transakcí s ACID vlastnostmi zajišťuje, že data zůstávají konzistentní i přes různé operace, což je důležité pro simulace reálného nasazení.

3.1.3.2 MongoDB

MongoDB, široce používaná NoSQL dokumentově orientovaná databáze, byla vybrána kvůli své schopnosti rychle zpracovávat a ukládat velké množství nestrukturovaných dat v cloudovém prostředí. Je ideální pro testování výkonu v aplikacích, které vyžadují flexibilitu v datových modelech a rychlou evoluci datové struktury.

Důvody výběru MongoDB:

- **Flexibilita dokumentů:** Umožňuje snadnou modifikaci a rozšiřování datových struktur bez nutnosti předem definovaných schémat.
- **Cloudová škálovatelnost:** Využití cloudového uložiště zvyšuje flexibilitu a škálovatelnost, což umožňuje efektivně rozložit zátěž při velkém objemu dat.

3.1.3.3 Redis

Redis byl vybrán pro jeho schopnost poskytnout extrémní výkon při práci s daty uloženými v operační paměti, což je zvláště výhodné v cloudovém prostředí pro scénáře vyžadující rychlý přístup k datům a vysokou dostupnost.

Důvody výběru Redis:

- **Rychlost přístupu k datům:** Data uložená v paměti umožňují bleskurychlý přístup, což je ideální pro real-time aplikace v cloudovém prostředí.
- **Vysoká dostupnost a snadná škálovatelnost:** Cloudové služby zvyšují dostupnost a škálovatelnost Redis databází, což je kritické pro vysoce dostupné systémy.

3.1.3.4 Neo4j

Neo4j, databáze orientovaná na grafy, byla zařazena do výběru kvůli své schopnosti efektivně zpracovávat složité dotazy na vztahy mezi datovými objekty v cloudovém prostředí. Je to klíčové pro testování výkonu v aplikacích, které analyzují sítě vztahů a propojení.

Důvody výběru Neo4j:

- **Efektivní zpracování grafů:** Specializace na operace s grafy umožňuje rychlé provádění složitých dotazů na vztahy.
- **Cloudové prostředí:** Využití cloudových technologií umožňuje snadnější správu a škálování grafických databází, což je ideální pro komplexní datové analýzy.

3.2 Návrh aplikace

V této části je popsán návrh aplikace, která byla vytvořena pro osobní testování a provádění benchmarků výkonnosti databází. Celkový návrh byl vypracován s využitím softwaru Enterprise Architect, který umožnil detailní modelování funkčních a nefunkčních požadavků, stejně jako vytvoření wireframe modelů všech aspektů aplikace.

Aplikace byla speciálně navržena pro generování, aktualizaci a mazání testovacích dat, což umožňuje komplexní testování a hodnocení výkonnosti různých databázových systémů. Návrh aplikace klade důraz na škálovatelnost a flexibilitu, aby byla zajištěna efektivita a snadná adaptabilita při přidávání nových funkcí nebo integraci nových technologií.

Použití Enterprise Architect zaručilo vysokou míru přesnosti a konzistence ve fázi návrhu, což výrazně usnadňuje implementaci a testování aplikace. Dále modelování v tomto nástroji poskytlo pevný základ pro budoucí rozvoj a údržbu aplikace, umožňuje snadné provádění změn a zajišťuje, že všechny komponenty systému jsou dobře dokumentovány a vzájemně kompatibilní.

3.2.1 Enterprise Architect

EA od společnosti Sparx Systems je rozšířený nástroj pro modelování softwaru, který podporuje UML a další metody pro vizualizaci a dokumentaci informačních systémů. Software je využíván pro detailní modelování aplikací, včetně analýzy požadavků, specifikace architektury a plánování projektů.

EA umožňuje uživatelům vytvářet různé typy diagramů, jako jsou diagramy tříd, sekvenční diagramy a diagramy použití, což zlepšuje srozumitelnost projektů a usnadňuje komunikaci mezi zúčastněnými stranami. Nástroj také integruje funkce pro generování dokumentace a reportů, což umožňuje efektivní sdílení a revizi projektových materiálů.

Díky své flexibilitě a široké podpoře pro integraci s jinými technologiemi je EA ideální pro správu komplexních softwarových projektů v různých průmyslových odvětvích.

3.2.2 Funkční požadavky

V této části jsou specifikovány klíčové funkční požadavky pro aplikaci, která byla navržena pro interní provádění benchmarků výkonnosti databází. Každý z těchto požadavků detailně popisuje očekávané chování aplikace, která je primárně určena pro výzkumné a vývojové účely.

Aktualizace uživatelů:

- **Požadavek:** Aplikace musí umožňovat aktualizaci stavu uživatelských profilů. Mělo by být možné aplikovat nový stav na všechny profily nebo selektivně na profily filtrované podle země.
- **Vstupy:** Vybraný stav, volitelně země.
- **Výstupy:** Potvrzení o úspěšné aktualizaci stavu.

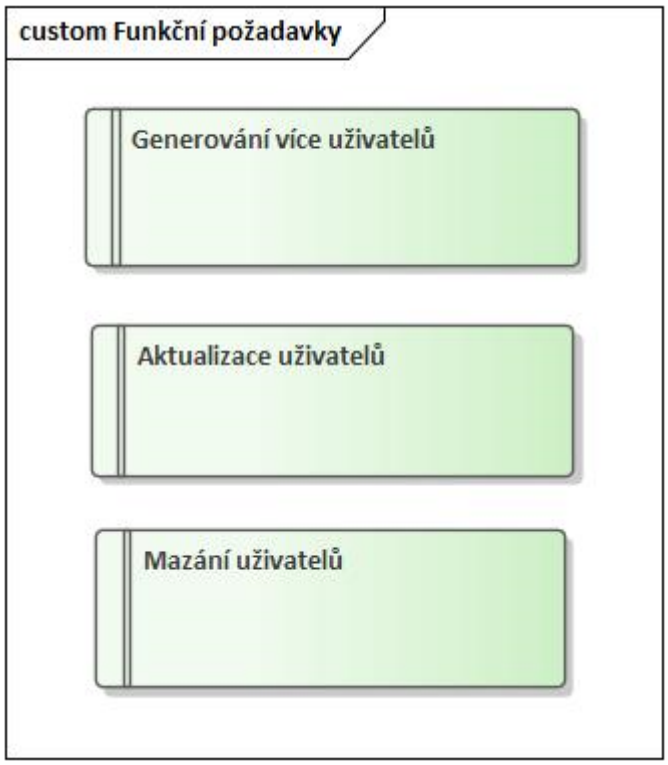
Generování více uživatelů:

- **Požadavek:** Aplikace musí poskytovat funkci pro hromadné generování uživatelských profilů. Měla by umožnit definovat počet profilů k vygenerování pro simulaci zátěže a testování škálovatelnosti databází.
- **Vstupy:** Počet uživatelů k vygenerování.
- **Výstupy:** Seznam vygenerovaných uživatelských profilů s přidělenými identifikátory a dalšími atributy.

Mazání uživatelů:

- **Požadavek:** Aplikace musí umožňovat mazání uživatelských profilů. Měla by být schopna odstranit všechny profily nebo provádět selektivní mazání profilů podle iniciál jména a příjmení.
- **Vstupy:** Volba mazání všech uživatelů nebo iniciály jména a příjmení.
- **Výstupy:** Potvrzení o úspěšném odstranění vybraných profilů.

Tyto požadavky tvoří základní operace, které jsou nezbytné pro účely testování a vývoje aplikace. Implementace těchto funkčních požadavků zajistí, že aplikace bude schopna efektivně spravovat a manipulovat s uživatelskými daty v rámci testování výkonnosti databázových systémů.



Obrázek 10. Návrh funkčních požadavků v Enterprise Architect

3.2.3 Nefunkční požadavky

Níže jsou definovány základní nefunkční požadavky pro aplikaci, navržené speciálně pro interní potřeby projektu. Tyto požadavky se zaměřují na aspekty zajišťující stabilitu, výkon a efektivitu systému pro účely benchmarkingu databázových technologií.

Doba Odezvy:

- Požadavek: Musí být zajištěno, že doba odezvy aplikace je minimalizována. Odpověď na všechny operace by měla být poskytnuta do stanoveného časového limitu X sekund, aby bylo možné efektivně provádět testy bez zbytečných prodlev.

Škálovatelnost:

- Požadavek: Aplikace musí být schopna škálovat pro podporu rostoucího množství testovacích dat a simulací. Systém musí efektivně zvládat zvýšenou zátěž, která může vzniknout během intenzivních testovacích období.

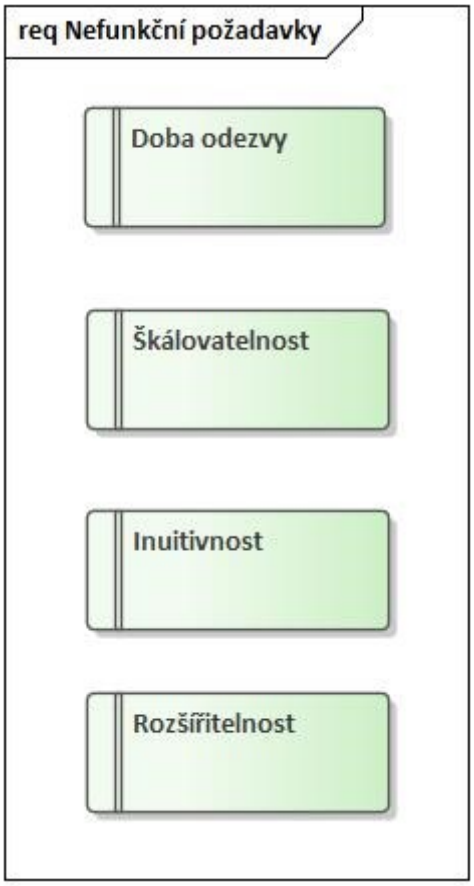
Intuitivnost:

- Požadavek: Uživatelské rozhraní aplikace musí být jednoduché a přímé, aby bylo možné snadno spravovat testovací procedury a data. Musí být zajištěno, že navigace a ovládání aplikace jsou intuitivní, což umožňuje rychlé a efektivní provádění benchmarků bez nutnosti rozsáhlého zaškolení.

