Univerzita Hradec Králové
Fakulta informatiky a managementu
Název katedry

Analýza NoSQL databází a jejich srovnání

Diplomová práce

Autor: Bc. Jan Drnek
Studijní obor: AI2-K

Vedoucí práce: doc. Ing. Filip Malý, Ph.D.

Hradec Králové srpen 2017

Prohlášení:

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

vlastnoruční podpis

V Hradci Králové dne 17.8.2017 Drnek Jan

Poděkování:

Děkuji vedoucímu diplomové práce doc. Ing. Filip Malému, Ph.D., za metodické vedení práce, za cenné připomínky a odborné rady, které mi pomohly k vypracování diplomové práce.

Anotace

Databázové systémy NoSQL jsou na vzestupu díky každodennímu velkému nárůstu počtu dat. Termín Big data se skloňuje každým dnem víc a víc a databázové systémy NoSQL nahrazují a doplňují databázové systémy SQL tam, kde díky své pevně dané struktuře nestačí.

Diplomová práce pokrývá hlavní kategorie NoSQL databázových systémů. Jmenovitě databáze typu klíč - hodnota, sloupcové databáze, dokumentové databáze a grafové databáze. Popsán je termín Big data, který se k této problematice úzce váže, stejně jako proces ETL a další nástroje oboru Business Inteligence.

U jednotlivých zástupců NoSQL databázových systémů jsou popsány základní principy fungování, důvody vzniku, instalace, práce s databází a v závěru jsou jednotlivé databázové systémy navzájem porovnány.

Klíčová slova: NoSQL, Big data, databáze, ETL, replikace, sharding, CAP teorém, E-R model, databáze typu klíč-hodnota, sloupcové databáze, dokumentové databáze, grafové databáze, SQL, Apache Cassandra, MongoDB, Neo4j, Redis, YCSB

Annotation

Title: Analysis of NoSQL databases and their comparison

NoSQL database systems are on the rise due to daily increase of a large amount of data. Term Big data is discussed more and more every day, and NoSQL systems replace and complement SQL database systems, where they are not enough because of their given structure.

This diploma thesis covers main category of NoSQL database systems. Particularly key-value databases, column databases, document databases and graph databases. The term Big data is described, which is closely related to this issue, as well as the ETL process and other Business Intelligence tools.

For individual representatives of NoSQL database systems are described basic principles, installation, working with databases and at the end database systems are compared with each other.

Key words: NoSQL, Big data, database, ETL, replication, sharding, CAP theorem, E-R model, key-value database, column database, document database, graph database, SQL, Apache Cassandra, MongoDB, Neo4j, Redis, YCSB

Obsah

Seznam obrázků

Seznam tabulek

Seznam grafů

1 Úvod

Diplomová práce se zaobírá problematikou NoSQL databází. Termín NoSQL je zkratkou pro „Not only SQL“, z čehož vyplývá, že druhů těchto databází existuje celá řada. Zkoumané jsou hlavní kategorie spadající do této oblasti, a to konkrétně databáze typu klíč – hodnota, sloupcové databáze, dokumentové databáze a grafové databáze.

V úvodu je rozebrána historie databází jako takových a příchod SQL databází. Na to navazují nedostatky SQL databází, které se začínají projevovat s velkým nárůstem dat. Vzhledem k těmto faktům je jedna kapitola věnována i termínům jako jsou Big data, proces ETL, datové sklady a celkově technologie spadající do kategorie Business Inteligence.

Následně jsou popsány společné charakteristiky NoSQL databází a pojmy, které je nutno v této souvislosti znát, jako je replikace, sharding nebo CAP teorém.

Diplomová práce není rozdělena na teoretickou a praktickou část, protože se tyto části navzájem prolínají. V další části práce jsou rozebrány výše vypsané druhy databází. Ke každému druhu je uveden princip fungování, výhody těchto systémů, důvod vzniku a důvody proč SQL v těchto případech nevyhovuje. Ke každému druhu databáze je vybrán právě jeden zástupce, který danou kategorii reprezentuje. Na tomto zástupci je vždy popsán proces instalace databáze, spuštění databáze a praktická práce s databází zahrnující dotazy všeho druhu od ukládání, přes modifikace, mazání až po samotné dotazování se.

V závěru práce je provedeno šest porovnávacích testů na jednotlivých zástupcích za pomoci technologie Yahoo! Cloud System Benchmark. V závislosti na výsledcích těchto testů jsou napsána doporučení podle čeho se při výběru databázového systému rozhodovat, aby nedošlo k pozdějším komplikacím z důvodu špatného rozhodnutí.

2 Historie a vznik databází

Pro pochopení vzniku NoSQL databází a návaznosti vzniku jejich jednotlivých druhů na sebe je potřeba si vysvětlit historii databází jako takových. Jak bude popsáno v následující kapitole, je potřeba si vysvětlit vývoj databází, který vedl ke vzniku dnes již tradičních relačních databází založených na principu SQL. Stejně tak, jako se objevila potřeba SQL, tak se dnes čím dál tím více objevuje potřeba NoSQL databází.

2.1.1 Počátky

Hlavním důvodem vzniku databází byla potřeba ukládání dat. Již nestačilo data ukládat pouze do kartoték, protože s příchodem informační doby začaly přibývat jejich velké objemy.

Nejjednodušším způsobem ukládání dat byly **datové soubory**. Výhoda tohoto způsobu je ve snadné implementaci a pochopení toho, co je uloženo. Naopak nevýhoda pramení z nutnosti sekvenčního procházení dat a absence podpory paralelního přístupu do databáze. Způsob ukládání dat tohoto typu se v dnešní době stále používá, a to pro načítání konfigurace programů. [1]

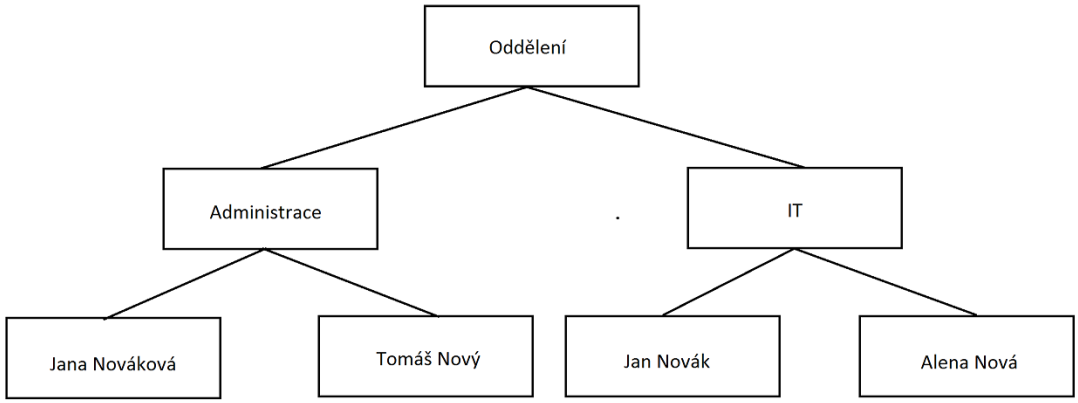
configuration.properties

```
# Enable feature to send updates. Possible values: true/false
SEND_UPDATES=true

# Application language. Possible values: czech/english
LANGUAGE=czech

# Log level. In case of problems. Enable debug level
LOG_LEVEL=info

```

Obrázek 1 Konfigurační soubor
Zdroj: Vlastní zpracování

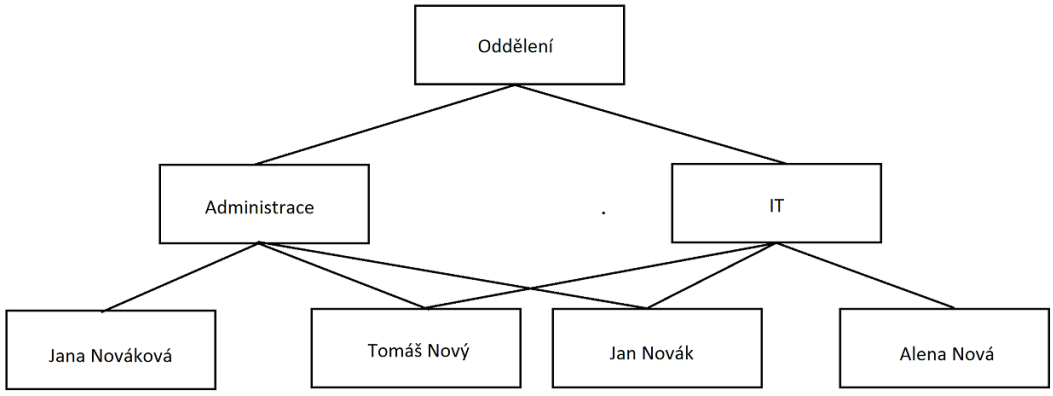
Poté přišel na scénu **hierarchický model** se stromovou strukturou. Jeden rodič má až N potomků a každý potomek může mít maximálně jednoho rodiče. Výhoda tohoto modelu je v jeho abstrakci. Uživatel databáze je odstíněn od konkrétního

fyzického uložení dat a může efektivně vyhledávat díky vztahům rodič-potomek. Naopak nevýhoda modelu vychází z potřeby nějaké změny databáze, kdy je nutné změnit celou databázi. Tento typ databáze se začíná zase znovu používat na webu jako alternativa k relačnímu modelu. [1]



Obrázek 2 Hierarchický model
Zdroj: Vlastní zpracování

Jako další přichází na trh **síťový model**, který rozšiřuje výše zmíněný hierarchický model o vazby M:N. Potomek tedy může mít více předků a tím vzniká místo stromu síť. Výhoda tohoto modelu spočívá v přirozenějším modelování vztahů mezi entitami. Naopak nevýhodou bylo její načasování, firma IBM věnovala své finanční prostředky do síťových vylepšení hierarchické databáze, místo aby se zaměřila na postavení síťové databáze. Ve stejné době vznikal také relační databázový model, kde uzly představují entity a orientované hrany představují vztahy. [1]

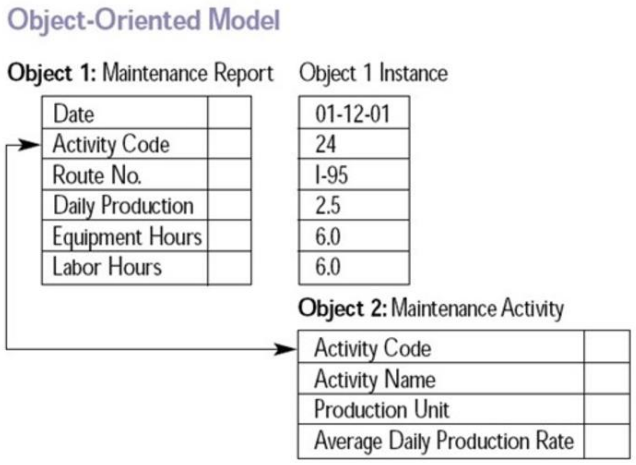


Obrázek 3 Síťový model
Zdroj: Vlastní zpracování

Na řadu přichází **relační model**, ten se od hierarchického a stromového liší tím, že nebyl postaven na principu předek-potomek. Relační model je postaven na matematickém aparátu relačních množin, který zjednodušil definici vazeb mezi entitami. Další přínos vzniku tohoto modelu je v důrazu na referenční integritu, tedy důraz na primární a cizí klíče. Vznikl také jazyk SEQUEL, který je předchůdcem jazyku SQL. Relační databáze jsou velice jednoduché na pochopení. Je možné si je totiž představit jako několik tabulek, které mají sloupce a řádky a jsou mezi sebou propojené relacemi (vazbami). Zástupci relačních databází jsou například: Oracle, MySQL, MSSQL. Podrobněji budou popsány dále. [2], [3]
Dále přichází na řadu **objektový model**, s objektovým jazykem a plnou podporovou objektových vymožeností, jako jsou dědičnost, polymorfismus apod. Existují dva hlavní proudy, které se snaží o implementaci objektů do databáze:

1. Evoluční – vychází se z toho, co je již hotové (relační databáze), a postupně se na tento typ databáze implementují objektové vlastnosti.
2. Revoluční – implementuje se vše od začátku.

Databáze objektové vycházejí z principů objektově orientovaného modelování a programování, příkladem jsou Caché, ObjectDB, GlobalsDB. Kombinací těchto dvou typů databází (objektové a relační databáze) vznikají objektově relační databáze. Jedná se o rozšíření relačního modelu o objekty pro manipulaci s novými datovými typy, například objektově relační databázový systém Oracle 8. [4]



Obrázek 4 Objektově orientovaný model
Zdroj: Zpracováno dle [47]

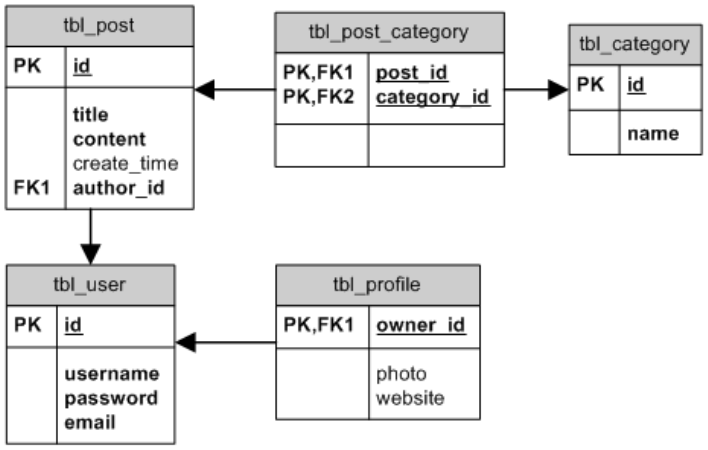
2.1.2 SQL

Jedná se o dotazovací jazyk, který je používán pro práci s daty v relačních databázích. Vznikl v 70. letech 20. století pro potřebu ovládání dat v databázích. Cílem bylo vytvořit jazyk, který by byl blízký jazyku přirozenému. Firma, která začala s vývojem jazyka SQL, byla IBM a později se přidaly i další společnosti. V roce 1979 přišla na trh databázová platforma Oracle Database od dnešní společnosti Oracle Corporation.

Databáze se staly velmi významnými a nepostradatelnými, proto bylo nutné jazyk standardizovat. [5]

Datový model

2.1.2 Při vytváření databázového schématu je nejprve nutné uvědomění si jednotlivých entit a jejich vzájemných vztahů, které se zobrazují na tzv. E-R diagramu. Výsledné schéma by mělo splňovat jistá pravidla, například třetí normální formu. U NoSQL databází se pravidla nedodržují. Na obrázku níže je možné vidět E-R diagram pro SQL databáze. Datový model se skládá z 5 tabulek, kde každá tabulka má svůj primární klíč zaručující jedinečnost. Tabulky jsou mezi sebou propojeny tzv. cizími klíči, přičemž cizí klíč jedné tabulky představuje primární klíč druhé tabulky, která je ve vztahu k jiné tabulce. Další atributy tabulek identifikují danou tabulku, například v tabulce uživatele najdeme atributy: uživatelské jméno, heslo nebo e-mail. [5]



Obrázek 5 E-R diagram
Zdroj: Zpracováno dle [5]

Tabulky

V databázovém prostředí se pracuje s databázovými tabulkami, které jako hlavní stavební prvky mají sloupce, řádky a jejich hodnoty. Tabulka představuje objekt reálného světa, zatímco sloupec představuje atribut daného objektu. Do řádků jsou pak ukládány jednotlivé hodnoty představující konkrétní objekt. Sloupce mají různé datové typy, které musí být respektovány, a nelze do nich ukládat hodnoty jiných datových typů.