Rozšiřitelnost:

- Požadavek: Aplikace musí být navržena s ohledem na budoucí rozvoj a potenciální potřebu integrace nových technologií nebo testovacích scénářů. Systém by měl umožňovat snadné rozšíření o nové funkce a moduly, což usnadňuje adaptaci na měnící se testovací požadavky.

Tyto požadavky byly specifikovány s cílem zajistit, že aplikace zůstane výkonná, stabilní a efektivní v podmínkách zaměřených výhradně na testování a vývoj. Zajištění splnění těchto požadavků je klíčové pro dosažení účelnosti a účinnosti benchmarkové aplikace v průběhu celého vývojového cyklu.



Obrázek 11. Návrh nefunkčních požadavků v Enterprise Architect

3.2.4 Drátěný model (Wireframe)

3.2.4.1 Uživatelské rozhraní

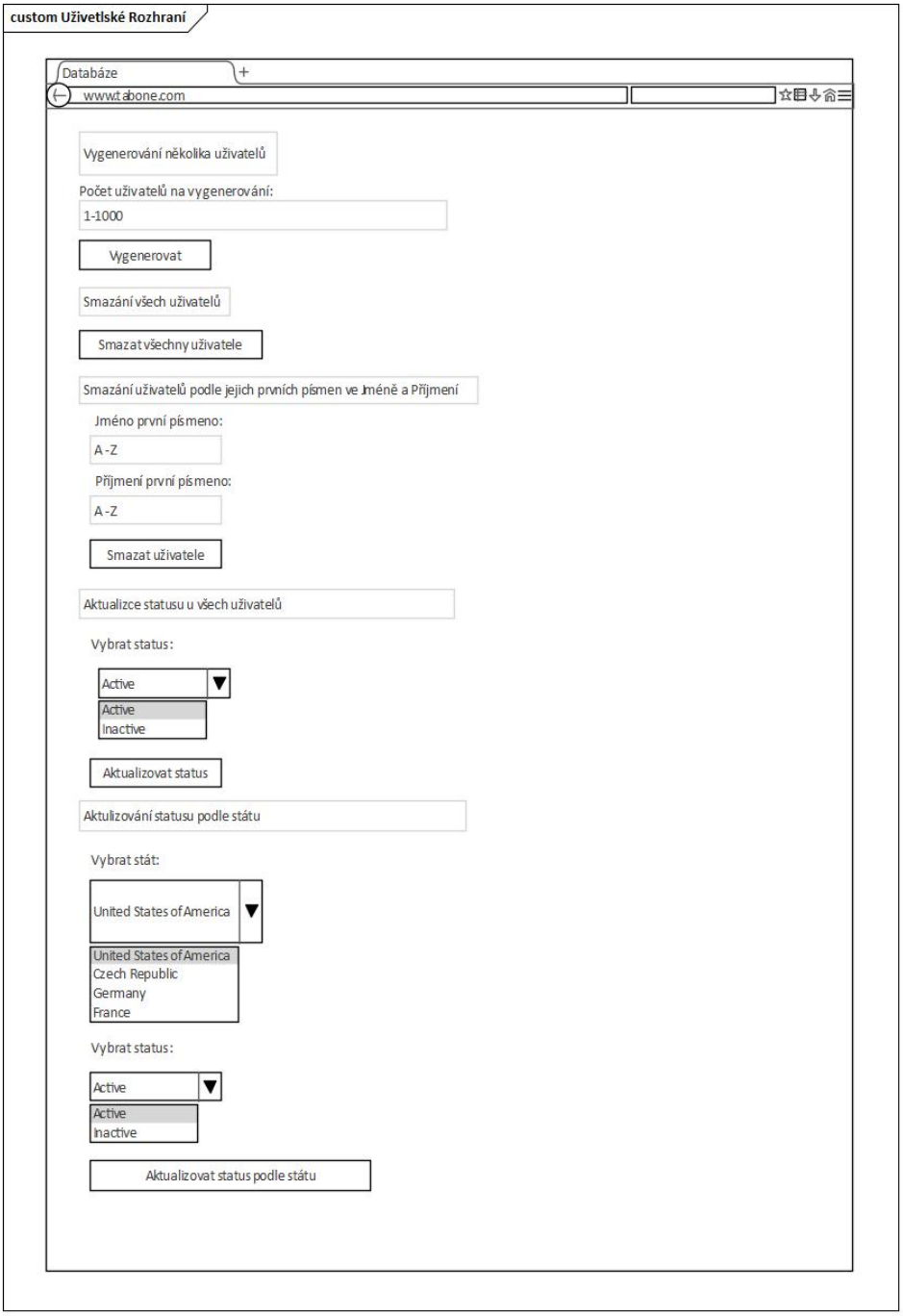
V této části je prezentován drátěný model uživatelského rozhraní aplikace, který byl vytvořen v nástroji Enterprise Architect. Tento model vizualizuje strukturu a layout aplikace, která je určena pro interní účely testování a benchmarking databází. Model zobrazuje základní uživatelské operace a způsob, jakým jsou data a příkazy spravovány v rámci aplikace. Tento model slouží jako základní krok k implementaci uživatelského rozhraní aplikace, zajišťuje, že všechny uživatelské interakce jsou logicky uspořádány a efektivně zpracovány.

Struktura Rozhraní:

1. **Generování více uživatelů:**
    - Sekce obsahuje pole pro zadání počtu uživatelů, kteří mají být vygenerováni. Uživatelé mohou specifikovat počet od 1 do 1000, což umožňuje flexibilitu při testování škálovatelnosti databází.
2. **Mazání uživatelů:**
    - Nabízí se dvě možnosti mazání: mazání všech uživatelů a mazání uživatelů podle iniciál jména a příjmení. Tyto funkce umožňují efektivní správu testovacích dat.
3. **Aktualizace stavu uživatelů:**
    - Uživatelé mohou aktualizovat stav všech profilů nebo provádět selektivní aktualizaci stavu podle země. Rozbalovací seznamy nabízejí možnosti pro výběr státu a nového stavu, což poskytuje přizpůsobivý přístup k správě stavů uživatelských profilů.

Vizuální Aspekty:

- **Layout:** Rozhraní je organizováno do jasně definovaných sekcí, každá s vlastními ovládacími prvky, což zvyšuje přehlednost a uživatelskou přívětivost.
- **Přístupnost:** Elementy ovládání jsou navrženy s ohledem na jednoduchost a intuitivnost, což usnadňuje navigaci a minimalizuje potenciální uživatelské chyby.



Obrázek 12. Návrh drátěného modelu v Enterprise Architect pro uživatelské rozhraní

3.2.4.2 Výsledky měření

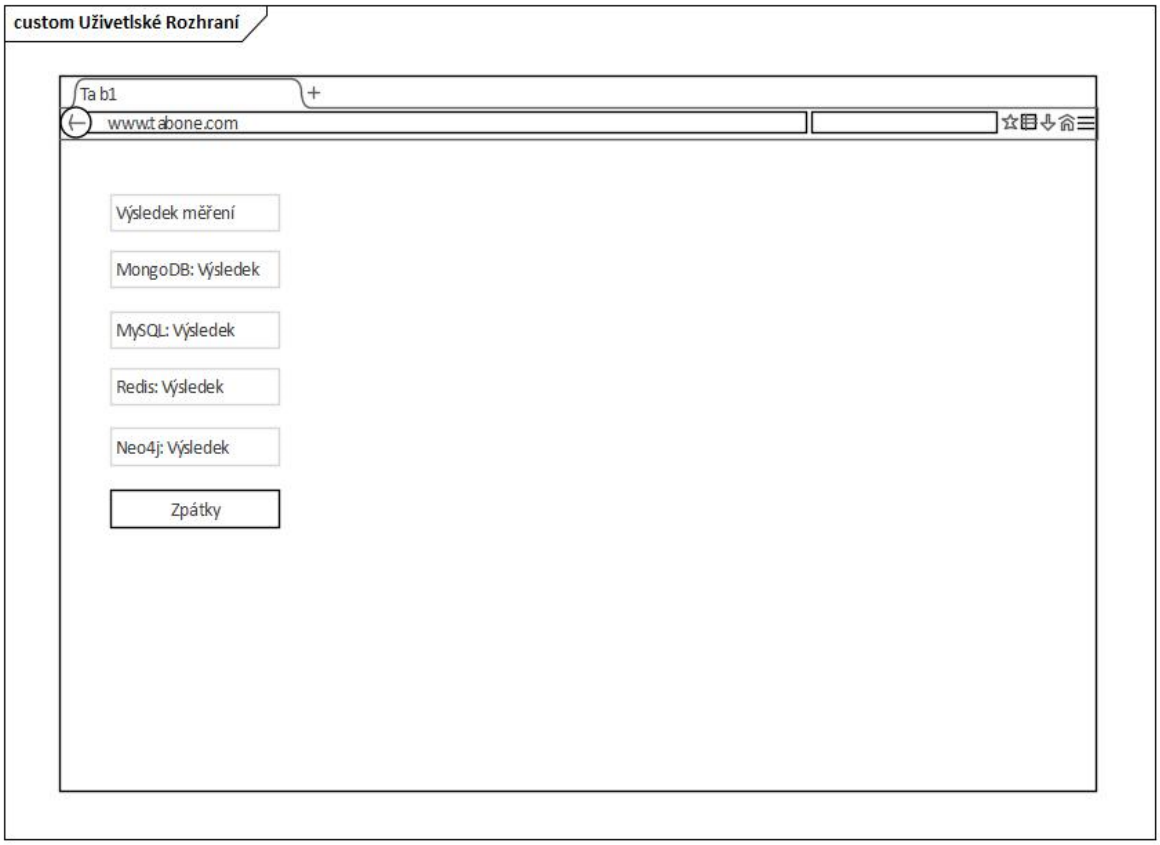
Na této stránce drátěného modelu je zobrazena sekce výsledků měření, speciálně navržená pro interní použití a analýzu výkonnosti databází. Stránka je strukturována tak, aby poskytovala přehledný a efektivní způsob prezentace časových metrik jednotlivých testů.

Struktura stránky:

- **Sekce Výsledků Měření:**
  - Pro každou databázi (MongoDB, MySQL, Redis, Neo4j) jsou přímo zobrazeny časy potřebné pro provedení testů, jak celkově, tak na jednoho uživatele. Toto přímé zobrazení umožňuje rychlé vyhodnocení a porovnání efektivity databázových systémů.
  - Výsledky jsou uvedeny v jasném a stručném formátu, což umožňuje okamžitou analýzu a vývojářům nabízí rychlý přehled o výkonu testovaných technologií.

Design a Usability:

- **Přehlednost:** Rozhraní je navrženo pro maximální přehlednost, což minimalizuje potřebu zdlouhavého procházení a umožňuje efektivní interpretaci dat.
- **Přístupnost:** Prvky ovládání a navigace jsou intuitivní, což zajišťuje snadnou orientaci v aplikaci bez zbytečných komplikací.



Obrázek 13. Návrh drátěného modelu v Enterprise Architect pro výsledky měření

3.3 Implementace benchmarkové aplikace

V této podkapitole je podrobně popsána implementace benchmarkové aplikace, navržené výhradně pro interní potřeby měření a porovnávání výkonnosti různých databázových systémů. Hlavním cílem této kapitoly je poskytnout ucelený přehled o technických aspektech realizace projektu, od architektury aplikace po konkrétní implementační kroky.

3.3.1 Frontend

Frontend benchmarkové aplikace byl jednoduše navržen a implementován s cílem podporovat testování výkonnosti databází. Struktura a design byly minimalizovány, aby se zvýraznila funkcionalita a zefektivnilo používání aplikace.

Použité technologie:

Pro implementaci byly vybrány HTML a CSS, což zajišťuje dostatečnou flexibilitu a rychlost pro potřeby testování:

- **HTML:** Zajišťuje základní strukturu aplikace, poskytuje framework pro umístění funkcí, které již byly popsány ve fázi návrhu drátěného modelu [viz. podkapitola 3.2.4 – „Drátěný model (Wireframe)“].
- **CSS:** Použit pro aplikaci vizuálních stylů, které podporují čistotu a přehlednost uživatelského rozhraní, čímž se minimalizují vizuální rušivé prvky a zvyšuje se uživatelská přívětivost.

```
<h1>Vygenerování několika uživatelů</h1>
    <form action="/generate-users" method="post">
        <label for="user_count">Počet uživatelů pro vygenerování:</label>
        <input type="number" id="user_count" name="user_count" min="1"
max="1000" required>
        <input type="submit" value="Vygenerovat uživatele">
    </form>
```

Zdrojový kód 1. Příklad HTML pro generování uživatelů

3.3.2 Backend

V rámci tohoto projektu byl vyvinut backendový systém s cílem zajištění robustní, bezpečné a škálovatelné infrastruktury pro správu dat a procesů. Pro realizaci těchto cílů byly vybrány technologie a databáze, které jsou kompatibilní s moderními požadavky na výkon, dostupnost a správu dat.

Konkrétně byly implementovány různé typy databází, každá z nich zvolena na základě specifických vlastností a výhod, které přináší pro jednotlivé aspekty aplikace. MongoDB, MySQL, Neo4j a Redis byly integrovány tak, aby podporovaly různorodé operace, od transakčního zpracování po rychlé dotazy a efektivní správu vztahů mezi daty.

Pro každou databázi byl vytvořen specifický připojovací mechanismus, pomocí kterého jsou bezpečně spravovány přístupy a manipulace s daty. Informace o připojení, včetně přihlašovacích údajů a konfiguračních parametrů, jsou centralizovaně uchovávány a načítány z konfiguračního souboru .env, čímž je zajištěna bezpečnost citlivých údajů.

3.3.2.1 MySQL připojení

Připojení k MySQL databázi je zajištěno funkcí get_mysql_connection, při které jsou přihlašovací údaje čteny z .env souboru a knihovna mysql.connector je použita pro stabilní a efektivní komunikaci s databází [36].

```
def get_mysql_connection():
    load_dotenv("konfigurace.env")
    mysql_user = os.getenv('MYSQL_USER')
    mysql_password = os.getenv('MYSQL_PASS')
    mysql_host = os.getenv('MYSQL_HOST')
    mysql_database = os.getenv('MYSQL_DB')
    connection = mysql.connector.connect(
        host=mysql_host,
        user=mysql_user,
        password=mysql_password,
        database=mysql_database
    )
    return connection
```

Zdrojový kód 2. Funkce pro připojení MySQL

3.3.2.2 MongoDB připojení

Konfigurační údaje jsou načteny z .env souboru a klient pro interakci s MongoDB je vytvořen. Standardní MongoDB URI spojení s podporou pro SRV záznamy je použito, což umožňuje snadné škálování a správu.

```
def get_mongo_connection():

    load_dotenv("konfigurace.env")

    db_user = os.getenv('MONGO_DB_USER')
    db_password = os.getenv('MONGO_DB_PASS')
    db_name = os.getenv('MONGO_DB_NAME')
    db_host = os.getenv('MONGO_DB_HOST')

    client = MongoClient(f"mongodb+srv://{db_user}:{db_password}@{db_host}/
    ?retryWrites=true&w=majority&appName"f"=dokumentFilip")

    db = client[db_name]
    return db
```

Zdrojový kód 3. Funkce pro připojení MongoDB

3.3.2.3 Redis připojení

Redis je připojen přes funkci get_redis_connection, přičemž knihovna redis je použita pro práci s tímto in-memory úložištěm. Připojovací údaje jsou načteny z .env souboru, což zaručuje snadnou správu a konfiguraci.

```
def get_redis_connection():
    load_dotenv("konfigurace.env")

    redis_host = os.getenv('REDIS_HOST')
    redis_port = os.getenv('REDIS_PORT')
    redis_password = os.getenv('REDIS_PASS')

    rj = Client(host=redis_host, port=redis_port, password=redis_password,
    decode_responses=True)

    return rj
```

Zdrojový kód 4. Funkce pro připojení Redis

3.3.2.4 Neo4j připojení

Pro grafickou databázi Neo4j je použita knihovna neo4j od Neo4j, Inc. URI a přihlašovací údaje jsou načteny z .env souboru, čímž je umožněno spojení s databází pomocí bezpečného autentizačního mechanismu.

```
def get_neo4j_driver():
    uri = os.getenv("NEO4J_URI")
    user = os.getenv("NEO4J_USER")
    password = os.getenv("NEO4J_PASS")

    driver = GraphDatabase.driver(uri, auth=(user, password))
    return driver
```

Zdrojový kód 5. Funkce pro připojení Neo4j

4 POSTUP MĚŘENÍ VÝKONOSTI A PREZENTACE VÝSLEDKŮ

V rámci této kapitoly jsou prezentovány metodologie a výsledky měření výkonnosti implementovaného backendového systému. Cílem bylo kvantifikovat efektivitu a rychlost vybraných operací databáze, které jsou kritické pro celkovou uživatelskou zkušenost a provozní stabilitu aplikace. Měření bylo zaměřeno na základní databázových funkcí jako jsou INSERT, DELETE a UPDATE, které byly provedeny na různých typy databází (MongoDB, MySQL, Neo4j, a Redis) za použití stejného datového modelu a podmínek.

Pro každou z těchto funkcí byl vytvořen scénář testování, který simuluje běžné operace prováděné v produkčním prostředí. Výsledky těchto testů poskytují užitečný náhled na výkonnostní charakteristiky jednotlivých systémů a jejich schopnost zvládat specifické typy zátěže. Dále jsou tyto informace zásadní pro rozhodovací procesy týkající se optimalizace a škálování systému.

4.1 Měření pomocí funkce INSERT

Operace INSERT byla zaměřena na vložení tisíce uživatelů do každé z databází. Tento proces umožňuje posoudit, jak rychle mohou databáze zpracovávat nová data a jaké jsou případné limitace při vyšším objemu transakcí. Zjištění z tohoto testu naznačují, jak dobře je systém připraven na vertikální a horizontální škálování, jelikož konzistentní a rychlé odezvy i při zvyšování zátěže jsou klíčové pro efektivní škálování.