Jako další stavební prvky se v databázovém prostředí pracuje s primárními, unikátními a cizími klíči. Primární klíč je speciálním identifikátorem každého řádku v tabulce. Cizí klíč pak funguje jako sloupec, do kterého se ukládá primární klíč spřízněné tabulky, čímž vzniká vazba. Vazby jsou vysvětleny v následující kapitole. [6]

Vazby

Důležitým faktorem relačních databází je propojení tabulek vazbami. Existuje několik typů těchto vazeb:

- 0:1 – v první tabulce může, ale nemusí existovat vázaný záznam ke druhé tabulce. Jestliže záznam existuje, musí být právě jeden odpovídající záznam ve druhé tabulce.
- 1:1 – první tabulka obsahuje záznam ke druhé tabulce a odpovídající záznam musí být právě jeden ve druhé tabulce.
- 0:N – v první tabulce může, ale nemusí existovat vázaný záznam k tabulce druhé. Jestliže záznam existuje, může být v druhé tabulce více než jeden.
- 1:N – v první tabulce může, ale nemusí existovat vázaný záznam ke druhé tabulce, odpovídajících záznamů v tabulce druhé může být více než jeden.

2.1.2.2

- N:M – v první tabulce může existovat více záznamů, které odpovídají více záznamům ve druhé tabulce.

Limitace

Nelze hovořit o limitaci jako takové, protože vlastnosti ACID nebo normální formy dělají relační databáze tím, čím jsou, avšak z pohledu NoSQL databází se jedná o nepřípustné vlastnosti, které zabraňují efektivní práci s nimi.

ACID

V případě klasické relační databáze, kde je potřeba znát velmi dobře strukturu ukládaných dat, jsou datové struktury rozděleny na malé kompaktní celky, které jsou pak dále uloženy v samostatné tabulce. Relační databáze zachovávají záruku konzistence dat tím, že implementují transakční zpracování dotazů a dodržují vlastnosti ACID – Atomicity, Consistency, Isolation a Durability. Tyto vlastnosti umožňují databázím bezpečné sdílení dat. Atomicita představuje vlastnost „všechno, nebo nic“. Pokud dochází k aktualizaci databáze, pak se tato aktualizace nazývá transakce. Transakce je nedělitelná operace, při níž jsou provedeny všechny kroky, neb nic. Konzistence je vlastnost, která zajistí, že jakákoli změna hodnot instance je v souladu s integritními omezeními. Izolace je vlastnost nutná v případě, že existují souběžné transakce vyskytující se ve stejné době. Trvanlivost je vlastnost, která zaručuje, že transakce, které jsou zpracovány, budou uchované natrvalo. [7]

Normální formy

Jedná se o techniku organizace dat v databázi. Je to systematický přístup dekompozice do tabulek za účelem zamezení redundance dat. Optimální strukturu databáze zajistí tři úrovně. [8] [9]

- První normální forma – Jedná se o definici datových položek, to znamená, že se určí jednotlivé tabulky, jejich sloupce a datové typy.
- Druhá normální forma – Zamezení opakování se dat, to znamená, máme-li tabulku „Uživatel“, u něhož evidujeme fakturační a domácí adresu, pak aby byla splněna druhá normální forma, je zapotřebí vytvořit novou tabulku „Adresa“, která bude ukládat jednotlivé informace o adresách a za pomoci cizího klíče bude spojena s tabulkou „Uživatel“.
- Třetí normální forma – Navazuje na druhou a týká se toho, že každá tabulka by měla mít svůj primární klíč, na který se pak mohou odkazovat cizí klíče.

2.1.3 NoSQL

Z důvodů nových přístupů přestávají koncepty SQL postupně stačit, vzhledem k přísné struktuře databáze, dodržování jednotlivých normálních forem nebo dodržování konceptů ACID.

V dnešních aplikacích se klade důraz na cloud, škálování, decentralizaci dat, grafové ukládání, kde jsou ukládány entity jako vrcholy a spojnice jako vazby, nebo problematické datové typy, jako jsou key-value uložiště, nestrukturované dokumenty nebo třeba RDF grafy.

Jako odpověď na tyto požadavky přišly různé druhy databází založené na často zcela odlišných principech. Tyto databáze se řadí do kategorie NoSQL

V první řadě je potřeba vysvětlit, že NoSQL neznamená NO SQL, ale spíše se jedná o zkratku vystihující „Not only SQL", tedy nejen SQL. Jde o sadu databázových přístupů a konceptů, kde jsou data ukládána pomocí jiných prostředků než u standardních SQL databází. Může jít od nejjednodušších přístupů, jako jsou jednoduché key-value databáze, až po robustní projekty, jako jsou třeba grafové databáze. Při výběru takové NoSQL databáze je zapotřebí důkladné analýzy požadavků a následného vyhodnocení, který druh databáze je pro konkrétní aplikaci nejvhodnější. Motivace použití jsou různé, a to od jednoduchého designu databáze až po širokou možnost horizontálního i vertikálního škálování.

Další důležitou informací, kterou je potřeba zmínit, je fakt, že NoSQL databáze nemají za úkol nahradit již standardizované SQL databáze. Jejich úkolem je tyto databáze doplnit, popřípadě je nahradit v situacích, ve kterých jsou SQL databáze nedostačující, nebo dokonce nevyhovující.

Za hlavního představitele NoSQL databází je považován Carlo Strozzi, který s tímto pojmem přišel v roce 1998. [10]

Jako výhody NoSQL databází uvedl, že nemají žádné pevné schéma, jsou schopné zvládnout zpracování velkých objemů dat lépe a mají nižší náklady kladené na server. Naopak za nevýhody považoval, že jsou nestrukturované, nemají v podstatě žádná pravidla a omezení, nejsou dostatečně vyzrálé, mají silnou nekonzistenci dat, replikace velkých databází může selhat a nemají žádný standardní jazyk, jako je SQL. [11]

Další, kdo tento pojem použil, byl Eric Evans v roce 2009. Od této doby se termínem NoSQL označují nové databázové systémy, které vznikly jako podpora zpracování big data. Tyto databáze by se daly popsat jako databáze pro správu velkého množství dat, které jsou ve větší míře ne-relační, distribuované, škálovatelné a podporují replikaci. Jsou to databáze, které nemusejí mít žádné datové schéma, mají velmi jednoduché rozhraní pro práci s daty a open source přístup.

Naopak od konceptu objektových databází, který se doposud moc neprosadil, se koncept NoSQL databází velmi dobře prosazuje a rozšiřuje, protože přichází jako odpověď na praktické potřeby dnešní internetové společnosti. NoSQL technologie jsou vyvíjeny dvěma způsoby, a to interně pro potřeby velkých společností, jako je Google, Amazon nebo Facebook. Druhým způsobem je pak open source vývoj zprostředkovaný různými komunitami nadšenců nebo malých firem.

Hlavní výzvy pro NoSQL databáze:

- Zralost – NoSQL databáze jsou velmi efektivní, ale prozatím nejsou natolik prověřené jako relační databáze.
- Uživatelská podpora – za výhodu, ale zároveň nevýhodu je považován fakt, že tyto databáze vznikly v rámci start up projektu a jsou open source. Chybí poskytovatel s dobrým jménem a zázemím.
- Administrace – NoSQL databáze mohou být složitější na instalaci, konfiguraci a následnou údržbu.
- Standardizace přístupu k datům – ve srovnání s relačními databázemi, které nabízejí přístup pomocí jazyka SQL. U NoSQL si každá databáze vyvinula vlastní dotazovací jazyk a programátorské rozhraní. Pokud jsou dotazy složité, jedná se o velmi netriviální programátorské znalosti.
- Experti – na trhu je velký nedostatek odborníků.

Velmi důležité je však podotknout, že NoSQL databáze nemají za úkol nahradit nám dobře známé relační databáze a vlastně ani jiný typ databází. Jejich cílem je nabídnout řešení pro nové typy aplikací. Spousty let vývoje přinesly spoustu možností efektivních technologií využívaných v relačních databázích. Relační

databáze mají řadu výhod, které je dělají nepostradatelnými. Těmito výhodami jsou nepochybně jejich dlouholeté použití v komerčních projektech, stabilita, spolehlivost a robustnost, existence standardizovaných rozhraní a dotazovacích jazyků pro komunikaci s nimi a v neposlední řadě také silná konzistence dat. NoSQL databáze prozatím tyto vlastnosti nemá, ale dá se předpokládat, že postupem času se situace může změnit.

Na relační a NoSQL databáze je potřeba nahlížet jako na dvě možnosti způsobu ukládání dat, přičemž každá databáze je určená pro jiný druh aplikací. [12]

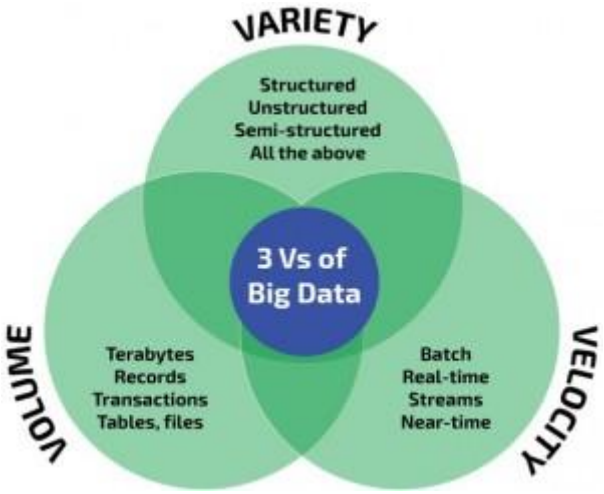
Tabulka 1 Porovnání relační a NoSQL databáze

Relační databáze	**NoSQL databáze**
Integrita dat je zásadní.	Stačí, pokud je většina dat většinu času v pořádku.
Datový formát je konzistentní a dobře definovaný.	Datový formát nemusí být známý nebo konzistentní.
Předpokládáme dlouhodobé uložení dat.	Vzhledem k velkému množství dat často ukládáme pouze určité „časové okno" (např. poslední měsíc, poslední rok).
Aktualizace dat jsou časté.	„Write-once/read-many", tedy vložená data už typicky nejsou dále modifikována (nebo alespoň ne příliš často). Obvykle data neustále přibývají, aniž by byla modifikována. Již nepotřebné záznamy jsou pak smazány.
Předvídatelný (lineární) nárůst velikosti dat.	Nepředvídatelný (exponenciální) nárůst velikosti dat.
Nástroje pro dotazování dat umožňují přístup i ne-programátorům.	Typicky pouze programátoři píší (implementují) zpracování dat.
Probíhající pravidelné zálohy dat.	Pro řešení výpadků je využívána replikace dat.
Přístup k datům zajišťuje jediný server.	Data jsou umístěna na více serverech, přistupujeme tedy ke clusteru uzlů.

Zdroj: Zpracováno dle [12]

3 Big data

Jak už název napovídá, jedná se o velké množství dat, u nichž lze velikost jen těžko definovat. Přesto je možné pár definic termínu big data nalézt. Například Margaret Rouse uvádí definici: *„Big Data je vyvíjející se termín, který popisuje jakékoliv množství strukturovaných, semistrukturovaných a nestrukturovaných dat, která mají potenciál k tomu, aby z nich byly vydolovány nějaké informace.“* [13]
Společnost Gartner uvádí definici: *„Data, jejichž velký objem (high-volume), velká rychlost nárůstu (high-velocity) a velká různorodost (high-variety) neumožňují zpracování za pomoci již známých a z praxe ověřených technologií a přístupů.“* [14]
Tyto tři vlastnosti jsou dnes již běžné známé jako tzv. „3V model", viz obrázek č. 6.



Obrázek 6 3V model
Zdroj: Zpracováno dle [44]

Jak lze z obrázku vyčíst, jedna taková Big Data kolekce se může skládat z dat různých formátů, jako jsou transakce, tabulky nebo třeba soubory různé velikosti. Dohromady všechna data mají takovou velikost, že se nevejdou na jeden databázový server, nýbrž na desítky či stovky databázových serverů a nárůst počtu dat v jedné Big Data kolekci může být často až exponenciální, proto je kladen důraz na rychlost zpracování. Různorodost dat odkazuje na definici Margaret, kde lze vidět, že je dána často zcela odlišnou strukturou dat. To znamená, že data mohou přijít ve formátu JSON, XML nebo také jako multimediální soubor. Způsob uložení

dat do takové kolekce také není jednotný, naopak data mohou být ukládána například v reálném čase, pravidelně odesílaných dávkách nebo ve streamech, což je specifický zdroj dat, který nemá začátek ani konec, nelze ho opakovaně procházet a z hlediska rychlé transformace nemá smysl ho celý ukládat.

Mezi zdroje, které produkují data vhodná pro Big Data kolekce, patří například sociální sítě, elektronické senzory, obchodní portály nebo také strojově generovaná data, jako jsou například data a analýzy z obchodování na burze. Pro představu velikosti Big Data lze skvěle využít schéma od společnosti IBM, kde je model obohacen o jedno další „V“, a to Veracity, což je důvěryhodnost dat. Společnost IBM odhaduje, že v roce 2020 bude vlastnit mobilní telefon šest miliard lidí, denně bude vyprodukováno dva a půl trilionu dat a celkově na discích bude uloženo čtyřicet zettabytů dat.

[15]

„Další uváděný význam „veledat“ spočívá v tom, že umožňují uvnitř množit informace a mezi nimi pozorovat a pochopit vztahy, kterým jsme dosud nerozuměli.“ [16]

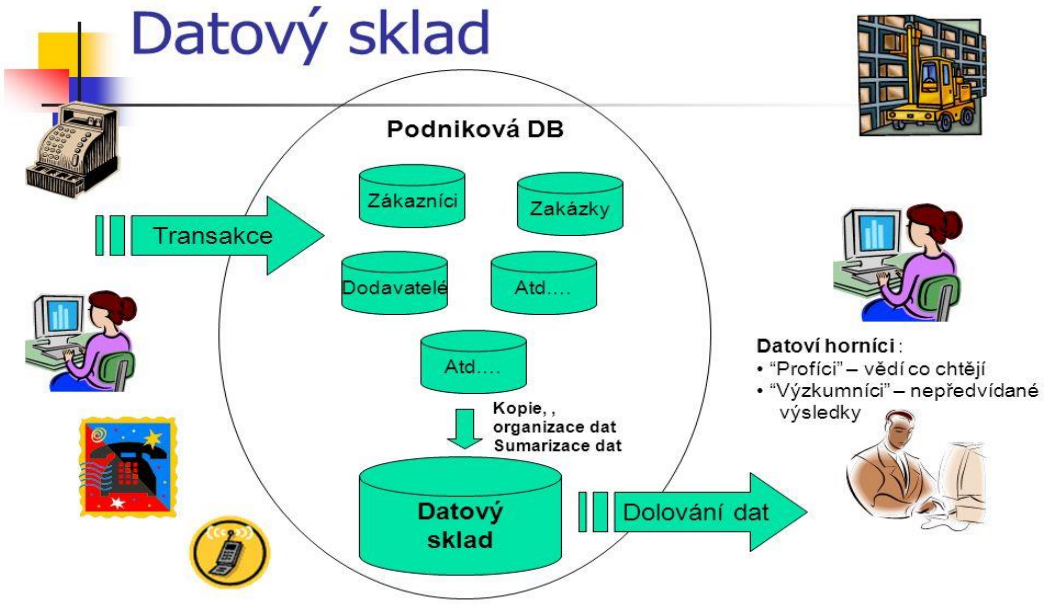
Veledata přinášejí tři posuny uvažování. První přináší schopnost analyzovat rozsáhlé objemy dat o předmětu, aniž by bylo potřeba se smířit s menšími podmnožinami. Druhý posun uvažování přijímá neuspořádanost dat z reálného světa a opouští důraz na přesnost. Třetí posun uvažování přináší větší spoléhání na korelace.

Sbírání a shromažďování dat je dnes zcela běžné, dříve se s tímto úkolem dokázaly vypořádat pouze velké instituce, například stát nebo církev. Nejstarší záznamy zpracování dat pocházejí z roku 5000 př. n. l. Jednalo se o data sumerských obchodníků, kteří zaznamenávali prodávané zboží na hliněných destičkách. Zpracování dat jednotlivými státy se týkalo sčítání lidu. Již ve starověkém Egyptě a Číně docházelo ke sčítání obyvatel. V minulých dobách bylo toto měření velmi časově, ale také finančně náročné, přesto výsledné informace byly jen přibližné. Z tohoto důvodu statistici zjistili, že přesnost vzorkování se zvyšuje při náhodném výběru, nikoli při zvětšování vzorku. Je důležité vybrat takový vzorek, který by mohl dobře reprezentovat celou populaci. Vznikla tedy mnohem levnější metoda shromažďování dat, která přinášela velkou přesnost a možnosti extrapolovat výsledek na celek. Tato metoda se rozšířila i do soukromého sektoru, například

v podnicích jí byla zjišťována kvalita výrobků. Přestože se vzorkování stalo velmi úspěšným a používaným, bylo třeba myslet na to, že i tento způsob má některé nedostatky. Přesnost vzorkování závisí na zajištění náhodnosti při výběru dat vzorku a takové náhodnosti se dosahuje velmi obtížně. Systematické odchylky při sběru dat mohou vést k velkým chybám ve výsledcích zpracování extrapolovaných dat. Například při volebních průzkumech jsou problémy tohoto typu velmi zjevné. Když je možné shromažďovat velké objemy dat, princip vzorkování přestává dávat smysl. Nyní se v mnoha technologických a společenských aspektech začíná dávat přednost více neuspořádaným datům před menším množstvím přesných informací. Veledata nám pomáhají přiblížit se k realitě mnohem více, než když jsme byli závislí na malých datech s přesností. [16]

3.1 Datové sklady

Velké objemy dat jsou s nejvyšší pravděpodobností ukládány do datových skladů, kam se dostávají z nejrůznějších zdrojů a informačních systémů pomocí procedur ETL. Ve většině případů se jedná o data ve strukturované podobě. Data se do datových skladů přenáší v určených časových cyklech a poté je nad nimi provedena analýza. [17]



Obrázek 7 Datový sklad
Zdroj: Zpracováno dle [45]

Pojem datový sklad má několik významů. Nejjednodušeji se tento pojem dá pochopit jako komplexní data uložená ve struktuře, která umožňuje analýzu a dotazování. Jak ukazuje obrázek výše, data přicházejí z jednotlivých uskutečněných transakcí a ukládají se do datových skladů. Poté přichází na řadu dolování dat z těchto skladů a poslední fází je analýza nad upravenými a očištěnými daty. [17]

3.1.1 ETL

ETL neboli **extract, transform a load**, kde extract je proces čtení dat z databáze a transformace je proces konverze získání dat v jejich původní podobě. Transformace probíhá s použitím pravidel, převodní tabulky nebo kombinací dat s jinými daty. Load je proces zápisu dat do cílové databáze. ETL se používá k migraci dat z jedné databáze do druhé pro vytvoření datových tržišť a datových skladů a také pro převod databáze z jednoho formátu do druhého. [18]

Pojem datové tržiště je vysvětlováno jako databáze nebo kolekce databází, jejímž cílem je pomoci manažerům přijímat strategická rozhodnutí týkající se jejich podnikání. Vzhledem k tomu datový sklad kombinuje databázi v rámci celé firmy. Datová tržiště jsou obvykle menší a zaměřují se na určitá oddělení. Některá datová tržiště jsou podmnožinami větších datových skladů. [19]

3.1.1.1

Keboola Connection a GoodData

Jedná se o dvojici nástrojů business inteligence, které umožňují extrahovat data z několika různých zdrojů a potom z nich automatizovaně zpracovat výstupy. Keboola Connection dokáže do sebe natáhnout jakákoliv data, například data z Google Analytics, Adwords, informace o zákaznících z firemních CRM, e-shopu nebo support systémů. Nejprve je potřeba na začátku naimplementovat a nakonfigurovat dané připojení a extrakci a poté už vše běží automatizovaně. Keboola si data profiltruje, zkombinuje a pospojuje a tím vzniká jednotný reporting. Pro vizualizaci dat je možné použít jakýkoli nástroj pro vizualizaci, např. PowerBI, Tableau nebo GoodData. Data připravená k vizualizaci se nahrají do

některého z výše uvedených nástrojů. Poté přichází zpracování dat do grafů, tabulek apod.

Keboola funguje na principu ETL. Obsahuje komponenty jako „Extractors“, což je komponenta sloužící k natažení dat z různých zdrojů, jako jsou Google Analytics, AWS, různé SQL databáze apod. Poté přichází následující krok transformace, ke které slouží komponenta „Transforms“, kde je navolen input mapping, což mohou být vstupní tabulky, ze kterých pomocí scriptů napsaných v SQL nebo ve skriptovacích jazycích, jako je Python, vznikne output mapping neboli výstupní tabulky. Výsledné tabulky pomocí „Writters“ jsou tzv. nataženy do vizualizačního nástroje. Pomocí komponenty „Orchestrations“ se nakonfiguruje, jak často se data mají do systému Keboola Connection nahrávat.

V případě využití GoodData jako „writeru“ lze vypracovat nástěnku nebo tzv. dashoard, který je znázorněn na obrázku níže. Na dashboardu lze vykreslovat grafy, tabulky a jiné statistické prvky. [20]

Obrázek 8 Ukázka dashboardu v GoodData
Zdroj: Zpracováno dle [20]

3.2 Kombinovaná data

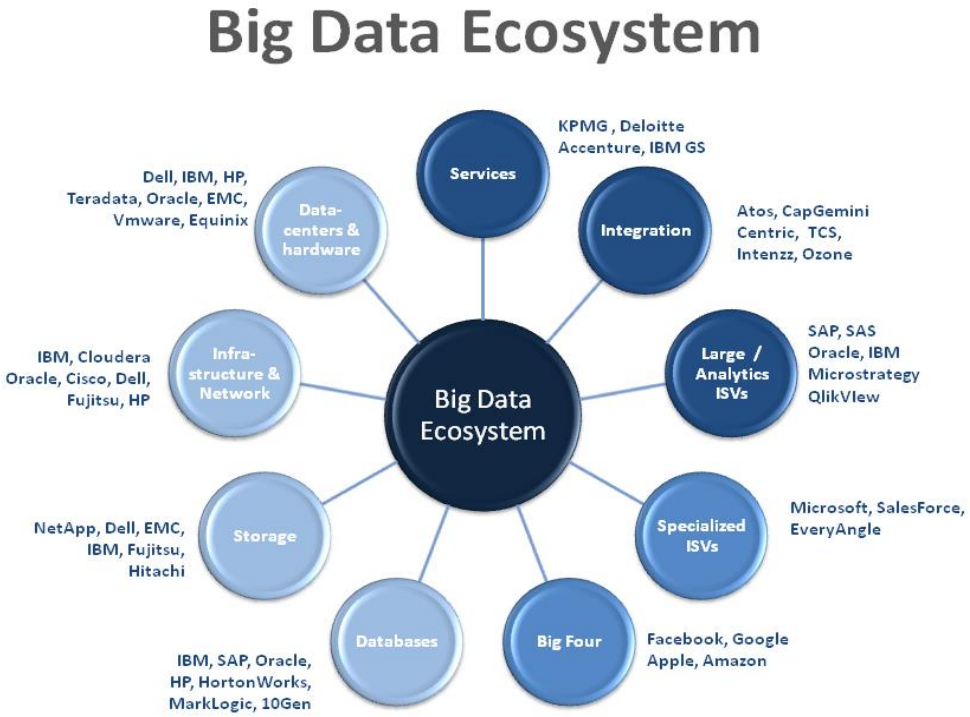
Zdá se, že po shromáždění a analýze jsou data zcela bezcenná a nedají se znovu použít. To však dnes zcela neplatí, protože získaná data mají skrytou hodnotu a tu lze využít kombinací jedné datové množiny s jinou. Je možné propojovat jednotlivé množiny dat a díky tomu získat neobvyklé výsledky.

3.3 Rizika

V dnešní době jsou lidé pod neustálým dohledem, o každém člověku je shromažďováno velké množství dat. Existuje neustálý dohled nad jednáním lidské populace, například při placení bankovními kartami nebo komunikaci pomocí mobilních telefonů. Lidé nejsou špehováni pouze vládními tajnými agenturami, ale také například nadnárodními společnostmi, jako je Amazon, který monitoruje nákupní preference, nebo společnost Google, která zase monitoruje oblíbenost internetových stránek. Dá se konstatovat, že veledata mají svou stinnou stránku, a sice že ohrožují soukromí každého jedince. [16]

3.4 Budoucnost

Budoucnost podnikání dnes závisí na big datech, podniky potřebují znát chování zákazníků, jejich potřeby a zájem. Možné využití big dat je i ve zdravotnictví, například ke zlepšení zdravotní péče. Firmy je budou muset v budoucnosti využívat jejich výhody, pokud budou chtít nadále zůstat konkurenceschopné. [21]
Big data jsou budoucností i pro středně velké podniky. Pokud firmy budou umět jenom sbírat data, a ne s nimi pracovat, budou jim tato data vlastně k ničemu. Než se začne analyzovat, je potřeba si položit otázku, co vlastně firmy chtějí zjistit? Historická data ukrývají budoucnost a pomocí prediktivní analýzy se tato budoucnost dokáže odhalit dříve, než ji odhalí konkurence. [22]



Obrázek 9 Big Data Ecosystem
Zdroj: Zpracováno dle [21]

4 Základní principy NoSQL databází

Dříve platilo, že pod pojmem databáze bylo možné si představit centralizovanou relační databázi. Postupem času vznikaly nové typy databází, ale většina se neprosadila tak široce, jako klasické relační databáze. V poslední době se rozvíjejí požadavky na systémy pro správu dat a tím vznikly nové typy databázových technologií. Díky tomu se dnes mluví o konkrétních databázových systémech, přičemž každý z těchto systémů je unikátní svou nabídkou funkcí a vnitřním fungováním. Dnes se již tedy nebavíme o databázích, protože pod tímto pojmem se dá představit velké množství typů a přístupů. Samozřejmě i NoSQL databáze mají své společné principy.

U NoSQL databází se vynaložilo větší úsilí při návrhu struktury dat a při jejich ukládání proto, aby vyhodnocení dotazů bylo rychlejší a systém zvládl vyhodnotit více dotazů najednou. U NoSQL jsou specializovaná úložiště, která umožňují seskupování různých záznamů do celků, na které se bude ve většině případů přistupovat společně, a dále replikaci dat na více uzlech.

NoSQL mají výhodu oproti tradičním databázím ve flexibilitě schématu uložených dat. U relačních databází je také možné měnit schéma jednotlivých tabulek, ale problém je v tom, že tyto operace je možné dělat pomocí jazyka pro definici dat - DDL, ale tento jazyk není přístupný všem, protože není součástí sady příkazů. Změnu tedy může provádět jenom administrátor systémů. Schémata spravovaných dat jsou u NoSQL velmi flexibilní. Každá NoSQL databáze má úroveň flexibility jinou. [12]

Pro pochopení souvislostí a nutnosti existence takového množství druhů nejen NoSQL databází je potřeba si vysvětlit několik základních termínů a principů.

CAP theorem

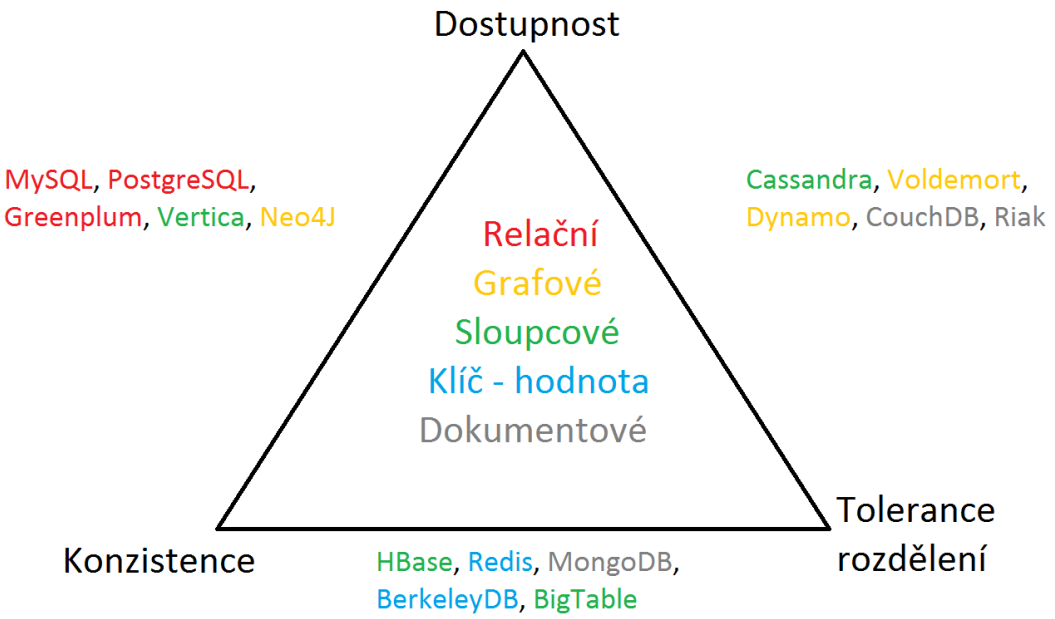
Tento pojem byl poprvé prezentován americkým vědcem Ericem Brewerem v roce 2000 na konferenci „Principles of Distributed Computing".

Problematika teorému se úzce váže k NoSQL databázím, kde je běžně jeho dostupnou definicí věta: „Pro distribuovaný systém není možné splnit více jak dvě ze tří garancí.“ Zmíněnými garancemi jsou myšlena následující měřítka:

- **Konzistence** (Consistency) – Konzistencí je myšlen fakt, že systém při čtení vrátí poslední zápis.
- **Dostupnost** (Availability) – Dostupností je myšleno, že systém vrátí odpověď na každý požadavek. Zde se setkáváme s problémem, že v některých situacích nelze zaručit vždy aktuální hodnotu daného záznamu, protože daná část distribuovaného systému nemá ještě aktuální data, např. z důvodu síťového problému.
- **Tolerance rozdělení** (Partition tolerance) – Systém dokáže zpracovat požadavky navzdory tomu, že došlo k rozdělení sítě, např. z již výše zmíněného síťového problému.

Vzhledem k tomu, že v podstatě nelze zajistit stoprocentní stabilitu sítě, je vždy nutné počítat s možností síťového rozdělení. A jestliže dojde k síťovému rozdělení, je potřeba se rozhodnout, jestli má systém odmítnout požadavek, nebo vrátit potenciálně neaktualizovaná data. [23], [24]

Na konferenci Geecon 2017 v polském Krakově měl DuyHai Doan přednášku, ve které prezentoval CAP teorém obohacený o jednotlivé databáze, kde je krásně vidět, na co se podle zmíněných garancí CAP teorému která databáze hodí. Předělaný obrázek si můžete prohlédnout zde.