4.1.1 INSERT funkce pro MySQL

V MySQL databázi jsou vložena data nových uživatelů pomocí SQL příkazu INSERT. AUTO_INCREMENT hodnota tabulky je resetována pro demonstrační účely. Po vložení dat jsou změny potvrzeny metodou commit, a konektor je následně zavřen.

```
def save_to_mysql(first_name, last_name, email, phone, state):
    conn = get_mysql_connection()
    cursor = conn.cursor()
    cursor.execute("ALTER TABLE users AUTO_INCREMENT = 1")
    cursor.execute("INSERT INTO users (first_name, last_name, email, phone,
country, status) "
                   "VALUES (%s, %s, %s, %s, %s, 'ACTIVE')",
                   (first_name, last_name, email, phone, state))
    conn.commit()
    cursor.close()
    conn.close()
```

Zdrojový kód 6. Funkce INSERT pro MySQL

4.1.2 INSERT funkce pro MongoDB

Do kolekce users v MongoDB jsou vkládány data nových uživatelů. Připojení k databázi je získáváno z funkce get_mongo_connection(). Identifikátor vloženého dokumentu je vrácen po úspěšném vložení.

```
def save_to_mongo(first_name, last_name, email, phone, country):
    db = get_mongo_connection()
    collection = db['users']
    result = collection.insert_one({
        'first_name': first_name,
        'last_name': last_name,
        'email': email,
        'phone': phone,
        'country': country,
        'status': 'ACTIVE'
    })
    return result.inserted_id
```

Zdrojový kód 7. Funkce INSERT pro MongoDB

4.1.3 INSERT funkce pro Redis

V Redis databázi jsou uloženy data uživatelů pod unikátním klíčem, který je derivován z user_id. Tento identifikátor je inkrementován a uložen pro každého nového uživatele. Data jsou ukládána s použitím Redis JSON a standardní Redis set.

```
def save_to_redis(first_name, last_name, email, phone, country):
    rj = get_redis_connection()
    user_id = int(rj.get('user_id_counter') or 1000)
    user_key = f"user:{user_id}"
    user_data = {
        'first_name': first_name,
        'last_name': last_name,
        'email': email,
        'phone': phone,
        'country': country,
        'status': 'ACTIVE'
    }
    rj.jsonset(user_key, Path.rootPath(), user_data)
    rj.set('user_id_counter', user_id + 1)
    return user_id
```

Zdrojový kód 8. Funkce INSERT pro Redis

4.1.4 INSERT funkce pro Neo4j

V Neo4j databázi jsou vytvářeny nové uzly uživatelů s použitím funkce get_neo4j_driver(). Uživatelé jsou reprezentováni uzlem User s atributy jako jméno, příjmení, email, telefon a země. Po spuštění dotazu je session ukončena a driver je zavřen, čímž jsou uvolněny zdroje.

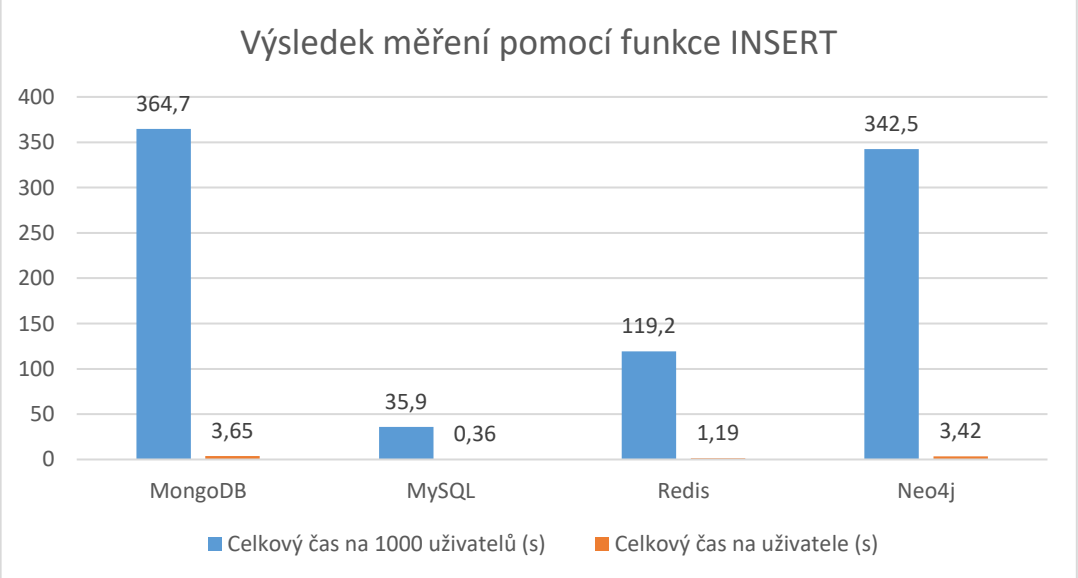
```
def insert_to_neo4j(first_name, last_name, email, phone, country):
    driver = get_neo4j_driver()
    query = """CREATE (u:User {first_name: $first_name, last_name:
$last_name,   email: $email, phone: $phone, country: $country, status:
'ACTIVE'})"""
    params = {
        'first_name': first_name,
        'last_name': last_name,
        'email': email,
        'phone': phone,
        'country': country,
    }
    with driver.session() as session:
        session.write_transaction(lambda tx: tx.run(query, params))
    driver.close()
```

Zdrojový kód 9. Funkce INSERT pro Neo4j

4.1.5 Výsledky měření pomocí funkce INSERT

Tabulka 1. Výsledky měření databází pomocí funkce INSERT

Databáze	Celkový čas na 1000 uživatelů (s)	Celkový čas na uživatele (s)
MongoDB	364,7	3,65
MySQL	35,9	0,36
Redis	119,2	1,19
Neo4j	342,5	3,42



Graf 1. Výsledky měření databází pomocí funkce INSERT

4.1.6 Vyhodnocení měření pomocí funkce INSERT

Analýza výkonu čtyř různých databází—MongoDB, MySQL, Redis a Neo4j—při vložení 1000 uživatelů ukázala značné rozdíly v časech potřebných pro vložení jednotlivých záznamů. Průměrné časy na vložení jednoho uživatele se pohybovaly od 0.36 sekundy u MySQL, což bylo nejrychlejší, až po více než 3 sekundy u MongoDB a Neo4j. Redis, s průměrným časem 1.18 sekundy na uživatele, se umístil mezi těmito dvěma skupinami.

Tyto výsledky naznačují, že výběr databáze by měl být řízen specifickými požadavky aplikace, zvláště pokud jde o požadavky na rychlost zpracování a typ ukládaných dat. MySQL se jeví jako vhodná volba pro aplikace vyžadující rychlé zpracování velkých množství strukturovaných dat, zatímco MongoDB a Neo4j jsou preferovány pro aplikace, kde je potřeba flexibilní manipulace s daty a komplexní dotazování. Redis nabízí dobrý výkon pro rychlé operace s klíčem a hodnotou, které vyžadují rychlý přístup a manipulaci.

4.2 Měření pomocí funkce DELETE

Pro testování databází pomocí funkce DELETE byly vytvořeny dva scénáře: první zahrnoval odstranění všech uživatelů z databází, zatímco druhý se zaměřil na smazání uživatelů, jejichž jména a příjmení začínají specifickými písmeny. Tyto testy jsou klíčové pro hodnocení efektivity databází v situacích, kdy je potřeba rychle uvolnit prostor nebo provést selektivní čištění dat. Schopnost databází rychle a efektivně reagovat na DELETE operace je indikátorem dobře škálovatelného systému, který může efektivně zvládat i vysoké objemy smazání bez významného dopadu na celkový výkon.

4.2.1 Měření výkonnosti pomocí funkce DELETE pro odstranění všech uživatelů

V této podsekci je zkoumána výkonnost databáze pomocí funkce DELETE, při které byly odstraněny všechny uživatelské záznamy z databází. Tento test je prováděn s cílem posoudit, jak efektivně a rychle jsou databázové systémy schopny uvolnit prostor a zpracovat velké objemy dat. Význam tohoto měření spočívá ve schopnosti systémů rychle reagovat na požadavky na údržbu databáze nebo přípravu dat pro migrace. Tato analýza poskytuje důležité informace o odolnosti a optimalizaci databázových systémů v situacích vyžadujících promptní zpracování datových změn.

4.2.1.1 DELETE funkce pro všechny uživatele MySQL

Pro MySQL je funkce DELETE implementována funkcí delete_from_mysql(). Připojení k databázi je navázáno pomocí get_mysql_connection(). Pomocí kurzoru je spuštěn SQL příkaz DELETE FROM users, který odstraní všechny záznamy z tabulky uživatelů. Po provedení operace jsou změny potvrzeny a připojení uzavřeno.

```
def delete_from_mysql():
    conn = get_mysql_connection()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM users")
    conn.commit()
    cursor.close()
    conn.close()
```

Zdrojový kód 10. Funkce DELETE pro všechny uživatele MySQL

4.2.1.2 DELETE funkce pro všechny uživatele MongoDB

V MongoDB jsou všechny dokumenty uživatelů odstraněny funkcí delete_from_mongo(). Připojení k databázi je získáno přes get_mongo_connection(), a metoda delete_many() s prázdným filtrem je použita na kolekci 'users' k odstranění všech dokumentů. Operace je automaticky potvrzena MongoDB.

```
def delete_from_mongo():
    db = get_mongo_connection()
    db['users'].delete_many({})
```

Zdrojový kód 11. Funkce DELETE pro všechny uživatele MongoDB

4.2.1.3 DELETE funkce pro všechny uživatele Redis

Redis provádí smazání všech uživatelů funkcí delete_from_redis(). Připojení je získáno přes get_redis_connection(). Iterace přes všechny klíče začínající "user:" je provedena pomocí scan_iter(), a každý nalezený klíč je následně smazán funkcí delete().

```
def delete_from_redis():
    r = get_redis_connection()
    for key in r.scan_iter("user:*"):
        r.delete(key)
```

Zdrojový kód 12. Funkce DELETE pro všechny uživatele Redis

4.2.1.4 DELETE funkce pro všechny uživatele Neo4j

V Neo4j je funkce DELETE prováděna funkcí delete_from_neo4j(). Pomocí driveru získaného z funkce get_neo4j_driver() je otevřena session, ve které je spuštěn Cypher příkaz pro smazání všech uzlů reprezentujících uživatele. Po dokončení operace je session uzavřena a driver zavřen.

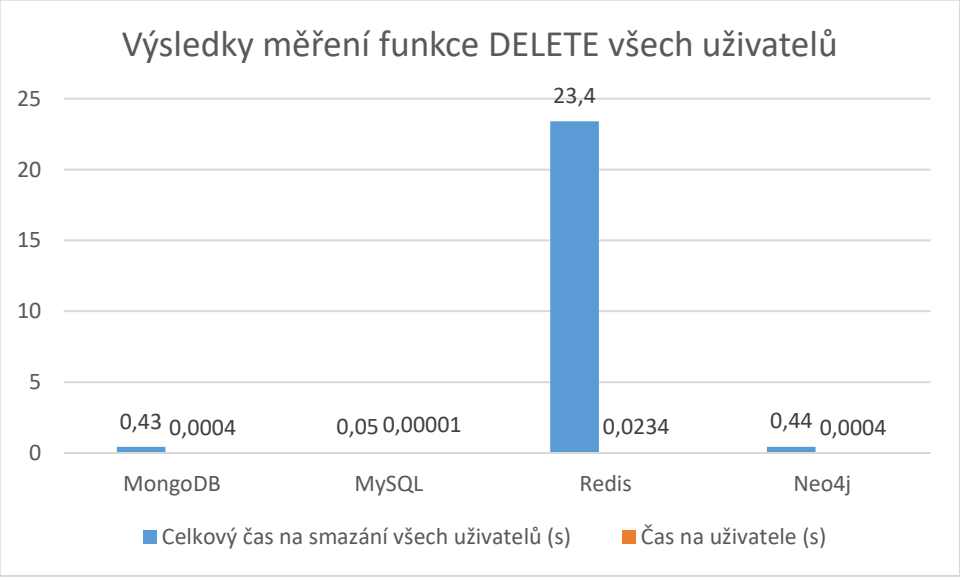
```
def delete_from_neo4j():
    driver = get_neo4j_driver()
    with driver.session() as session:
        session.run("MATCH (u:User) DELETE u")
    driver.close()
```

Zdrojový kód 13. Funkce DELETE pro všechny uživatele Neo4j

4.2.1.5 Výsledky měření pomocí funkce DELETE pro všechny uživatele

Tabulka 2. Výsledky měření databází pomocí funkce DELETE všech uživatelů

Databáze	Celkový čas na smazání všech uživatelů (s)	Čas na uživatele (s)
MongoDB	0,43	0,0004
MySQL	0,05	0,00001
Redis	23,4	0,0234
Neo4j	0,44	0,0004



Graf 2. Výsledky měření databází pomocí funkce DELETE všech uživatelů

4.2.2 Měření výkonnosti pomocí funkce DELETE podle iniciál

V této podsekci je zkoumána výkonnost databáze funkcí DELETE, kdy jsou smazáni uživatelé na základě iniciál jejich jmen a příjmení. Tento přístup umožňuje posoudit, jak efektivně jsou databázové systémy schopny provádět selektivní mazání záznamů, což je důležité pro aplikace, které vyžadují dynamické spravování dat na základě specifických atributů. Analýza této operace poskytuje cenné informace o schopnostech databází efektivně lokalizovat a odstraňovat záznamy, což je kritické pro optimalizaci udržitelnosti a výkonu v reálném čase.

4.2.2.1 DELETE funkce podle iniciál pro MySQL

Pro MySQL je definována funkce delete_custom_users_mysql(), která smaže záznamy uživatelů s iniciály odpovídajícími zadaným hodnotám v jménu a příjmení. Připojení je získáno a SQL dotaz je proveden s použitím LIKE operátoru, který umožňuje flexibilní porovnání řetězců. Počet odstraněných řádků je určen a vrácen.

```
def delete_custom_users_mysql(first_name_initial, last_name_initial):
    conn = get_mysql_connection()
    cursor = conn.cursor()
    query = """
    DELETE FROM users WHERE first_name LIKE %s AND last_name LIKE %s
    """
    cursor.execute(query, (first_name_initial + '%', last_name_initial + '%'))
    deleted_count = cursor.rowcount
    conn.commit()
    cursor.close()
    conn.close()
    return deleted_count
```

Zdrojový kód 14. Funkce DELETE podle iniciál pro MySQL

4.2.2.2 DELETE funkce podle iniciál pro MongoDB

V MongoDB je funkce delete_custom_users_mongo() použita pro odstranění uživatelů, jejichž jména a příjmení začínají na specifikované iniciály. Připojení k databázi je navázáno pomocí get_mongo_connection(). Uživatelé jsou identifikováni a odstraněni s použitím regulárních výrazů, které filtrují jména a příjmení podle zadání. Výsledek, který obsahuje počet odstraněných dokumentů, je vrácen.

```
def delete_custom_users_mongo(first_name_initial, last_name_initial):
    db = get_mongo_connection()
    regex_first = f"^{first_name_initial}"
    regex_last = f"^{last_name_initial}"
    result = db['users'].delete_many({
        "first_name": {"$regex": regex_first, "$options": "i"},
        "last_name": {"$regex": regex_last, "$options": "i"}
    })
    return result.deleted_count
```

Zdrojový kód 15. Funkce DELETE podle iniciál pro MongoDB

4.2.2.3 DELETE funkce podle iniciál pro Redis

V Redis je implementována funkce delete_custom_users_redis(), kde jsou procházeny klíče a uživatelé s odpovídajícími iniciály jsou identifikováni a smazáni. Počet smazaných záznamů je inkrementován a vrácen.

```
def delete_custom_users_redis(first_name_initial, last_name_initial):
    rj = get_redis_connection()
    deleted_count = 0
    for key in rj.scan_iter("user:*"):
        user = rj.jsonget(key)
        if (user['first_name'][0].upper() == first_name_initial.upper() and
            user['last_name'][0].upper() == last_name_initial.upper()):
            rj.delete(key)
            deleted_count += 1
    return deleted_count
```

Zdrojový kód 16. Funkce DELETE podle iniciál pro Redis

4.2.2.4 DELETE funkce podle iniciál pro Neo4j

Funkce delete_custom_users_neo4j() v Neo4j provádí odstranění uzlů reprezentujících uživatele, kteří mají jména a příjmení začínající na zadané iniciály. Cypher dotaz je spuštěn v rámci otevřené session a počet odstraněných uzlů je vrácen.

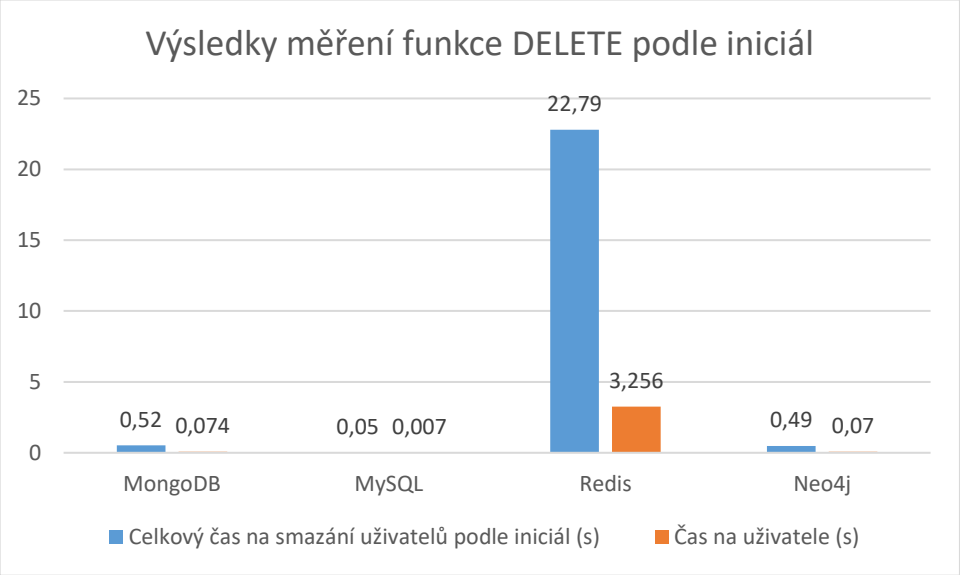
```
def delete_custom_users_neo4j(first_name_initial, last_name_initial):
    driver = get_neo4j_driver()
    with driver.session() as session:
        result = session.run("""
        MATCH (u:User)
        WHERE u.first_name STARTS WITH $first_name_initial
        AND u.last_name STARTS WITH $last_name_initial
        DELETE u
        RETURN count(u) as count
        """, {'first_name_initial': first_name_initial, 'last_name_initial':
last_name_initial})
        deleted_count = result.single()["count"]
    driver.close()
    return deleted_count
```

Zdrojový kód 17. Funkce DELETE podle iniciál pro Neo4j

4.2.2.5 Výsledky měření pomocí funkce DELETE podle iniciál

Tabulka 3. Výsledky měření databází pomocí funkce DELETE podle iniciál

Databáze	Celkový čas na smazání uživatelů podle iniciál (s)	Čas na uživatele (s)
MongoDB	0,52	0,074
MySQL	0,05	0,007
Redis	22,79	3,256
Neo4j	0,49	0,07



Graf 3. Výsledky měření databází pomocí funkce DELETE podle iniciál

4.2.3 Vyhodnocení měření databází pomocí funkce DELETE

Měření výkonnosti databází pomocí funkce DELETE bylo provedeno ve dvou různých scénářích: odstranění všech uživatelů a odstranění uživatelů na základě iniciál jejich jmen a příjmení. Tyto testy poskytly důležité poznatky o efektivitě a rychlosti databází v různých operacích mazání.