Obrázek 10 CAP teorém
Zdroj: Vlastní zpracování, podle DuyHai Doana

Replication
Replikace je zkopírování kompletně celé databáze na další server nebo servery. Replikace převážně slouží k vylepšení rychlosti přístupu a zálohování dat. Může se dále dělit na synchronní a asynchronní replikaci. Nejčastějším modelem replikace je model master-slave, kde jeden uzel vystupuje jako hlavní a druhý jako podružný. [25]

Partitioning
Partition by se dalo přeložit jako oddíl, nicméně pojem partitioning je tak zažitý, že ho nebudu překládat. Jedná se o technologii, která umožní rozložit data z jedné tabulky do menších částí na základě definovaného členění. Typickým příkladem může být nějaký systém fungující po celém světě, ukládající data o uživatelích, kdy pro každý kontinent / stát existuje vlastní databázový uzel. Uživatelé jsou pak ukládáni do uzlu, který odpovídá jejich bydlišti. Uživatelé z jednoho kontinentu / státu se pak připojují do jednoho uzlu, čímž je zajištěna lepší rychlost odezvy a menší zátěž serveru. Mezi další výhody se řadí možnost spouštění agregačních funkcí ve více procesech, kde počet procesů se rovná počtu uzlů, a následné sloučení výsledků, čímž opět dosáhneme rychlejšího výpočtu.

4.1 Key value databáze

Jedná se o druh NoSQL databáze, která používá metodu jednoduchý klíč- hodnota pro ukládání dat. Klíčová hodnota se vztahuje ke skutečnosti, že databáze ukládá data jako kolekci dvojic klíč-hodnota. Jedná se o jednoduchý způsob ukládání dat. Klíč-hodnota je dobře zavedená koncepce v mnoha programovacích jazycích, např. Mapy v Javě. Programovací jazyky typicky odkazují na klíč-hodnotu jako asociativní pole nebo datové struktury. Klíčová hodnota je také běžně označovaná jako slovník nebo hash.
Jednoduchý příklad – telefonní seznam:

Tabulka 2 Příklad databáze klíč-hodnota

Klíč	Hodnota
Jan	(420) 123456789
Jana	(420) 987654321
Petra	(420) 123456788
Petr	(420) 123456799

Zdroj: Vlastní zpracování

Klíč

Klíč musí být unikátní. Jedná se o jedinečný identifikátor, který umožní přístup k hodnotě s ním spojené. V teorii klíč může být cokoliv, ale může se to lišit od druhu DB. V databázi REDIS je například maximální povolená velikost klíče 512 MB. Je možné použít jakoukoliv binární sekvenci jako klíč. Z výkonnostních důvodů by klíč neměl být příliš dlouhý. Příliš krátký klíč zase může způsobit problémy s čitelností. Klíč by měl dodržovat dohodnutá pravidla s cílem udržet věci konzistentní.

Hodnota

Hodnota může být cokoli, například text, značkovací kód jako HTML, programovací kód, obrázek, seznam nebo i jiný klíč – hodnota zapouzdřená v objektu.

Databáze typu klíč-hodnota mohou být použity pro mnoho scénářů, mezi které se řadí například uživatelské profily, články/komentáře, e-maily nebo z prostředí e-commerce situace jako nákupní košík, kategorie produktů, detaily produktů nebo třeba recenze produktů. Tyto databáze mohou také ukládat celé webové stránky, za pomoci URL jako klíče a webové stránky jako hodnoty. [26], [27]

Mezi databáze typu klíč-hodnota se řadí databáze Voldemort, Oracle Berkeley DB, Aerospike nebo Redis, který bude rozebrán v následující kapitole.

Redis

Jedná se o klasickou key-value databázi, kde pod daný klíč můžeme uložit různé datové struktury. Je dostupná jako open source a jedná se o jednu z nejpopulárnějších key-value databází. Redis je napsán v jazyce ANSI C a funguje ve většině POSIX systémů, jako jsou GNU/Linux, *BSD nebo OS X bez externích závislostí. Redis je vyvíjen právě na OS X a GNU/Linux a je doporučeno používat ho právě na těchto systémech. Je jím podporována široká škála programovacích jazyků, jako jsou Bash, C, C#, C++, PHP, Java, Perl, Objective-C a desítky dalších. [28]

Redis je převážně využíván na sociálních sítích, v analytických nástrojích, finančních nástrojích nebo třeba herním průmyslu, a to nejčastěji jako cache, fronta zpráv, pro analýzy v reálném čase nebo jako fultextové vyhledávání.

Vzhledem k povaze návrhu databáze je Redis nejčastěji využíván jako cache nebo fronta zpráv.

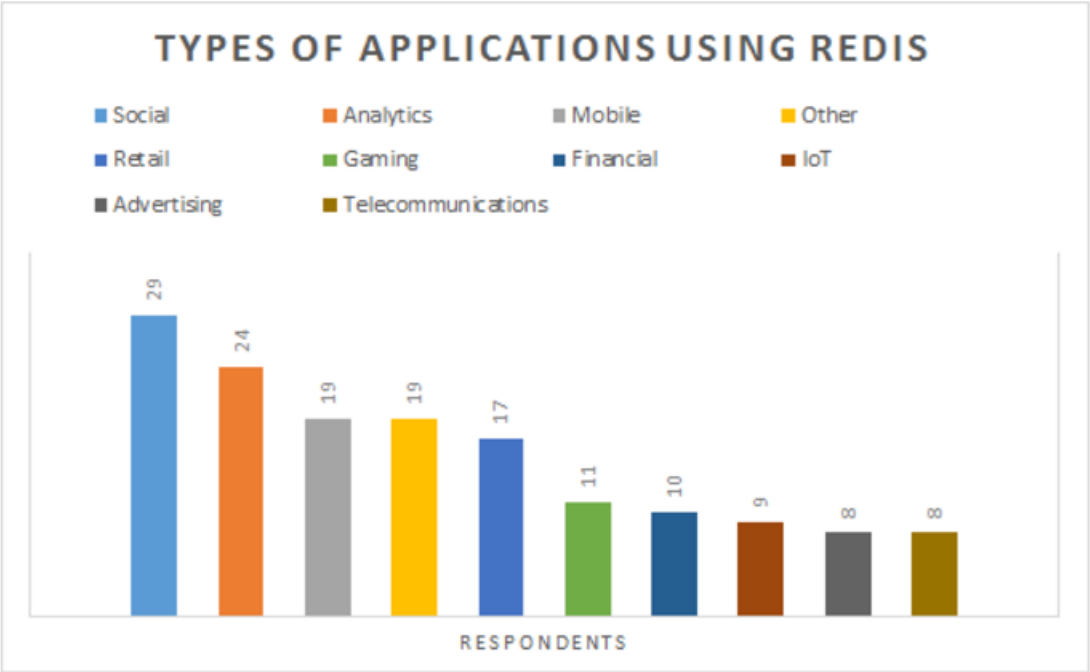
Pod jeden klíč je možné uložit několik datových struktur. Data jsou ukládána primárně v paměti, proto je Redis velmi rychlý, obsah paměti je ukládán na disk, takže je velká pravděpodobnost, že se o data nepřijde. Pracuje primárně s pamětí RAM.

Pokud dojde paměť a všechny hodnoty jsou již uloženy na disku, databáze Redis odmítne vytvořit nový klíč. [30]

Redis podporuje mimo jiné několik dalších využitelných funkcionalit, jako jsou transakční zpracování, lua skripty, klíče s omezenou dobou, automatický failover a další funkcionality využitelné v běžné praxi. Pro testování, učení se a hraní si s databází existuje dokonalý nástroj dostupný na https://try.redis.io, kde si lze databázi vyzkoušet. Je také možné využít možnosti návodu, který uživatele provede základními příkazy. [28]

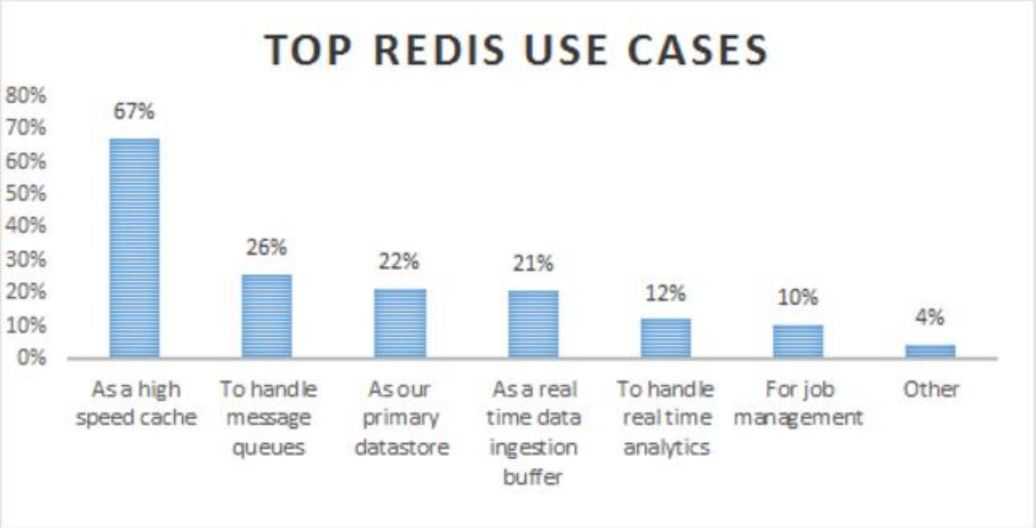
Společnost Redis udělala výzkum v roce 2015, ve kterém zjišťovala celkem od 116 uživatelů několik základních informací, které pak zpracovala do grafů.

Na prvním grafu můžeme vidět softwarová odvětví, ve kterém je Redis používán a kterých je celkem 10.



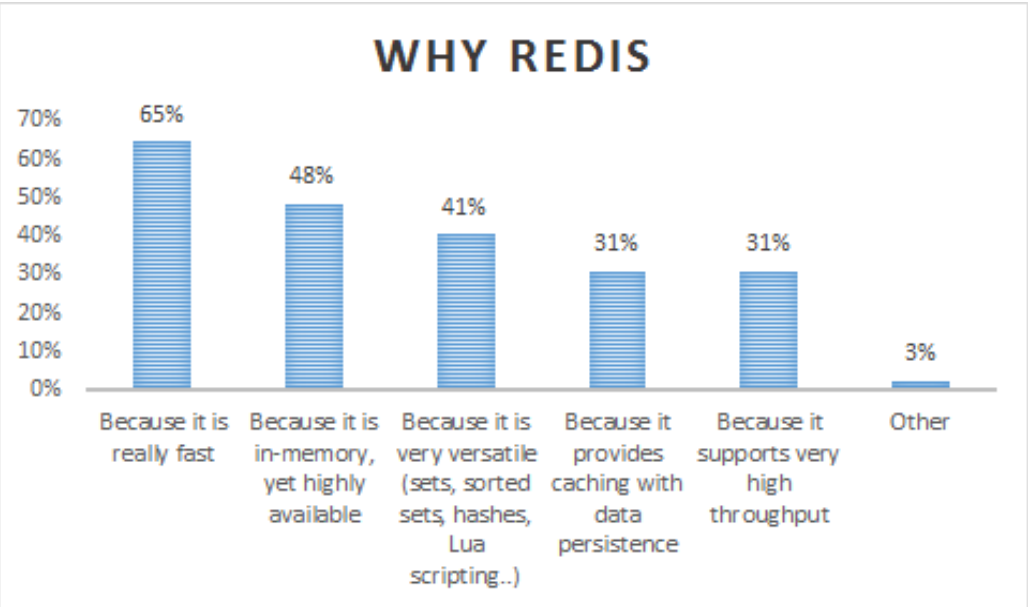
Obrázek 11 Oblasti využití REDIS
Zdroj: Zpracováno dle [29]

Na druhém grafu můžeme vidět účel použití tohoto druhu databáze v aplikacích. Suverénně nejčastěji byla používána jako cache, viz obrázek č. 12.



Obrázek 12 Účel využití REDIS v aplikacích
Zdroj: Zpracováno dle [29]

Na posledním grafu jsou vidět důvody použití této key value databáze. Na prvním místě byla jako důvod udávána rychlost jako další skutečnost, že i přes možnost perzistence je stále vysoce dostupná v operační paměti. Dalším důvodem je fakt, že Redis má zabudovanou replikaci, transakce a další způsoby uložení na disk. [29]



Obrázek 13 Důvody použití REDIS
Zdroj: Zpracováno dle [29]

4.1.1 Použití v praxi

V této podkapitole bude uveden výběr společností, které využívají databázi Redis, na co ji používají a proč se rozhodly jít touto cestou. Mezi zákazníky této databáze se řadí společnosti, jako jsou například Twitter, Weibo, Pinterest, Snapchat nebo Digg.

GitHub

GItHub využívá Redis pro perzistentní key value úložiště sloužící k uložení směrovacích informací a dalších dodatečných dat ohledně směrování. [28]

Craighlist

Společnost Craiglist slouží jako reklamní stránka, kde může inzerovat každý. Inzeráty jsou rozděleny do kategorií zaměstnání, bydlení, osobní inzerce, služby, koncerty, diskuzní fóra a pár dalších. Redis je využíván k tzv. shardingu, což je způsob partitioningu rozdělujícího data mezi jednotlivé stroje, kterým se říká shardy. Podle typu se data uloží do všech shardů nebo pouze do specificky určených. Typickým příkladem může být celosvětová aplikace rozdělující uživatele podle geografického umístění. Jejich infrastruktura je založená na uzlech, kde jeden uzel je definován klíčem „ip:port" a odkazuje na jednu instanci Redis databáze, do níž jsou data ukládána. [28]

StackOveflow

StackOverflow je webová aplikace, která slouží programátorské komunitě ke sdílení problémů a jejich řešení a která využívá Redis jako cache pro celou síť. Cachována je každá stránka navštívená každým uživatelem včetně anonymních. Nejpomalejší částí procesu je pak samotné ukládání bytů do sítě. Většina cachovaných dat je datového typu string, takže i přes nízké využití CPU je dosaženo dobrých kompresních poměrů. [31]

Flickr

Flickr je komunikativní web pro sdílení fotografií. Společnost Flickr pro změnu využívá převážně datový typ List a používá ho jako frontu pro updaty. Flickr zvolil Redis z důvodu rychlosti, a to i přes riziko ztráty některých dat. To mu ale nevadí

z hlediska využití Redisu jako „firehose" služby, což ve zkratce znamená API poskytující nonstop přístup ke streamu dat. [28]

4.1.2 Instalace databáze

Oficiální podpora existuje pouze pro operační systémy typu GNU/Linux. Instalace je standardně prováděna podle balíčkového manageru daného operačního systému.

Pro operační systém Windows neexistuje oficiální podpora, nicméně společnost Microsoft Open Tech vyvíjí a udržuje distribuci pro Windows. [32]

Po instalaci máme k dispozici několik základních spustitelných souborů pro práci s databází. Jedná se o soubory:

- redis-server – nastartuje server s konfigurací nastavenou ze souboru redis.conf;
- redis-sentinel – určený pro monitoring a failover;
- redis-cli – klient sloužící pro komunikaci se serverem;
- redis-benchmark – slouží pro kontrolu výkonu;
- redis-check-aof – užitečný při řešení událostí s poškozenými datovými soubory;
- redis-check-dump – užitečný při řešení událostí s poškozenými datovými soubory.

[28]

4.1.3 Práce s databází

Jak již bylo řečeno, Redis nabízí několik základních datových typů, které budou následně vysvětleny a popsány.