Odstranění všech uživatelů:

- Výsledky ukázaly výrazně rychlejší operace v MySQL, který dosáhl časů kolem 0.05 sekundy, což naznačuje, že jeho architektura je optimalizovaná pro rychlé mazání velkých objemů dat.
- MongoDB a Neo4j také prokázaly dobré výsledky s časy pod jednu sekundu, což je indikativní pro jejich schopnost efektivně zpracovávat operace na grafově orientovaných a dokumentově orientovaných databázích.

- Redis vykázal delší časy, což může být ovlivněno jeho strukturou dat a způsobem, jakým ukládá a indexuje informace.

Odstranění uživatelů podle iniciál:

- Tento scénář ukázal podobné trendy, kde MySQL zůstává nejrychlejší, což ukazuje na jeho robustní indexace a optimalizace pro selektivní dotazy.
- Neo4j a MongoDB opět demonstrovaly solidní výkon, což reflektuje jejich schopnost zvládat komplexní dotazy na výběr dat.
- Redis, ačkoliv není primárně určen pro složité dotazování, ukázal vyšší časy, které by mohly být redukovány s lepšími indexačními strategiemi nebo optimalizací dotazů.

Tato analýza poskytuje důležitý přehled o schopnostech každé databáze zvládat různé typy operací DELETE. Výběr databáze by měl být prováděn s ohledem na specifické požadavky aplikace, zejména pokud jde o potřebu rychlých a efektivních operací mazání. MySQL se ukázalo jako nejvhodnější pro operace, které vyžadují rychlost a efektivitu, zatímco MongoDB a Neo4j poskytují výhody pro aplikace, kde jsou důležité flexibilita a složité dotazy. Redis, přestože nebyl nejrychlejší v testech mazání, může být vhodný pro aplikace, kde je důležitý rychlý přístup k datům pro čtení a zápis.

4.3 Měření pomocí funkce UPDATE

Zde bude zkoumáno výkonnost databázích pomocí funkce UPDATE. Měření bylo zaměřeno na dva hlavní scénáře: první, kde byl aktualizován status všech uživatelů na 'active' nebo 'inactive', a druhý, kde byly prováděny aktualizace statusu podle země, ve které uživatelé sídlí. Tato měření poskytují užitečné informace o schopnosti databází efektivně zvládnout hromadné operace aktualizace, stejně jako o jejich schopnosti zvládat komplexnější dotazy spojené s lokálními aktualizacemi. Tyto testy jsou klíčové pro posouzení, jak dobře mohou databázové systémy reagovat na požadavky aplikací vyžadující časté a rozsáhlé změny dat.

4.3.1 Měření výkonnosti pomocí funkce UPDATE na změnu statusu všech uživatelů

V této sekci je zkoumána výkonnost databází pomocí funkce Update, při které byl změněn status všech uživatelů na 'active' nebo 'inactive'. Tato funkce je zásadní pro situace vyžadující rychlé přizpůsobení systému změnám v uživatelských pravidlech nebo reakci na mimořádné události. Schopnost databází efektivně provádět hromadné aktualizace je testována, aby bylo možné posoudit, jak dobře mohou systémy zvládat náhlé a rozsáhlé změny v datech.

Toto měření poskytuje klíčové informace o tom, jak rychle a efektivně mohou být data aktualizována ve velkém měřítku pomocí funkce Update.

4.3.1.1 UPDATE funkce pro nastavení statusu všem uživatelům MySQL

Ve funkci update_mysql_status(status) je pro MySQL použito připojení získané funkcí get_mysql_connection(). SQL příkaz pro aktualizaci je spuštěn pomocí kurzoru, který nastavuje status všech uživatelů v tabulce 'users'. Po provedení aktualizace jsou změny potvrzeny a spojení s databází je uzavřeno.

```
def update_mysql_status(status):
    conn = get_mysql_connection()
    cursor = conn.cursor()
    cursor.execute("UPDATE users SET status = %s", (status,))
    conn.commit()
    cursor.close()
    conn.close()
```

Zdrojový kód 18. Funkce UPDATE pro změnu statusu uživatelů MySQL

4.3.1.2 UPDATE funkce pro nastavení statusu všem uživatelům MongoDB

Funkce update_mongo_status(status) je používána pro aktualizaci statusu všech uživatelů v MongoDB. Připojení k databázi je získáno pomocí get_mongo_connection(). V kolekci 'users' je použita metoda update_many() s prázdným filtrem, která aplikuje změnu statusu na všechny dokumenty. Počet aktualizovaných dokumentů je vracen.

```
def update_mongo_status(status):
    db = get_mongo_connection()
    db['users'].update_many({}, {"$set": {"status": status}})
```

Zdrojový kód 19. Funkce UPDATE pro změnu statusu uživatelů MongoDB

4.3.1.3 UPDATE funkce pro nastavení statusu všem uživatelům Redis

Funkce update_redis_status(status) v Redisu začíná získáním připojení přes get_redis_connection(). Prochází všechny klíče uživatelů a pomocí jsonset() je aktualizován status v JSON dokumentech. Počet aktualizovaných záznamů je inkrementován a vrácen.

```
def update_redis_status(status):
    rj = get_redis_connection()
    for key in rj.scan_iter("user:*"):
        rj.jsonset(key, Path('.status'), status)
```

Zdrojový kód 20. Funkce UPDATE pro změnu statusu uživatelů Redis

4.3.1.4 *UPDATE funkce pro nastavení statusu všem uživatelům Neo4j*

Pro Neo4j je definována funkce update_neo4j_status(status), kde je pomocí get_neo4j_driver() získáno připojení. V rámci otevřené session je spuštěn Cypher příkaz, který aktualizuje status všech uzlů reprezentujících uživatele. Počet aktualizovaných uzlů je vracen a driver je uzavřen.

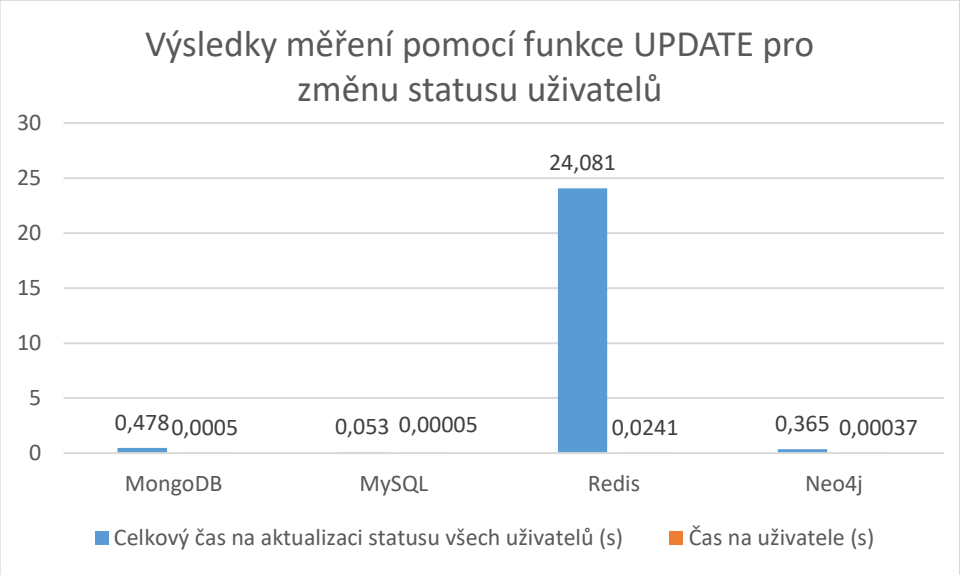
```
def update_neo4j_status(status):
    driver = get_neo4j_driver()
    with driver.session() as session:
        session.run("MATCH (u:User) SET u.status = $status", status=status)
    driver.close()
```

Zdrojový kód 21. Funkce UPDATE pro změnu statusu uživatelů Neo4j

4.3.1.5 *Výsledky měření pomocí funkce UPDATE pro změnu statusu všech uživatelů*

Tabulka 4. Výsledky měření databází pomocí funkce UPDATE pro změnu statusu všech uživatelů

Databáze	Celkový čas na aktualizaci statusu všech uživatelů (s)	Čas na uživatele (s)
MongoDB	0,478	0,0005
MySQL	0,053	0,00005
Redis	24,081	0,0241
Neo4j	0,365	0,00037



Graf 4. Výsledky měření databází pomocí funkce UPDATE pro změnu statusu všech uživatelů

4.3.2 Měření výkonnosti pomocí funkce UPDATE na změnu statusu podle státu

Zde bude zkoumána výkonnost databází pomocí funkce UPDATE, při které je status uživatelů aktualizován na základě státu, ve kterém sídlí. Tato segmentovaná aktualizace je klíčová pro systémy, které vyžadují přizpůsobení reakcí podle specifických datových atributů. Schopnost databází efektivně provádět aktualizace podle určitého kritéria je testována s cílem posoudit, jak dobře mohou systémy zvládat dotazy s vysokým stupněm specifičnosti. Analýza těchto operací odhaluje, jaký dopad mají na celkovou výkonnost a efektivitu databázových systémů.

4.3.2.1 UPDATE funkce pro nastavení statusu podle státu MySQL

Funkce update_mysql_country_status(country, status) je používána pro aktualizaci statusu uživatelů v MySQL, kteří sídlí ve specifikovaném státě. Připojení k databázi je získáno pomocí get_mysql_connection(). SQL příkaz pro aktualizaci je spuštěn, který nastavuje status uživatelů v tabulce 'users' na základě jejich země. Po provedení aktualizace jsou změny potvrzeny a spojení s databází je uzavřeno.

```
def update_mysql_country_status(country, status):
    conn = get_mysql_connection()
    cursor = conn.cursor()
    cursor.execute("UPDATE users SET status = %s WHERE country = %s", (status,
country))
    conn.commit()
    cursor.close()
    conn.close()
```

Zdrojový kód 22. Funkce UPDATE pro změnu statusu podle státu Neo4j

4.3.2.2 UPDATE funkce pro nastavení statusu podle státu MongoDB

Ve funkci update_mongo_country_status(country, status) je status uživatelů v MongoDB aktualizován na základě státu, ve kterém sídlí. Připojení k databázi je navázáno přes get_mongo_connection(). Metoda update_many() je použita s filtrem na zemi, což umožňuje hromadnou aktualizaci statusu pro všechny dokumenty odpovídající dané zemi.

```
def update_mongo_country_status(country, status):
    db = get_mongo_connection()
    db['users'].update_many({"country": country}, {"$set": {"status": status}})
```

Zdrojový kód 23. Funkce UPDATE pro změnu statusu podle státu MongoDB

4.3.2.3 UPDATE funkce pro nastavení statusu podle státu Redis

Funkce update_redis_country_status(country, status) v Redisu začíná získáním připojení přes get_redis_connection(). Prochází všechny klíče uživatelů a aktualizuje jejich status, kteří mají zadanou zemi ve svém záznamu. Tato operace je prováděna pomocí jsonset() pro každý relevantní klíč.

```
def update_redis_country_status(country, status):
    rj = get_redis_connection()
    for key in rj.scan_iter("user:*"):
        if rj.jsonget(key, Path('.country')) == country:
            rj.jsonset(key, Path('.status'), status)
```

Zdrojový kód 24. Funkce UPDATE pro změnu statusu podle státu Redis

4.3.2.4 UPDATE funkce pro nastavení statusu podle státu Neo4j

Neo4j funkce update_neo4j_country_status(country, status) aktualizuje status uživatelů, kteří sídlí v určitém státu. Připojení je získáno z get_neo4j_driver(). V rámci otevřené session je spuštěn Cypher příkaz, který aktualizuje status uzlů uživatelů na základě země. Po dokončení aktualizace je session uzavřena.

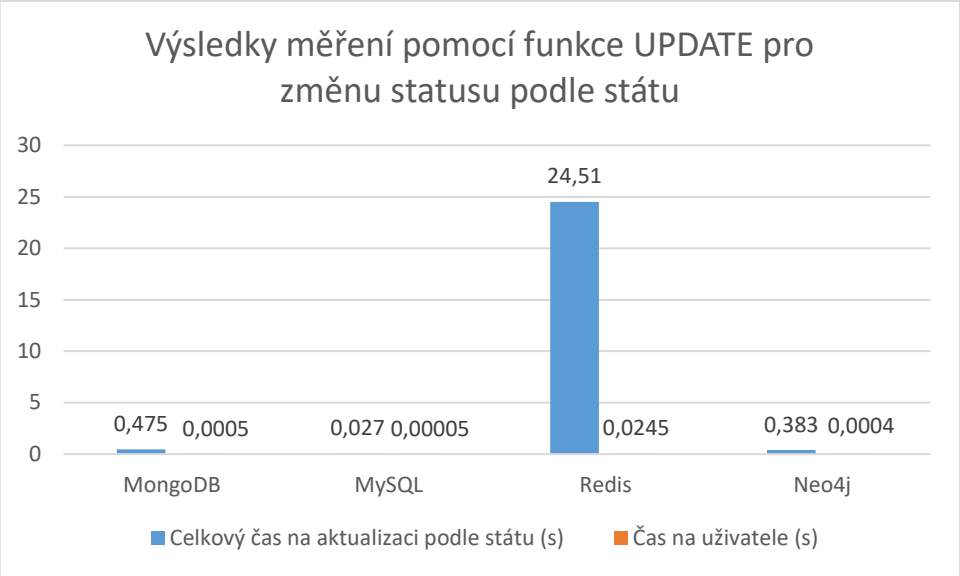
```
def update_neo4j_country_status(country, status):
    driver = get_neo4j_driver()
    with driver.session() as session:
        session.run("MATCH (u:User {country: $country}) SET u.status =
$status",
                    country=country, status=status)
    driver.close()
```

Zdrojový kód 25. Funkce UPDATE pro změnu statusu podle státu Neo4j

4.3.2.5 Výsledky měření pomocí funkce UPDATE pro změnu statusu podle státu

Tabulka 5. Výsledky měření databází pomocí funkce UPDATE pro změnu statusu podle státu

Databáze	Celkový čas na aktualizaci podle státu (s)	Čas na uživatele (s)
MongoDB	0,475	0,0005
MySQL	0,027	0,00005
Redis	24,51	0,0245
Neo4j	0,383	0,0004



Graf 5. Výsledky měření databází pomocí funkce UPDATE pro změnu statusu podle státu

4.3.3 Vyhodnocení měření databází pomocí funkce UPDATE

Provedená měření výkonnosti databází pomocí funkce UPDATE byla zaměřena na dvě specifické aktualizační úlohy: změnu statusu u všech uživatelů a změnu statusu uživatelů dle jejich státu. Z těchto testů byly získány cenné informace o rychlosti a efektivitě databázových systémů při zpracování aktualizačních požadavků.

Aktualizace statusu všech uživatelů:

- MySQL prokázalo, že je schopno zpracovávat aktualizace s vysokou rychlostí, což signalizuje jeho optimalizaci pro rychlé zpracování masových aktualizací.
- MongoDB a Neo4j rovněž dosáhly krátkých časů aktualizace, což poukazuje na jejich efektivitu při zpracování operací na grafových a dokumentových databázích.
- Redis, s delšími časy aktualizace, ukázal, že jeho data management může vyžadovat optimalizaci pro lepší zpracování hromadných aktualizací.

Aktualizace statusu podle státu:

- MySQL opět vykazovalo nejnižší dobu aktualizace, což zdůrazňuje jeho schopnost rychle reagovat na selektivní dotazy.
- MongoDB a Neo4j úspěšně zvládly komplexní dotazy spojené s aktualizacemi založenými na geografických údajích, což ukazuje na jejich silné stránky v manipulaci s daty s vysokou úrovní detailů.

- Redis projevil zvýšené časy, což naznačuje potřebu prohloubení jeho indexačních a dotazovacích strategií pro efektivnější výsledky.

Toto vyhodnocení odhaluje, že volba databáze by měla být zvážena s důrazem na specifické nároky aplikací, které vyžadují efektivní a rychlé aktualizační operace. Zatímco MySQL se jeví jako nejvhodnější volba pro operace vyžadující rychlou odpověď, MongoDB a Neo4j nabízejí robustní řešení pro aplikace s komplexními požadavky na dotazy. Redis, ačkoliv se nejeví jako nejlepší pro rychlé aktualizace, může být ideální pro aplikace, kde je kritické rychlé získávání dat.

ZÁVĚR

Tato bakalářská práce se detailně zabývala různými aspekty databázových technologií, zahrnující jak teoretické, tak praktické přístupy k rozhodování o výběru nejvhodnější databáze pro různé typy aplikací. V teoretické části byla podána komplexní analýza rozhodovacích kritérií, včetně škálovatelnosti, výkonnosti, datového modelu, bezpečnosti a nákladů na databázové systémy. Tato analýza poskytla hluboký náhled do současných trendů v databázových technologiích, jako jsou cloudové databáze, umělá inteligence a in-memory databáze.

Dále byla provedena komparativní analýza mezi relačními a NoSQL databázemi, která odhalila klíčové rozdíly a naznačila možné aplikace těchto technologií v závislosti na konkrétních požadavcích aplikací. Tyto teoretické poznatky byly následně aplikovány v praktické části práce, kde byla navržena a implementována benchmarková aplikace. Tato aplikace byla využita k testování a porovnání výkonnosti různých typů databází, čímž byla poskytnuta objektivní data o efektivitě jednotlivých databázových technologií pro různé operace.

Výsledky těchto testů přinesly cenné informace o tom, jak efektivita jednotlivých databázových operací, jako jsou aktualizace a mazání dat, může výrazně ovlivnit celkový výkon aplikací. Bylo demonstrováno, že vhodný výběr databáze může zásadně zlepšit reakční schopnosti a škálovatelnost systémů, což je zvláště kritické pro aplikace, které vyžadují rychlou odezvu a zpracování velkých objemů dat.

Tato práce tak představuje komplexní pohled na databázové technologie a nabízí solidní základ pro výběr a implementaci nejefektivnějšího řešení pro databázové systémy v moderním informačním prostředí. Výsledky této práce slouží jako důležitý zdroj informací pro vývojáře, administrátory databází a technologické strategie, které jsou zaměřeny na optimalizaci databázových operací a na zlepšení celkové efektivity aplikací, což umožňuje organizacím udržet krok s neustálým technologickým pokrokem a dynamicky se měnícím trhem.