Datové typy

Jednoduchá hodnota

Pro uložení základní hodnoty, jako je číslo nebo řetězec, slouží příkazy „GET" pro získání a „SET" pro uložení hodnoty. Pro číselné hodnoty pak lze používat příkazy jako „INCR" nebo „DECR" pro snižování, resp. pro zvyšování číselné hodnoty. [28]

List

Pro práci s listem slouží příkazy „RPUSH" a „LPUSH" pro uložení prvku na konec listu, resp. na začátek listu. V případě, že list neexistuje, je vytvořen. Příkazem „LRANGE" vytahujeme sublist listu. Příkaz „LRANGE" přebírá dva parametry. První slouží jako první index sublistu a druhý parametr jako koncový index sublistu. Hodnota -1 v druhém parametru znamená konec listu. Příkaz „LLEN" vrací velikost listu. „LPOP" smaže a vrátí první prvek z listu. „RPOP" maže a vrací poslední prvek listu. [28]

Set

Set je podobný listu, avšak nemá pořadí, každý prvek může být v setu právě jednou. Příkaz „SADD" přidá prvek do setu. Příkaz „SREM" prvek smaže. Příkazem „SISMEMBER" lze zkontrolovat, zdali se prvek v setu nachází. Příkaz vrátí 0 nebo 1. Pro získání všech prvků setu slouží příkaz „SMEMBERS". Příkazem „SUNION" lze sloučit 2 sety dohromady. Příkaz přebírá sety jako parametry.

Ve verzi Redis 1.2 byly přidány seřazené sety, kde pro přidání se používá příkaz „ZADD", který přebírá jako první parametr tzv. skóre, podle něhož řadí hodnoty. Pomocí „ZRANGE" pak získáme prvky obdobně jako v listu. [28]

Hash

Posledním datovým typem, který lze do databáze Redis uložit, jsou hashe sloužící jako mapy, které mohou reprezentovat objekty. Příkaz „HSET" slouží pro uložení mapy, kde první parametr značí název mapy, druhý parametr klíč a třetí samotnou hodnotu. Toto lze perfektně použít k uložení objektu, např. v případě objektu typu „User" lze použít „HSET user:1000 name 'Jan'" pro uložení jména a obdobně přidat příjmení a další informace o uživateli. V tomto případě číslo 1000 v podstatě značí index uživatele v mapě. [28]

4.2 Sloupcové databáze

Sloupcové databáze nebo taky z anglického názvu tzv. „columnar stores" jsou databáze, jež mají oproti tradičním SQL databázím volnější datový model, jenž pochází z návrhu nazvaného „Bigtable", představeného vývojáři z Googlu v roce

2008. Nejznámějším a nejpopulárnějším představitelem vycházejícím z tohoto modelu je jednoznačně databáze Apache Cassandra.

Je potřeba zmínit, že kromě sloupcových databází spadajících do kategorie NoSQL existují i tzv. sloupcově orientované SQL databáze nebo také z angličtiny column oriented RDBMS. Mezi jejich představitele patří na příklad MonetDB nebo Vertica, avšak tato práce se tímto typem databází nezabývá. [12]

Základním stavebním prvkem sloupcových databází je řádek (row), kde každý řádek má svůj unikátní klíč (row key), může mít rozdílný počet sloupců, přičemž každý sloupec je definován jménem (column name), obsahuje svou hodnotu (column value) a časové razítko (timestamp). Sloupce se slučují do skupin sloupců (column families), které jsou tvořeny při návrhu schématu databáze. Jednotlivý řádek pak může v rámci skupiny sloupců vytvořit nový sloupec, který ostatní řádky nemají. Dalším stavebním prvkem v tomto typu databází je supersloupec (super column), jehož hodnota vzniká sloučením hodnot více sloupců. Typickým příkladem může být supersloupec „Adresa", který se bude skládat z podsloupců, jako jsou „Město", „Adresa", „PSČ" atd. V podstatě by se dalo říct, že nám supersloupec určitým způsobem nahrazuje třetí normální formu v tradičních SQL databázích. Počet supersloupců v jedné rodině není omezený.

Jeden klíč řádku může být použit napříč různými rodinami sloupců, čímž vzniká jeden celistvý logický datový celek. Koncept těchto databází předpokládá, že rodina sloupců obsahuje sloupce stejného typu a je nanejvýš očekáváno , že budou dotazovány společně, což je také doporučováno. Sloupcové databáze při práci s jednou rodinou sloupců pak dosahují největší efektivity. [12]

Obdobné struktury by se samozřejmě dalo dosáhnout i za pomocí tradičních relačních databází typu SQL. V podstatě každá rodina sloupců by byla představena jako jedna relační tabulka. Avšak tady už narážíme na první problém s tím spojený, a tím je nutnost existence všech sloupců pro všechny záznamy v dané rodině. Takže tam, kde sloupcová databáze dovoluje mít pro jeden záznam sloupce A, B a C a pro druhý záznam pouze sloupce A a D, v SQL databázi by oba sloupce musely mít všechny sloupce, a to A, B, C i D s tím rozdílem, že nevyužité sloupce by obsahovaly hodnotu null. [12]

Ačkoliv je tedy možné dosáhnout stejné struktury za pomoci tradiční relační databáze, vyplatí se pro určité případy zvolit tento přesně navržený model databáze ze dvou hlavních důvodů. Prvním z nich je výše zmíněná možnost přítomnosti pouze vybraných sloupců pro každý řádek jedné rodiny sloupců a schopnost databáze se s tímto vypořádat. Druhým důvodem je pak samotná existence rodin sloupců, u které je opravdu důležitý návrh a zařazení jednotlivých sloupců do jednotlivých rodin a tím dosažení distribuovanosti dat za pomoci efektivního dotazování se právě pouze jedné rodiny sloupců. [12]

Apache Cassandra

Databáze Apache Cassandra, jak společnost Apache uvádí, je sloupcová databáze vhodná pro systémy, jež potřebují škálovatelnost a vysokou dostupnost bez ztráty výkonu. Cassandra dále podporuje replikaci mezi různými datovými centry, což má za následek nízkou latenci a možnost zotavení se z výpadku jednoho uzlu. [33]

Mezi zákazníky tohoto druhu sloupcové databáze se řadí společnosti jako Facebook, Reddit, Instagram, GitHub, eBay, Weatcher channel a mnoho dalších celosvětově využívaných informačních systémů a aplikací. Z názvů společností jasně vyplývá, že databáze Apache Cassandra je navržena na obří projekty využívané napříč celou zeměkoulí. Ideálním příkladem demonstrace použití tohoto typu sloupcové databáze je společnost Netflix, která zpracuje 10 miliónů transakcí každou sekundu. Netflix v minulosti využíval databáze Oracle k ukládání dat. Ta se ale ukázala jako nedostačující k nepřetržitému poskytování provozu a výkonu, a tak byl Netflix donucen zvolit jinou platformu, kterou se stala právě databáze Apache Cassandra provozovaná na cloudu Datastax. Od té doby jsou ukládána data všech 48 miliónů uživatelů v této databázi a společnost Netflix nezaznamenala žádné výpadky. [34]

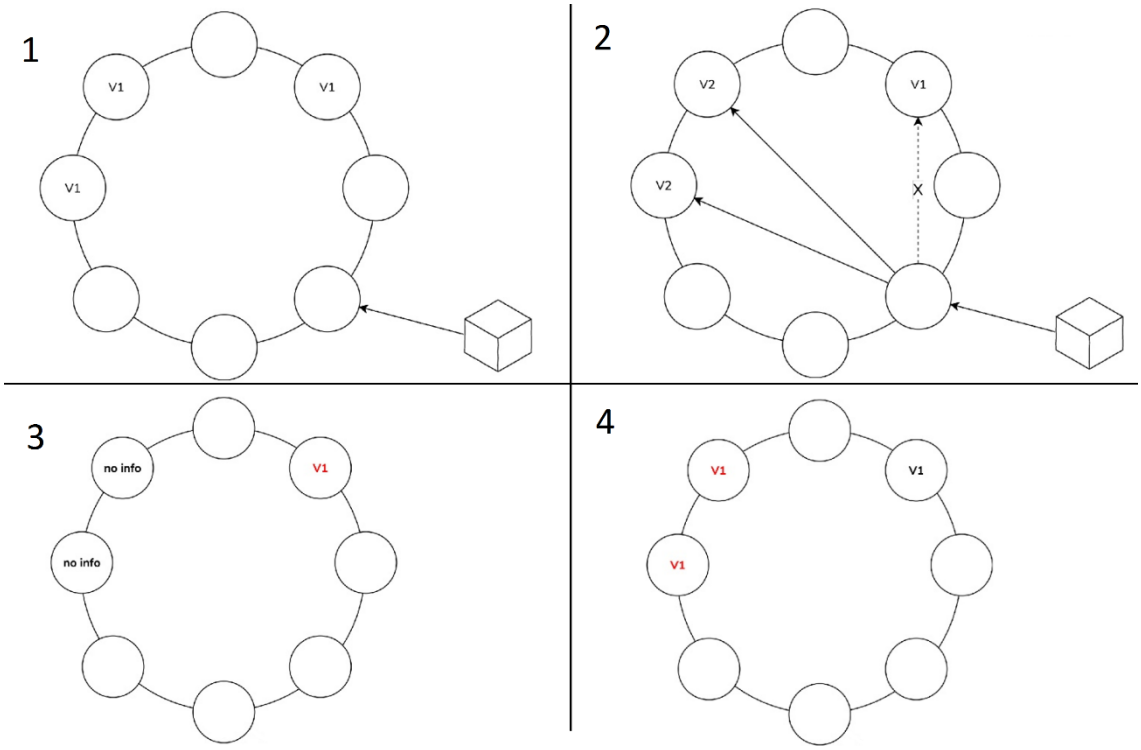
Úskalí návrhu

Na konferenci Geecon 2017 v polském Krakowě měl Michał Matłoka, který působí jako CEO ve společnosti SoftwareMill, přednášku pojmenovanou „Cassandra - how to fail?", kterou začal citátem, jenž si zde dovolím použít: „It's a great tool and we like it, but too often we see teams run into trouble using it. We recommend using Cassandra carefully. Teams often misunderstand the use case for Cassandra,

attempting to use it as a general-purpose data store when in fact it is optimized for fast reads on large data sets based on predefined keys or indexes." [35]

Přednáška byla vedena v duchu nejčastějších chyb při návrhu a používání této databáze z důvodu častých chyb při návrhu nebo zvolení této databáze na špatných místech. Pro demonstraci důležitosti si zde uvedeme jeden příklad.

Úkolem byla snaha mazat záznamy. Zní to sice možná jako primitivní operace, ale nese to s sebou jistá úskalí, která mohou dostat data do nekonzistentního stavu. Situace si lze dobře popsat na následujícím obrázku.



Obrázek 14 Čtyři stavy databáze
Zdroj: Zpracováno dle [35]

Obrázek je rozdělen do čtyř stavů databáze, kde kolečka představují jednotlivé repliky databáze, krychle, potom odkud jsou vysílány požadavky. Popsaná kolečka jsou repliky, v nichž se nachází data, která chceme smazat, aktuálně ve verzi 1. Vyšleme tedy požadavek na smazání, který je automaticky odeslán do všech replik obsahujících tato konkrétní data, jenže v tomto okamžiku dojde k výpadku jedné z nich, způsobenému například výpadkem sítě. Data v bezproblémových replikách se aktualizují do verze 2, tedy se smažou, a v odpojené replice zůstanou tato data stále ve verzi 1. Po opětovném připojení se spustí obnovovací mechanismus, který

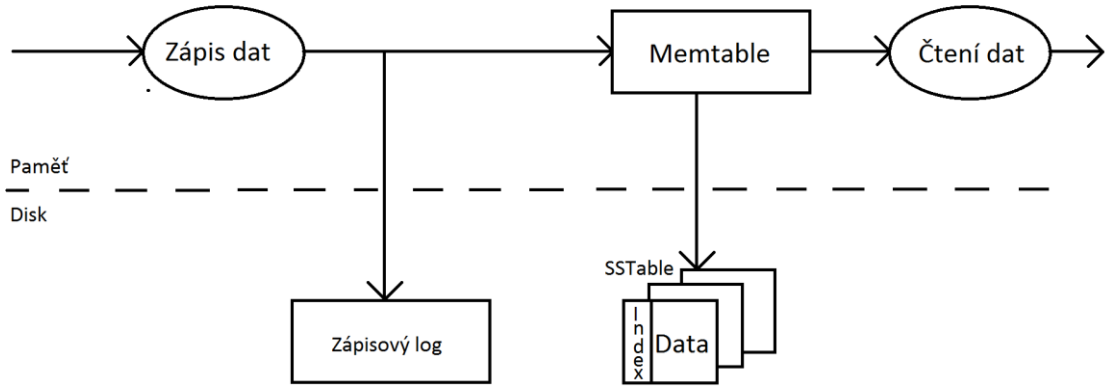
špatně obnoví smazaná data, čímž dojde k jejich znehodnocení. Michał to trefně nazval povstáním zombie.

Motivací tohoto příkladu byla snaha demonstrovat potřebu použití Cassandry tam, kde není potřeba data mazat, nebo dokonce aktualizovat. Vždy je lepší vložit nový záznam a pomocí časové stopy získat ten nejnovější. Mezi dalšími příklady uváděl zvolení Cassandry na malé projekty, kde říkal, že se opravdu hodí, až když v databázi máme záznamy v řádech milionů, ne-li více.

Princip uložení dat

V systému Cassandra jsou data ukládána do tzv. SSTable, což je zkratka od Sorted String Table, kterou původně navrhla společnost Google.

Každá SSTable je časově neměnná, takže do ní nelze zapisovat napřímo. Při zapisování putují data nejdříve do zápisového logu, z něhož pak jdou do paměti, do tzv. memtable, a když je plná, propíše se do SSTable. Čtení probíhá z SSTable i z Memtable. Princip čtení a zápisu je znázorněn na následujícím obrázku: [12]



Obrázek 15 Síťový model
Zdroj: Zpracováno dle [12]

4.2.1 Použití v praxi

V následující podkapitole bude uvedeno několik společností, které databázi Apache Cassandra používají. Dále bude zmíněno, proč si ji zvolily a jaké výsledky jim to přineslo.

Netflix

Společnost Netflix má 48 miliónů uživatelů a zpracuje 10 miliónů transakcí za vteřinu. Z důvodu velké zátěže byla nucena přejít do cloudového řešení, konkrétně

Amazon web services spolu s databází Apache Cassandra místo předchozího řešení od společnosti Oracle. Od doby, kdy společnost přešla na databázi Cassandru, nezaznamenala žádný výpadek. [51]

Facebook

Společnost Facebook využívala Cassandru jako vyhledávač mezi zprávami mezi jednotlivými uživateli. Od toho společnost upustila, když migrovala systém zpráv na databázi HBase z důvodu složitého udržování konzistence s databází Apache Cassandra. Stále je však využívána Apache Cassandra na ukládání „to se mi líbí" u jednotlivých příspěvků / komentářů. Toto se jeví jako ideální případ použití Cassandry z toho důvodu, že není nutná konzistence v reálném čase na všech uzlech. V praxi to pak může vypadat tak, že člověk z Evropy uvidí na stejném příspěvku jako člověk z Ameriky méně, resp. více „to se mi líbí" a k synchronizaci může dojít až po nějakém čase. [36]

eBay

Společnost eBay čítala k roku 2012 celkem 97 miliónů aktivních uživatelů, více jak 200 miliónů prodejních předmětů, 2 miliardy zhlédnutých stránek denně, 80 miliard databázových volání denně a více jak 5 petabytů úložného prostoru, ke kterým je nutno připočítat více jak 80 petabytů prostoru pro analytické operace. Cassandru si zvolila z důvodu škálovatelnosti, replikace, výkonu při zápisu a podpory frameworku Hadoop. Avšak společnost eBay zdůrazňuje, že využila NoSQL, avšak nenahradila tím tradiční relační databáze. Mezi případy použití se řadí sledování detekce podvodného jednání, doporučené předměty nebo předměty, které jsou podobné jako prohlížený předmět atd. [34]

4.2.2 Instalace databáze

Oficiální distribuci lze stáhnout ze stránek cassandra.apache.org, kde jsou dostupné Debian balíky a RPM balíky, tzn. že tedy neexistuje oficiální distribuce pro operační systém Windows. Nicméně existuje MSI instalační balíček od společnosti Datastax, který obsahuje nejnovější verzi databáze Apache Cassandra, CQL shell a aplikaci Datastax DevCenter, která obsahuje grafický nástroj pro práci s databází. Aplikace je založená na platformě Eclipse RCP. [33]

4.2.3 Ukázka práce s databází

Tato podkapitola rozebírá různé druhy možností práce s databází, jako jsou standardní jazyk CQL nebo framework Apache Hadoop.

CQL

CQL nebo také „The Cassandra Query Language“ je hlavní jazyk určený pro komunikaci s databází Apache Cassandra. Základním způsobem pro možnost práce s tímto dotazovacím jazykem je CQL shell, který nalezneme spíše pod označením „cqlsh“, nebo lze využít grafický nástroj Datastax DevCenter. [33]

Struktura modelu databáze Cassandra a jazyka SQL se v průběhu vývoje přetransformovala do kompletně jiné podoby, než ve které byla ve verzi 1.0, a dnes již odpovídá modelu, v němž jsou jednotlivé rodiny sloupců vnímány jako tabulky, a stejně tak se i v jazyku CQL termín „table“ používá místo termínu „column family“. [12]

Ukázky práce s jazykem SQL jsou prováděny na operačním systému Windows 10. Je tedy potřeba mít nainstalovaný výše zmíněný MSI balíček. Po nastartování databáze pomocí skriptu cassandra.bat, umístěném ve složce bin domovského adresáře MSI balíčku, se nám otevře příkazový řádek, který slouží jako log databáze. Pak už jen stačí otevřít CQL shell nebo DevCenter a databáze je připravená k práci. Pro potřeby této práce postačí CQL shell.

Příklady CQL

Jako první je potřeba vytvořit keyspace, například Store, který bude představovat sklad. Keyspace vytvoříme pomocí následujícího příkazu:

```
CREATE KEYSPACE store
WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };
```

Tímto příkazem vytvoříme keyspace pojmenovaný store s nastavenými vlastnostmi pro replikaci. Jako první parametr nastavujeme parametr „class“, který představuje strategii replikace, a jako druhý parametr „replication_factor“, který představuje nastavení míry replikačního faktoru. „SimpleStrategy“ je určená pro testování a vývoj, protože přiřazuje stejný replikační faktor celému clusteru. Pro testování na úrovni FAT, UAT nebo pro samotné nasazení do produkce slouží

strategie „NetworkTopologyStrategy“. U replikačního faktoru klíč představuje název datacentra a číslo počet kopií. Jednotlivá datacentra s uvedeným počtem kopií jsou oddělena čárkou. [36]

Jako další příkaz přichází samotné vytvoření tabulky, nejdříve je ale potřeba dostat se v CQL shell do kontextu daného keyspace za pomoci následujícího příkazu:

```
USE store;
```

Poté už můžeme vytvářet tabulky pomocí následujícího příkazu, který je velice podobný jazyku SQL:

```
CREATE TABLE item (pk text PRIMARY KEY, weight float, description text);
```

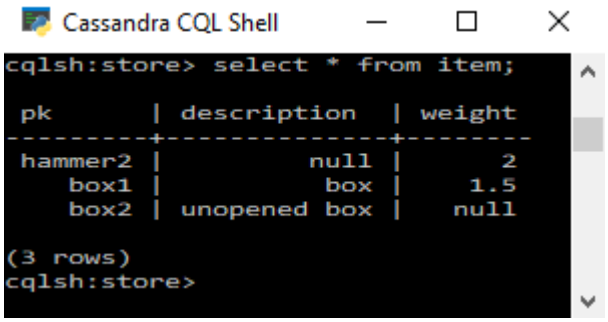
Následující příkaz vytvoří tabulku představující předmět ve skladu popsaný třemi sloupci, představujícími primární klíč, váhu a popis. Vložit do této tabulky záznamy můžeme pomocí následujících příkazů:

```
INSERT INTO item (pk, weight, description) VALUES ('box1', 1.5, 'box');
INSERT INTO item (pk, weight) VALUES ('hammer2', 2);
INSERT INTO item (pk, description) VALUES ('box2', 'unopened box');
```

Tyto příkazy vloží 3 záznamy do tabulky item umístěné v keyspace store. Pro zobrazení všech záznamů v tabulce můžeme využít příkaz:

```
SELECT * FROM item;
```

Tím dostaneme následující výsledek:



Obrázek 16 Výsledek dotazu databáze Cassandra
Zdroj: Vlastní zpracování

Výsledek na první pohled vypadá totožně jako v relační databázi, ale rozdíl spočívá v tom, že v tradiční relační databázi jsou hodnoty null fyzicky uloženy na disk z důvodu pevné struktury dat (každý řádek musí mít všechny sloupce), zatímco zde jsou sice reprezentovány jako null, protože jsou uvedené v definici tabulky, ale nejsou fyzicky uloženy na disku. [12]

Výše uvedená syntaxe vytvoření tabulky neumožňuje dynamické přidávání sloupců, na to je potřeba využít některou ze tří definovaných kolekcí, a těmi jsou: set, list nebo mapa. Tabulku využívající mapu lze tedy využít pomocí následujícího příkazu:

```
CREATE TABLE item (
pk text PRIMARY KEY, weight float, description map<text, text>);
```

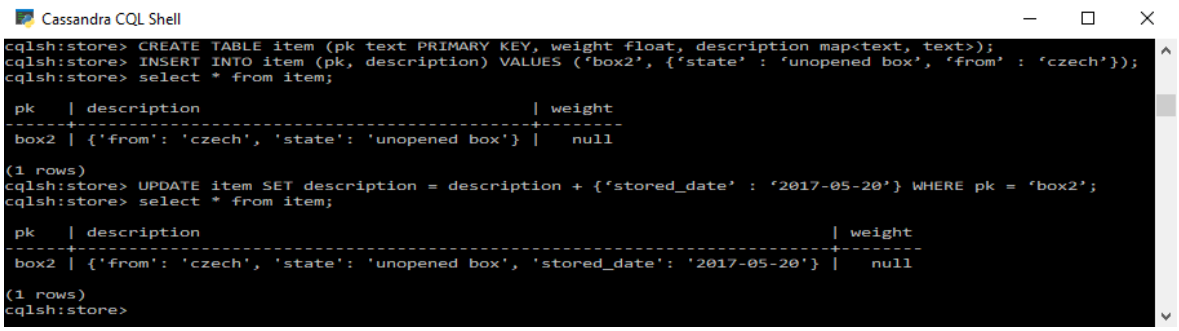
A vložit záznam pomocí:

```
INSERT INTO item (pk, description)
VALUES ('box2', {'state' : 'unopened box', 'from' : 'czech'});
```

Vložit další sloupec se pak do dané mapy, v našem případě tedy přidat další parametr popisu předmětu na skladě, dá pomocí následujícího příkazu:

```
UPDATE item SET description = description + {'stored_date' : '2017-05-20'}
WHERE pk = 'box2';
```

Zde je nutné si dát si pozor na potřebu přičtení nového sloupce ke stávajícím, jinak by mohlo dojít k přepsání ostatních. Avšak to vychází z jazyka Java, ve kterém je Cassandra napsaná. Po vypsání výsledků vidíme následující výsledek:



Obrázek 17 Výsledek dotazu 2 databáze Cassandra
Zdroj: Vlastní zpracování

Na vypsaném výsledku lze vidět, že kolekce představují rodiny sloupců. [37] Stejně jako v SQL můžeme sloupcům přiřazovat kromě standardních implementací indexů, jako je třeba B-strom, také vlastní implementace indexů. Index přiřazujeme následujícím příkazem: [12]

```
CREATE INDEX IN item (description);
```

Apache Hadoop

Apache Hadoop je open-source framework od společnosti Apache, určený pro spolehlivé, distribuované, škálovatelné výpočty.

Tento framework umožňuje distribuované zpracování velkého množství datových setů napříč jednotlivými clustery počítačů za použití jednoduchých programovacích modelů. Je navržen tak, aby škáloval od jednoho stroje až po tisíce strojů, kde každý stroj umožňuje výpočet a uložení dat.

Projekt zahrnuje čtyři základní moduly, a to:

- Hadoop Common – Společná knihovna obsahující soubory potřebné napříč moduly;
- Hadoop Distributed File System (HDFS) – Distribuovaný souborový systém poskytující vysoce prostupný přístup k datům;
- Hadoop YARN – Modul určený pro správu zdrojů a plánování;
- Hadoop MapReduce – Modul založený na YARNu určený pro paralelní zpracování velkých datových setů.

Mimo tyto čtyři základní moduly najdeme další, jako jsou Ambari, Avro, HBase, Spark, Cassandra a další. [38]

Apache Hadoop doplňuje databázi Cassandra v možnosti zpracování starších dat uložených v datových skladech a tím efektivně doplňuje Cassandru v možnosti uskutečnění efektivní podpory různých analytických požadavků jak na historická data, tak na data v aktuálním čase. [39]

Dokumentové databáze

Jak již název napovídá, jedná se o úložiště dat, které umožňuje ukládat dokumenty nejčastěji v podobě XML nebo JSON, resp. jeho binární reprezentace BSON. Hlavním důvodem vzniku byla odpověď na rychle se rozrůstající webové aplikace

a na velké množství dat, která generují a posílají si mezi sebou právě v těchto zmíněných formátech. V aplikacích bylo tedy nutno jednotlivá data pro další práci s nimi konvertovat na objektové entity, i.

Mezi nesporné výhody dokumentových databází patří možnost volného přidávání „sloupců". Z hlediska struktury databáze nelze mluvit o sloupcích, ale pro uvědomění si příbuznosti s ostatními typy databází slouží název „sloupec" jako dobrá analogie k sloupcovým databázím nebo SQL databázím. Přidání takového sloupce se provede jednoduše pomocí rozšíření dokumentu na dané úrovni. Stejně jednoduše lze data zanořovat do sebe. Datový model databáze reprezentuje dokument sám, čímž je zajištěna absolutní svoboda návrhu. Na následujícím obrázku lze vidět ukázku dokumentů typu JSON a XML.

Úskalí návrhu

Při zanořování jednotlivých úrovní dokumentů je potřeba myslet na všechny operace, které se s daným dokumentem budou provádět. Řeč je o operacích typu vložit nový dokument, upravit nebo smazat. Pro názornou ukázku si představme entitu Kategorie, která má své popisné atributy, jako jsou popis, datum vytvoření, autor nebo id, jež jsou stejně jako v ostatních databázích povinné. Kategorie bude obsahovat produkty, které postupem času budou nabývat na počtu. Situace by tedy šla vyřešit prostým přidáváním produktů do dokumentu představujícího kategorii do zanořené úrovně představující produkty. Jenže kdyby pak pro jednu takovou kategorii existovalo sto produktů nebo třeba tisíc produktů, pak by dokument nabyl obřích rozměrů a podstatně by se zpomalila rychlost čtení, zápisu i úpravy dat. V takovém případě hovoříme o denormalizovaném návrhu. Z tohoto příkladu vyplývá, že zanořené entity se vyplatí dávat do dokumentu pouze v případě vazby 1:1 nebo v případě, kdy víme, že konečný výčet poddružených entit nepřesáhne počet do pěti výskytů. Příkladem, kde je to únosné, může být domácí adresa a fakturační adresa. U tohoto typu víme, že další adresa nepřibude, maximálně třeba přechodné bydliště. Vazba 1:N se pak řeší obdobně jako v relačních databázích přidáním „cizího klíče" v podobě primárního klíče rodičovské tabulky do poddružené tabulky. Tím se zároveň zamezí duplicit.

Je tedy na zvážení analytika daného systému, aby v případě, že se rozhodne jít cestou dokumentových databází, rozhodl, zda se budou poddružené entity zanořovat, nebo se vytvoří nový dokument s referencí na rodičovskou entitu. Vše vychází z povahy dat a jejich následného dotazování. Nejkritičtější operací je samozřejmě úprava dat. Vždy při navrhování struktury takové databáze je třeba brát ohled na to, jestli bude existovat potřeba úpravy takových dat a jak často k takovým úpravám bude docházet.
[12], [40], [41]

Alternativní řešení – hybridní databáze

V případě, že daný systém potřebuje relační databázi a zároveň by se hodila dokumentová databáze, nabízí se jako možná alternativa databáze PostgreSQL, která poskytuje datový typ JSONB umožňující ukládat dokumenty typu JSON.
Hlavní výhoda tohoto přístupu spočívá v tom, že se jedná o standardní relační databázi, takže s kompletní tabulkou pracujeme pomocí jazyka SQL rozšířeného o podporu práce s daty uloženými v atributu datového typu JSONB. Takže se v podstatě jedná o klasickou relační databázi, která poskytuje možnost uložit v jednom sloupečku dokument a s ním pracovat jako s dokumentovou databází. Můžeme provádět příkazy, které jsou omezeny konkrétním atributem dokumentu, nebo naopak získávat pouze některé atributy dokumenty. Databáze navíc poskytuje možnost kombinovat standardní sloupečky spolu s atributy dokumentu např. v klauzuli „WHERE". Pro názornost budou uvedeny jednoduché příklady. Definice tabulky může vypadat takto:

```
CREATE TABLE product(
        ID SERIAL PRIMARY KEY,
        DESCRIPTION JSONB
);
```

Jedná se tedy o klasickou relační tabulku, která má 2 sloupečky, kde jedním je ID, které slouží jako primární klíč, a druhý sloupec je description, který umožňuje ukládat dokumenty typu JSON. Pro vložení záznamu postačí tradiční insert tak, jak ho známe z relačních databází:

```
INSERT INTO product (ID, DESCRIPTION) VALUES (
1, '{
        "name" : "hammer",
        "weight" : "3.5",
        "material" : "iron"}');
```

Takový záznam má z pohledu relačních databází pouze 2 sloupečky, ale z pohledu NoSQL databází má sloupečky čtyři. Navíc další vložený záznam může obsahovat další atributy dokumentu, popřípadě zcela odlišné atributy. Pokud bychom chtěli získat třeba váhy všech kladiv, která jsou ze železa, mohl by dotaz vypadat například takto:

```
SELECT DESCRIPTION->>'weight' AS WEIGHT
FROM product
WHERE DESCRIPTION->>'name' = 'hammer' AND DESCRIPTION->>'material' = 'iron';
```

Jak v klauzuli SELECT, tak v klauzuli WHERE můžeme samozřejmě kombinovat tradiční sloupečky spolu s atributy dokumentu.

Mongo DB

Jedná se o jednu z nejznámějších a zároveň nejpoužívanějších NoSQL databází vůbec, která byla založena v roce 2007 týmem ve složení Dwight Merriman, Eliot Horowitz a Kevin Ryan,v té době tzv. DoubleClick team, jehož jméno koresponduje i s tehdejší stejnojmennou firmou, jejímž současným vlastníkem je společnost Google. Tento tým se pro svou firmu snažil vytvořit úložiště dat, které by odpovídalo jejím potřebám, což bylo zvládnutí zpracování 400 000 reklam za sekundu.

Z tohoto důvodu vzniklo dnes již dobře známé MongoDB, které slouží pro velké projekty potřebující to nejlepší z tradičních databází a k tomu možnost škálování, flexibility a výkonu.