SEZNAM POUŽITÉ LITERATURY

[1] Database Scalability. *ScyllaDB* [online]. c2024 [cit. 2024-02-28]. Dostupné z: https://www.scylladb.com/glossary/database-scalability/

[2] OSMAN, A.Hokman. 7 Key Factors to Consider When Choosing a Database for Your Application. *Medium* [online]. 2023-06-17 [cit. 2024-02-28]. Dostupné z: https://bootcamp.uxdesign.cc/7-key-factors-to-consider-when-choosing-a-database-for-your-application-a8ff14cd8305

[3] WU, Alex. Factors to Consider in Database Selection. In: *ByteByteGo Newsletter* [online]. 2023-04-26 [cit. 2024-02-29]. Dostupné z: https://blog.bytebytego.com/p/factors-to-consider-in-database-selection

[4] SAMSON, Babatunde. Database Speed: What is it, and why should you care? *Medium* [online]. 2022-10-14 [cit. 2024-03-01]. Dostupné z: https://medium.com/techtrument/database-speed-what-is-it-and-why-should-you-care-2a7519a9a77f

[5] Estimate the Size of a Database. *Microsoft* [online]. 2023-04-04 [cit. 2024-03-01]. Dostupné z: https://learn.microsoft.com/en-us/sql/relational-databases/databases/estimate-the-size-of-a-database?view=sql-server-ver16

[6] Emerging Trends in Database Management Systems. *EMB blogs* [online]. 2024-01-23 [cit. 2024-05-12]. Dostupné z: https://blog.emb.global/emerging-trends-in-database-management-systems/

[7] Top 5 Current Database Trends. *Datamation* [online]. 2023-07-11 [cit. 2024-05-12]. Dostupné z: https://www.datamation.com/cloud/current-database-trends/

[8] D. FOOTE, Keith. Database Management Trends in 2023. *DATAVERSITY* [online]. 2022-12-06 [cit. 2024-05-12]. Dostupné z: https://www.dataversity.net/database-management-trends-in-2023/

[9] What Is a Data Warehouse? *Oracle* [online]. c2024 [cit. 2024-05-12]. Dostupné z: https://www.oracle.com/cz/database/what-is-a-data-warehouse/

[10] What is a data lakehouse? *Google Cloud* [online]. c2024 [cit. 2024-05-12]. Dostupné z: https://cloud.google.com/discover/what-is-a-data-lakehouse

[11] Embracing Database Technology Trends in 2024. *Linkedin* [online]. 2024-01-02 [cit. 2024-05-12]. Dostupné z: https://www.linkedin.com/pulse/embracing-database-technology-trends-2024-datadots-xpjsf

[12] Data Integrity. *Qlik* [online]. c1993-2024 [cit. 2024-05-12]. Dostupné z: https://www.qlik.com/us/data-management/data-integrity

[13] What is Data Integrity in a Database? Why Do You Need It? *Astera* [online]. c2024 [cit. 2024-05-12]. Dostupné z: https://www.astera.com/type/blog/data-integrity-in-a-database/

[14] NAEEM, Tehreem. What Is Data Integrity? *Fortinet* [online]. 2024-05-25 [cit. 2024-05-12]. Dostupné z: https://www.fortinet.com/resources/cyberglossary/data-integrity

[15] Normal Forms in DBMS. *GeeksforGeeks* [online]. 2023-11-06 [cit. 2024-05-12]. Dostupné z: https://www.geeksforgeeks.org/normal-forms-in-dbms/

[16] CHRIS, Kolade. Database Normalization – Normal Forms 1nf 2nf 3nf Table Examples. *FreeCodeCamp* [online]. 2022-12-21 [cit. 2024-05-12]. Dostupné z: https://www.freecodecamp.org/news/database-normalization-1nf-2nf-3nf-table-examples/

[17] A S, Ravikirian. What is Normalization in SQL? 1NF, 2NF, 3NF and BCNF in DBMS. *Simplelearn* [online]. 2023-12-21 [cit. 2024-05-12]. Dostupné z: https://www.simplilearn.com/tutorials/sql-tutorial/what-is-normalization-in-sql

[18] PETERSON, Richard. What is Normalization in SQL? 1NF, 2NF, 3NF and BCNF in DBMS. *simplelearn* [online]. 2023-05-15 [cit. 2024-05-12]. Dostupné z: https://www.guru99.com/database-normalization.html

[19] ZANINI, Antonello. Database Transactions 101: The Essential Guide. *Dbvis* [online]. 2023-02-14 [cit. 2024-05-12]. Dostupné z: https://www.dbvis.com/thetable/database-transactions-101-the-essential-guide/

[20] What is a database transaction? *Fauna* [online]. 2023-02-14 [cit. 2024-05-12]. Dostupné z: https://fauna.com/blog/database-transaction#how-do-database-transactions-work

[21] LAURENČÍK, Marek. SQL: podrobný průvodce uživatele. Praha: Grada Publishing, 2018. Průvodce (Grada). ISBN 978-80-271-0774-2.

[22] ACID Properties in DBMS. *GeeksforGeeks* [online]. 2023-04-11 [cit. 2024-05-12]. Dostupné z: https://www.geeksforgeeks.org/acid-properties-in-dbms/

[23] WASEEM, Khan. What is CAP theorem? In: *Khan Waseem* [online]. 2023-01-27 [cit. 2024-05-12]. Dostupné z: https://www.khanwaseem.com/blog/programming-language/what-is-cap-theorem

[24] GUPTA, Neha. Understanding the CAP Theorem: Balancing Consistency, Availability, and Partition. *Medium* [online]. 2023-04-21 [cit. 2024-05-12]. Dostupné z: https://medium.com/@ngneha090/understanding-the-cap-theorem-balancing-consistency-availability-and-partition-cb11c2b97e2b

[25] What is the CAP theorem? *IBM* [online]. 2023-09-29 [cit. 2024-05-12]. Dostupné z: https://www.ibm.com/topics/cap-theorem

[26] What is a key-value store? *Hazelcast* [online]. c2024 [cit. 2024-05-12]. Dostupné z: https://hazelcast.com/glossary/key-value-store/

[27] WHAT IS A KEY-VALUE DATABASE? *Redis* [online]. c2024 [cit. 2024-05-12]. Dostupné z: https://redis.io/nosql/key-value-databases/

[28] What Is a Key-Value Pair (KVP)? Definition and Examples. *Indeed* [online]. 2022-06-25 [cit. 2024-05-12]. Dostupné z: https://www.indeed.com/career-advice/career-development/key-value-pair

[29] Key-Value Database (Use Cases, List, Pros & Cons). *DatabaseTown* [online]. 2022-06-25 [cit. 2024-05-12]. Dostupné z: https://databasetown.com/key-value-database-use-cases/

[30] PYKES, Kurtis. What is A Graph Database? A Beginner's Guide. *Datacamp* [online]. 2023-10 [cit. 2024-05-12]. Dostupné z: https://www.datacamp.com/blog/what-is-a-graph-database

[31] DANCUK, Milica. What Is a Graph Database? *PhoenixNAP* [online]. 2021-04 [cit. 2024-05-12]. Dostupné z: https://phoenixnap.com/kb/graph-database

[32] ARCHER, Cameron. What are columnar databases? Here are 35 examples. *Tinybird* [online]. 2023-04-22 [cit. 2024-05-12]. Dostupné z: https://www.tinybird.co/blog-posts/what-is-a-columnar-database

[33] KUMAR, Dharmendra. What is Columnar Database? – A Comprehensive Guide 101. *Hevo* [online]. 2021-06-22 [cit. 2024-05-12]. Dostupné z: https://hevodata.com/learn/columnar-databases/

[34] What is a Document Database? *MongoDB* [online]. c2024 [cit. 2024-05-12]. Dostupné z: https://www.mongodb.com/resources/basics/databases/document-databases

[35] EMERICH, Alex. What are document databases? *Prisma* [online]. c2024 [cit. 2024-05-12]. Dostupné z: https://www.prisma.io/dataguide/types/document/what-are-document-dbs

[36] WILLIAMS, Alex. What is a Document Database? *PhoenixNAP* [online]. 2021-03-13 [cit. 2024-05-12]. Dostupné z: https://phoenixnap.com/kb/document-database

[37] MySQL Connector/Python Developer Guide. *MySQL* [online]. 2021-03-13 [cit. 2024-05-12]. Dostupné z: https://dev.mysql.com/doc/connector-python/en/

[38] Microsoft Docs - Microsoft SQL Documentation: Oficiální Dokumentace Společnosti Microsoft Corporation. Online. 2023. Dostupné z https:\\learn.microsoft.com/en-us/sql. [cit. 2024-05-12].

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

IBM International Business Machines Corporation

SQL Structured Query Language

IoT Internet of Things

NF Normální forma

BCNF Boyce-Coddova normální forma

ACID Atomicity Consistency Isolation Durability

SQL Structured Query Language

IDE Integrated Development Environment

UME Unified Modeling Language

EA Enterprise Architect

SEZNAM OBRÁZKŮ

SEZNAM ZDROJOVÝCH KÓDŮ

SEZNAM TABULEK

SEZNAM GRAFŮ

SEZNAM PŘÍLOH

Příloha P I: CD s bakalářskou prací a se zdrojovými kódy aplikace

PŘÍLOHA P I: CD

Přiložené CD obsahuje:

- Bakalářskou práci ve formátu .pdf: FilipSedlar_BP_A21355.pdf
- Zdrojové kódy aplikace a tabulky s grafy: FilipSedlar_BP_PRACT_A21355.zip