Mimo jiné vznikla i MongoDB univerzita, která vyškolila již více jak 350 000 expertů na tuto sloupcovou databázi. Dnes MongoDB funguje již v 18 státech světa

napříč Amerikou, Evropou nebo Asií, čítá dvacet miliónů stáhnutí a zaměstnává více jak 800 lidí po celém světě. [42]

MongoDB se řadí mezi přední představitele multiplatformních dokumentových databází, které nemají pevné schéma. To znamená, že podobně jako u Cassandry lze dynamicky přidávat „sloupce". Princip uložení spočívá v ukládání do dokumentů typu JSON, na disku pak BSON, což je JSON v binární podobě. Mezi další výhody patří široká komunita vývojářů, podpora velkého množství programovacích jazyků, mezi něž se řadí jazyky jako Java, Python, C++, Ruby, PHP a další. [40]

4.2.4 Použití v praxi

Databázi MongoDB využívají světově známé společnosti jako Cisco, Facebook, Google, UPS, EA, eBay, Bosch, Metlife a další. Následující kapitola uvádí výběr několika z nich a vysvětluje, jak tuto databázi využívají a co byl impuls pro zvolení právě této dokumentové databáze.

Google

Společnost Google, která vlastní nejpoužívanější internetový vyhledávač, používá databázi MongoDB ve svém Google Compute Engine, čímž je dosaženo možnosti nasazení MongoDB takřka v jakémkoliv cloudu. Vzhledem k absenci nutnosti placení poplatků lze pomocí Google Compute Engine dosáhnout efektivního, cenově dostupného řešení oproti tradičním databázovým systémům. Díky MongoDB klesla celková provozní režie a zvýšil se celkový výkon systému. Jednoduše a rychle lze přidat další MongoDB shard a pomocí něj aplikaci škálovat a v podstatě neomezeně zvyšovat výkon. [42]

Expedia

Jedná se o americkou dopravní společnost, která vlastní a spravuje mezinárodní online cestovní značky, jako jsou Trivago, hotels.com, expedia.com a další. Aplikace zvaná „Scratchpad" si pamatuje vyhledávání, umožňuje ho z jakéhokoliv zařízení, umí najít nejnižší ceny. Díky využití MongoDB a jeho flexibilnímu úložišti dokumentů a horizontálnímu škálování lze sbírat data v reálném čase. Flexibilní

datový model pak umožňuje jednoduše ukládat kombinace párů měst, termínů a destinací. [42]

DHL

Logistická společnost DHL za pomoci MongoDB rapidně zjednodušila celkovou architekturu ve srovnání s tradičními databázemi a za pomoci MongoDB OPS snížilo o 30 % celkové úsilí operátorského týmu DHL. Implementace MongoDB mohla jít z prototypu do produkčního prostředí během několika týdnů a ihned přinesla zlepšení. Celková úspora se čítá na 66% snížení nákladů díky menší zátěži hardwaru a elastickému škálování. [42]

4.2.5 Instalace databáze

MongoDB poskytuje hned několik licencí, a to od komunitních, které jsou zadarmo, až po komerční licence za peníze, které navíc obsahuje úložiště dat v paměti, možnost šifrování dat na úrovni Rest Api nebo pokročilou správu zabezpečení za pomoci autentizačních serverů LDAP nebo Kerberos. Oba typy licencí jsou oficiálně dostupné jak pro platformu Windows, tak pro platformy OS X nebo GNU/Linux a lze je stáhnout z oficiálních stránek www.mongodb.com/download-center.

Pro úspěšné nainstalování databáze je potřeba rozbalit archiv obsahující databázi, nastavit si správnou cestu ke složce „bin" v domovském adresáři databáze a poté jednoduše spustit pomocí příkazu „mongo", čímž dojde k nastartování databáze. [42], [43]

4.2.6 Ukázka práce s databází

Po spuštění databáze příkazem „mongo“, popsaném v kapitole Instalace databáze, se dostaneme do prostředí MongoDB shell, které nám pro naše účely testování databáze bohatě postačí.

Není nutné vytvářet nové instance databáze. Stačí se přepnout do námi požadované databáze a mongoDB se o vše postará. Přepnutí provedeme příkazem:

```
use myOwnDatabase
```

Po provedení tohoto příkazu se databáze přepne do daného kontextu a poté už lze provádět standardní operace, jako je vkládání, aktualizování nebo vyhledávání záznamů. Pro vložení dokumentu je potřeba zadat příkaz:

```
db.category.insert(
  {
    "title" : "Tools",
    "description" : "Classic tools",
    "products" : [
      {
        "title" : "Steel hammer",
        "weight" : 3.5,
        "description" : "Classic steel hammer"
      },
      {
        "title" : "Screwdriver",
        "weight" : 1,
        "description" : "Cross-point screwdriver"
      }
    ]
  }
)
```

Po provedení následujícího příkazu je vložen dokument do kolekce představující kategorii „klasické nářadí“, která má atributy název a popis a k tomu produkty spadající do této kategorie, jako jsou kladivo a šroubovák, které mají atributy název, popis a váha. Samozřejmě lze atributy dokumentu volně přidávat i odebírat, a to bez nutnosti držet pevně dané schéma.
Pro získání všech záznamů nám postačí příkaz:

```
db.category.find()
```

Pro získání konkrétní kategorie pak příkaz:

```
db.category.find( { "title": "Tools" } )
```

Stejně tak není problém vyhledávat podle atributů zanořeného dokumentu, takže vyhledávání podle produktů vypadá takřka totožně až na definici názvu zanořeného dokumentu:

```
db.category.find( { "products.title" : "Steel hammer" } )
```

Obdoba „WHERE“ klauzule neboli specifikování podmínek dotazu lze provádět pomocí následujícího příkazu v případě, že chceme použít větší než, resp. menší než:

```
db.category.find( { "products.weight": { $gt: 2 } } )
```

Tento příkaz vyhledá všechny kategorie obsahující produkty, které mají váhu větší než dva kilogramy. Pro použití menší než stačí použít „$lt“, vyjadřující less than, resp. greater than z anglického jazyka.
Pro specifikaci logického AND stačí přidávat specifikace za čárku následovně:

```
db.category.find( { "products.title: „Steel hammer“, “products.weight": { $gt: 2 } } )
```

Maličko složitější je pak logický OR, který potřebuje na začátku specifikovat, že se jedná o logický OR:

```
db.category.find( { "products.title: „Steel hammer“, “products.weight": { $gt: 2 } } )
```

Pro updatování dokumentu lze použít následující syntaxi:

```
db.category.update(
  { "title" : "Tools" },
  {
    $set: { "description": "new description" }
  }
```

```
)
```

Mazání záznamů se provádí pomocí následujícího příkazu:

```
db.category.remove( { "title": "Tools" } )
```

V případě nutnosti limitace počtu smazaných záznamů lze použít syntaxi:

```
db.category.remove( { "title": "Tools" }, { justOne: true } )
```

Smazání všeho se pak provede pomocí příkazu:

```
db.restaurants.remove( { } )
```

4.3 Grafové databáze

Jak již název napovídá, jedná se o databáze, jejichž strukturu tvoří graf, který jako hlavní stavební prvky má uzly a hrany mezi nimi. Uzel představuje jakoukoliv entitu, která může mít libovolný počet jakýchkoliv atributů. Hrany pak představují vztahy mezi jednotlivými entitami, které také mohou mít definované atributy. U hran lze navíc definovat orientaci, čímž je definován směr vztahu. Pro názornou ukázku si představme entitu vynálezce, kde jedním záznamem by byl vynálezce Prokop Diviš, jenž by měl atributy jako právě jméno, datum narození, obor působení atd., na druhou stranu by existovala entita vynález, kde konkrétním záznamem by byl v tomto případě „hromosvod“, který by měl také svoje parametry jako datum objevení, popis, název atd. Mezi těmito záznamy by existovala orientovaná hrana směrem od vynálezce k vynálezu pojmenovaná jako „vynalezl". Na toto pojmenovávání je mnohem praktičtější angličtina, kde se používají spojení jako: WORKS_AT, PRODUCES, PARTNER_OF atd. Může existovat neomezený počet jak vazeb, tak také entit. U vazeb lze navíc i definovat parametry, jako jsou doba platnosti vztahu nebo podmínky existence vazby.
Grafy jako takové obsahují spoustu různých možností a variant, podle nichž se dělí na určité typy. Jednotlivé grafické databáze tyto typy odráží a každá jedna grafová databáze pracuje ve většině případů pouze s jedním typem grafu.

V základu dělíme grafy na orientované a neorientované, přičemž přítomnost orientace je dána tím, jestli má vazba mezi dvěma hranami orientovanou, nebo neorientovanou vazbu. Neorientovaná hrana by se dala nazvat jako oboustranná. Dalším atributem, který nám grafy dělí na jednotlivé typy, je forma vyjádření vztahu, kde můžeme mít ohodnocené, nebo neohodnocené grafy. Hodnota zde vyjadřuje určité ohodnocení dané vazby, např. vazba může být ohodnocena číslem vyjadřujícím počet kilometrů v případě, že představuje cestu mezi dvěma body. Na takto ohodnocených grafech lze provádět algoritmy, jako je výpočet nejkratší cesty, princip čínského pošťáka atd. Mezi další dělení patří tzv. multigrafovost, která určuje, jestli mezi dvěma vrcholy může existovat více jak jedna vazba různého druhu.

Grafového rozpoložení lze dosáhnout i za pomoci tradičních relačních nebo sloupcových databází, avšak naráží se tam na problém, že tyto databáze nejsou koncipované k rychlému získávání sousedů dané entity, což je v podstatě princip grafových databází. Hlavním důvodem, proč tedy nelze použít ostatní databáze k tomuto účelu, je výkonnostní optimalizace pro daný typ problému. Takové databáze z tohoto pohledu nazýváme „non-native". Sice se nasimuluje uložení do grafu, ale způsob uložení a dotazování bude pořád odpovídat danému typu databáze. Naopak grafové databáze jsou tzv. „native", čemuž odpovídá opět princip uložení a dotazování, a především 2 vlastnosti, které dělají grafové databáze tím, čím jsou. Jedná se o tzv. „adjacency property" a „index-free adjacency", což znamená, že při zápisu dat je zajištěna vyšší rychlost tím, že každý uzel je uložen přímo do sousedních uzlů a vazeb. Během čtení (dotazování) je zajištěna vysoká rychlost bez použití indexů právě díky uložení sousedících uzlů. Dalším důležitým faktorem je obdoba ACID u těchto databází. Při vytvoření vazby mezi dvěma entitami nestačí přidat vazbu, ale je zapotřebí spolu s tím i updatovat obě dvě entity přidáním informace o vazbě. V případě, že jedna z těchto tří akcí selže, je graf označen jako poškozený. Nativní grafové databáze mají implicitně implementován mechanismus zajišťující ochranu kvality dat před problémy, jako jsou výpadek sítě, kolaps serveru nebo konkurenční transakce.

Mezi nejznámější grafové databáze patří OrientDB, Sones GraphDB, AllegroGraph, InfiniteGraph a Neo4j. V rámci této práce byla jako zástupce grafových databází vybrána databáze Neo4j.
[12], [50], [51]

Neo4j

Jedná se o jednu z nejpopulárnějších grafových databází z důvodu agilního vývoje, poskytování open-source licence a dalších vlastností, jako je integrace Rest API. Databáze je implementována v jazyku Java, čímž je zajištěna multiplatformnost napříč jednotlivými operačními systémy. Představena byla firmou Neo Technology v roce 2007 a od té doby našla uplatnění v mnoha systémech. Má silnou vývojářskou komunitu, která má aktivní fórum a přispívá i do vývoje samotné databáze. [52]

4.3.1 Použití v praxi

Mezi zákazníky, již využívají databázi Neo4j, se řadí nadnárodní společnosti jako Walmart, Airbnb, NASA, eBay, Microsoft, IBM, Cisco a další. Zde bude představeno několik firem, uvedeno bude, k čemu Neo4j používají a proč se rozhodly jít touto cestou.

eBay

Společnost eBay využívá databázi Neo4j pro směrování své služby „eCommerce Delivery Service". Motivací byla schopnost obsloužit zákazníky a obchodníky v co nejkratším čase tak, aby své zboží dostali nejlépe ještě ten den, kdy si ho objednají. Neo4j byla vybrána pro svou flexibilitu, rychlost a snadné použití. Vzhledem k tomu, že Neo4j podporuje grafy s vlastnostmi, nebylo o čem diskutovat, protože to bylo přesně to, co společnost eBay potřebovala. Flexibilní schéma splňovalo požadavky na snadnou rozšířitelnost a po po krátké době používání předčilo toto řešení to předchozí v rychlosti zpracovávání a škálování systému. [52]

NASA

NASA využívá Neo4j hned na několika projektech, jak uvádí David Meza, který je vedoucím architektem v Johnson space centru v americkém Houstonu společnosti NASA. Jako první použili Neo4j pro zrychlení prohledávání jejich interní znalostní

báze, obsahující veškeré získané poznatky. Dříve používali tradiční vyhledávání podle klíčů, ale bylo zapotřebí prozkoumat datové vztahy mezi jednotlivými poznatky za posledních 50 let. Grafové databáze tento problém snadno vyřešily spojením jednotlivých poznatků, čímž došlo k zabránění budoucím problémům. David uvádí, že díky Neo4j někdo z projektu Orion našel informace z projektu Apollo, které ušetřily dva roky práce a jeden milion dolarů. [52]

LinkedIn

LinkedIn je sociální síť zaměřená na pracovní trh a kariéry uživatelů, kteří tuto síť využívají a mluví nejčastěji anglickým jazykem. Společnost si všimla díry na trhu v podobě mladých profesionálů v Číně, a tak se rozhodla spustit službu Chitu, která se snaží najít práci uchazečům. Chitu nebyla jediná, kdo se o to snažil, a tak bylo zapotřebí spustit ji do produkce co nejdříve. Vzhledem k počtu lidí v Číně musela být aplikace připravena na obří nárůst uživatelů a funkcí. Neo4j zaplnila mezeru ve výsledném řešení Chitu a hned po nasazení se rapidně zlepšil výkon, rychlost zpracování požadavků a bonusem bylo snadné použití. Dotazy jsou nyní prováděné v rekordním čase a aplikace mohla jít do produkce za pouhé čtyři měsíce. [52]

Jízdomat

Jízdomat je aplikace sloužící k možnosti nabídnutí jízdy z bodu A do bodu B, kde se do vozidla může přidat spolujezdec, který si tuto cestu vyhledá. Toto se jeví jako typický příklad pro použití Neo4j, kde klíčová je důvěra mezi řidičem a spolujezdcem, kteří se ve většině případů neznají. Avšak díky databázi Neo4j lze snadno dohledat společné přátele přes X uzlů, kde ve výsledku bylo zjištěno, že maximální důvěryhodná vazba je přes 2 uzly. Vývojáři nejdříve využili Dijkstrův algoritmus napsaný v jazyku PHP, kde sami uvádí, že to bylo schopno výkonnostně pobrat do řádu tisíců uživatelů, poté přešli na databázi MySQL a výsledky skládali pomocí operací „JOIN“, což jim ve výsledku stačilo na řády desetitisíců uživatelů. Poté přešli na databázi Neo4j a dotazovací jazyk Cypher, který byl schopný vracet výsledky v řádu sekund. Po aplikaci pluginu od Michala Bachmana se výsledky vracely v řádu 10-20 ms. Jan Mittner, který tyto informace uváděl na své přednášce, představil zajímavé statistiky z této aplikace. V té době čítala 70 000 uživatelů. Když se pokusil najít, kolik zná uživatelů, pomocí jedné hrany, dostal

výsledek 196. Za použití dvou hran už to bylo 2740 uživatelů a za použití tří hran to bylo 20247 uživatelů. Pro čtyři hrany už se k výsledku nedobral. Každopádně se databáze Neo4j jeví jako ideální možnost pro problémy, jako jsou princip malého světa, resp. šest stupňů odloučení, což je teorie, která předpokládá, že každý člověk je spojen s každým člověkem za pomoci maximálně šesti lidí, kteří se navzájem znají. [51], [53]

4.3.2 Instalace databáze

Oficiální podpora existuje pro všechny běžně dostupné operační systémy, tzn., že je běžně dostupná distribuce jak pro operační systémy typu OS X, GNUd/Linux, tak i pro Windows. Oproti Cassandře a MongoDB je Neo4j distribuována pomocí spustitelného souboru. Pro Windows je tedy instalce provedena pomocí souboru s příponou „exe" a pro OS X soubor s příponou „dmg". Po otevření se objeví formulář pro vybrání umístění databáze, která se poté automaticky připraví. V případě potřeby je možno doladit si konfiguraci v nabídce „options". V základu je databáze Neo4j dostupná na portu číslo 7474. Po připojení pomocí webového prohlížeče se dostaneme do přehledného grafického rozhraní, kde po zadání základního jména a hesla dostaneme pokyn ke změně hesla.

4.3.3 Ukázka práce s databází

Tato podkapitola pojednává o různých možnostech práce s databází Neo4j.

Cypher

Cypher je deklarativní jazyk inspirovaný jazykem SQL, určený pro popisování vzorů v grafu za pomoci ascii-art syntaxe. Umožňuje určit, co chceme selectovat, aktualizovat, vložit nebo smazat bez potřeby přesného popisu toho, jak to udělat, resp. jak projít grafem. Cílem bylo vytvořit jednoduchý intuitivní jazyk, který bude podobný SQL. [52]

Webové rozhraní

Databáze Neo4j nabízí webové rozhraní, které je naprosto dostačující a ideální pro první setkání s databází, popřípadě pro testování možností databáze. Databáze

mimo jiné obsahuje skripty pro její naplnění testovacími daty a následné skripty s dotazy pro demonstraci funkcí databáze Neo4j.
Ve webovém prohlížeči máme několik záložek, z nichž hned první je konzole, v níž lze spouštět jednotlivé příkazy. Logicky je prvním příkazem vytvoření uzlu pomocí příkazu:

```
CREATE (TheHammer:Product {title:'Steel hammer', weight:3.5,
description:'Classic steel hammer'})
CREATE (TheScrewdriver:Product {title:'Cross-point screwdriver',
 weight: 0.5, description:'Classic cross-point screwdriver'})
CREATE (Tools:Category {title:'Tools category', description:'Category where
the tools belongs'})
```

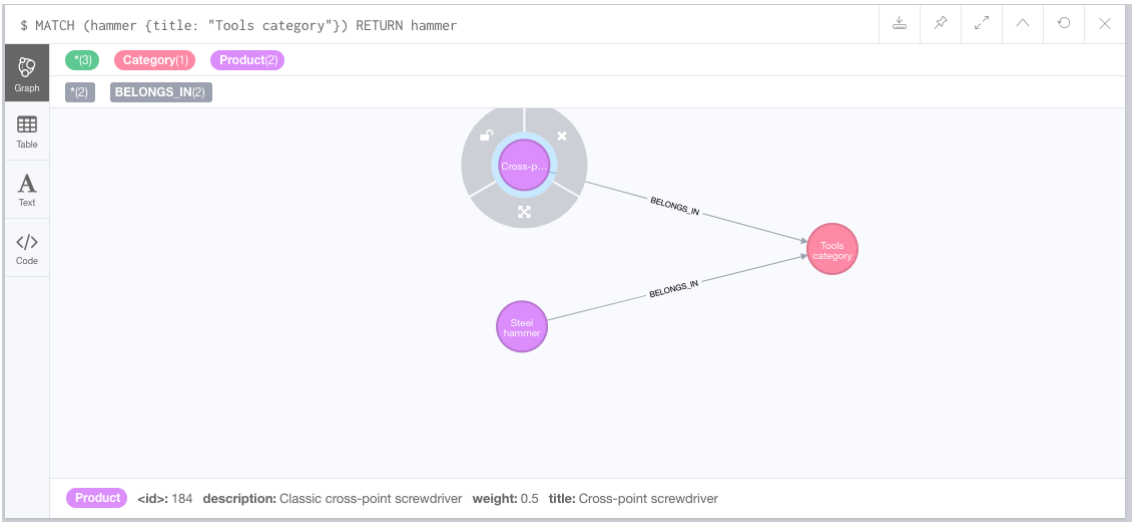
Zde je tedy vytvořen uzel typu produkt, konkrétně kladivo a šroubovák s atributy název, váha a popis. Dále je vytvořen uzel kategorie, a to konkrétně nářadí s atributy název a popis. Pro vytvoření vazby mezi těmito uzly je potřeba provést příkaz:

```
CREATE
 (TheHammer)-[:BELONGS_IN {only:['True']}]->(Tools),
 (TheScrewdriver)-[:BELONGS_IN]->(Tools)
```

Tímto příkazem jsou vytvořeny dvě vazby, kde to logicky odpovídá vazbě kladivo „patří do" kategorie nářadí. U kladiva je specifikovaný ještě atribut dané vazby, který vyjadřuje, že spadá jen a pouze do této kategorie, žádné jiné.
Pro vyhledávání slouží následující příkaz:

```
MATCH (hammer {title: "Tools category"}) RETURN hammer
```

Slovo MATCH je klíčové slovo, které musí být uvedeno, poté následuje název proměnné, do které ukládáme výsledek, a v závorce je obdoba „WHERE" klauzule, kde se specifikuje atribut, který hledáme. Zajímavé je, že není potřeba uvádět název uzlu. V grafickém prostředí Neo4j dostáváme vizuální výsledek uvedený na následujícím obrázku:



Obrázek 18 Webové rozhraní Neo4j
Zdroj: vlastní zpracování

Na obrázku je vidět konzole z webového prostředí databáze Neo4j, na níž vidíme výsledek výše napsaného dotazu. Dotaz jako takový vrátí pouze uzel „Tools category", avšak webové prostředí nabízí pokročilou práci s grafem, kde můžeme jednotlivé uzly nebo vazby přesouvat dle svého uvážení a dělat tak graf čitelnější. Po poklepání na daný uzel máme několik možností. Jednou z nich je rozšířit daný uzel, čímž se nám objeví uzly a vazby, které jsou na tento uzel navázané, což je i situace na obrázku. Mimo jiné na levé straně konzole máme možnost výstupu v tabulce, čistém textu nebo ve formátu JSON.

Dotaz se složitějšími podmínkami, který vrací, jen určitý atribut může vypadat následovně:

```
MATCH (lighttools:Product) WHERE lighttools.weight > 1 AND lighttools.weight < 4 RETURN lighttools.title
```

Dotaz vrátí všechny názvy produktů, které jsou těžší než jeden kilogram a zároveň lehčí než čtyři kilogramy. Na tomto dotazu už je vidět využití proměnné.

Skutečná síla grafových databází však přichází, až když potřebujeme vypočítat nejkratší cestu nebo třeba najít spojení mezi dvěma uzly za použití maximálně N uzlů. Typickým příkladem z reálného života může být nabízení podobných produktů nebo doporučování přátel „co byste mohli znát" na sociálních sítích. Pro

demonstraci tohoto dotazu nepostačí databáze znázorňující produkty a kategorie, takže si dovolím použít testovací dotaz z Neo4j. [53]

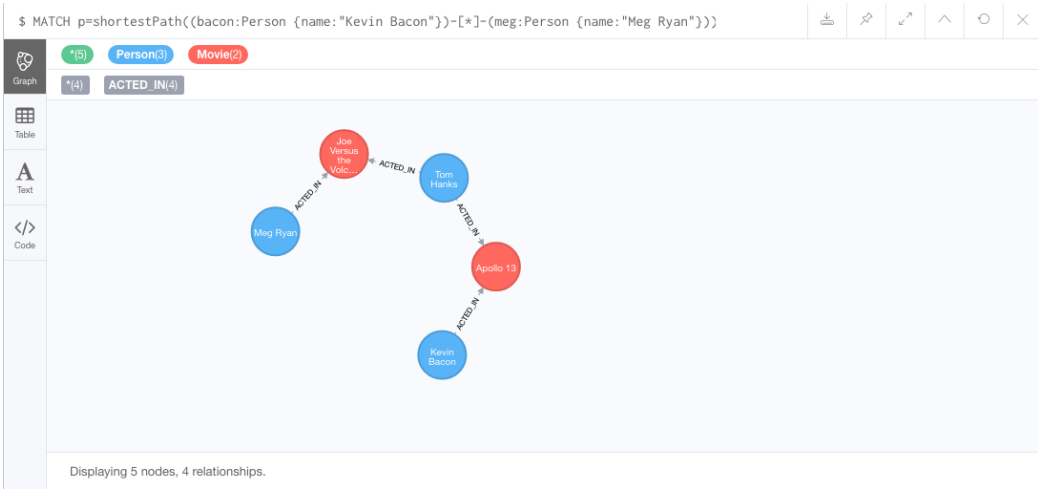
MATCH (bacon:Person {name:"Kevin Bacon"})-[*1..4]-(hollywood) RETURN DISTINCT hollywood

Databáze ukládá filmy a lidi, kteří se na daných filmech podíleli. Jejich práce na filmu je definována vazbou, takže možné vazby jsou „ACTED_IN", „DIRECTED" atd. Dotaz vrací všechny filmy a zainteresované lidi, kteří jsou vzdáleni maximálně přes čtyři vazby od herce jménem Kevin Bacon.

Dotaz pro nalezení nejkratší cesty od Kevina Bacona k herečce se jménem Meg Ryan by pak vypadal následovně:

MATCH p=shortestPath((bacon:Person {name:"Kevin Bacon"})-[*]-(meg:Person {name:"Meg Ryan"})) RETURN p

Výsledek takového dotazu je vidět na následujícím obrázku:



Obrázek 19 Výsledek dotazu pro nalezení nejkratší cesty od Kevina Bacona
Zdroj: Vlastní zpracování

Pro zobrazení kompletního grafu postačí příkaz:

MATCH (n) RETURN n

Pro smazání všeho pak příkaz:

MATCH (n) DETACH DELETE n

5 Porovnání jednotlivých databází

Porovnání jednotlivých databází probíhalo ve dvou rovinách. V první rovině jsou databáze porovnávány podle funkcionality, kterou nabízejí jako je možnost replikace, partitioning, podpora programovacích jazyků a další obecné aspekty databází. V druhé rovině byly porovnány databáze Redis, Apache Cassandra a Neo4j za pomoci technologie YCSB.

Všechny databáze byly instalovány a testovány na zařízení MacBook Pro s procesorem 2,7GHy Intel Core i5, pamětí 16GB 1867 MHy DDR3 a operačním systémem macOS Sierra ve verzi 10.12.4.

5.1 Obecné vlastnosti

V následující tabulce je přehled nabízených funkcionalit, co zkoumané čtyři databáze nabízejí. K porovnání byla použita webová aplikace dostupná na adrese www.db-engines.com, která umožňuje porovnat jednotlivé databáze vůči sobě navzájem. Aplikace dále obsahuje hodnocení jednotlivých databázových systémů.

Tabulka 3 Přehled nabízených funkcionalit

DB	Redis	Apache Cassandra	MongoDB	Neo4j
Model	Key - value	Wide – column store	Document store	Graph DBMS
Implementováno v jazyce	C	Java	C++	Java, Scala
License	Open source	Open source	Open source	Open source
Postaveno na cloudu	Ne	Ne	Ne	Ne
Schéma	Bez schématu	Bez schématu	Bez schématu	Bez schématu
API	RESP – REdis Serialization Protocol	CQL, API postaveno na Apache Thrift	Vlastní API používající JSON	Cypher, Java API, Neo4j-OGM, RESTful HTTP API, Spring Data Neo4j, TinkerPop 3
Podpora programovacích jazyků	C, C#, C++, Clojure, Crystal, D, Dart, Elixir, Erlang, Fancy, Go, Haskell, Haxe, Java, JavaScript,	C#, C++, Clojure, Erlang, Go, Haskell, Java, JavaScript info, Perl, PHP, Python, Ruby, Scala	Actionscript info, C, C#, C++, Clojure info, ColdFusion info, D info, Dart info,	.Net, Clojure, Elixir, Go, Groovy, Haskell, Java, JavaScript, Perl, PHP,

	(Node.js), Lisp, Lua, MatLab, Objective-C, OCaml, Perl, PHP, Prolog, Pure Data, Python, R, Rebol, Ruby, Rust, Scala, Scheme, Smalltalk, Tcl		Delphi, info, Erlang, Go info, Groovy info, Haskell, Java, JavaScript, Lisp info, Lua info, MatLab info, Perl, PHP, PowerShell info, Prolog info, Python, R info, Ruby, Scala, Smalltalk info	Python, Ruby, Scala
Skriptování na straně serveru	Lua scripting	Ne	Javascript	Ano – uživatelsky definované funkce a procedury
Triggery	Ne	Ano	Ne	Ano – event handler
Partitioning	Sharding	Sharding	Sharding	Ne
Replication	Master – slave	Nastavitelný replikační faktor	Master – slave	Kauzální clusterování pomocí Raft protokoli (Pouze v Enterprise verzi)
Podpora MapReduce	Ne	Ano	Ano	Ne
Konzistence	Eventual consistency	Eventual consistency / Immediate Consisteny (Podle nastavených pravidel při zápisu dat)	Eventual consistency / Immediate Consisteny (Podle nastavených pravidel při zápisu dat)	Immediate consisteny, v případě standalone mode, Eventual consistency, v případě rozdělení do clusterů
Podpora cizích klíčů	Ne	Ne	Ne	Ano
Podpora transakčního zpracování	Zamykání, podpora atomicity příkazů	Pouze atomicita a izolovanost pro jednoduché operace	Pouze atomické operace uvnitř jednoho dokumentu	Plná podpora ACID
Životnost /	Nastavitelný	Ano	Ano –	Ano

persistence	mechanismus snapshotů a operačních logů		nastavitelné	
Konkurentní manipulace dat	Ano (Datový přístup serializován stranou serveru)	Ano	Ano	Ano
Podpora udržování dat v paměti	Ano	Ne	Od verze 3.2	Ne
Uživatelské oprávnění	Přístup chráněný heslem	Přístupová práva nastavitelná pro každý objekt	Přístupová práva uživatele a jejich role	Přístupová práva uživatele, role a oprávnění, externí plugin obsahující autentifikaci za pomoci LDAP serveru / active directory

Zdroj: Vlastní zpracování podle stránky www.db-engines.com

5.2 Porovnání podle YCSB

YCSB pochází ze zkratky Yahoo! Cloud System Benchmark. Jedná se o technologii, která byla uvedena v roce 2010 společností Yahoo!. Framework byl navržen pro vyhodnocení výkonu počítačových programů. Postupem vývoje se zaměřil převážně na NoSQL databáze a jejich výkon. Technologie obsahuje šest základních testů, které jsou pojmenovány jako „Workload A“ až „Workload E“. Tyto testy byly provedené na databáze Redis, Apache Cassandra a MongoDB. YCSB neobsahuje podporu pro Neo4j a porovnávání by ani nedávalo smysl, vzhledem ke specifickému použití Neo4j, v systémech, kde jsou zapotřebí algoritmy z teorie grafů. [54]

Každý test obsahuje tabulku a graf. V tabulce jsou uvedené údaje jako průměrná latence, nejmenší latence, největší latence, počet operací, průměrná propustnost a celková doba testu. Graf obsahuje vývoj testu, kde na ose X je vidět vývoj v čase po deseti vteřinách a na ose Y je počet provedených operací.

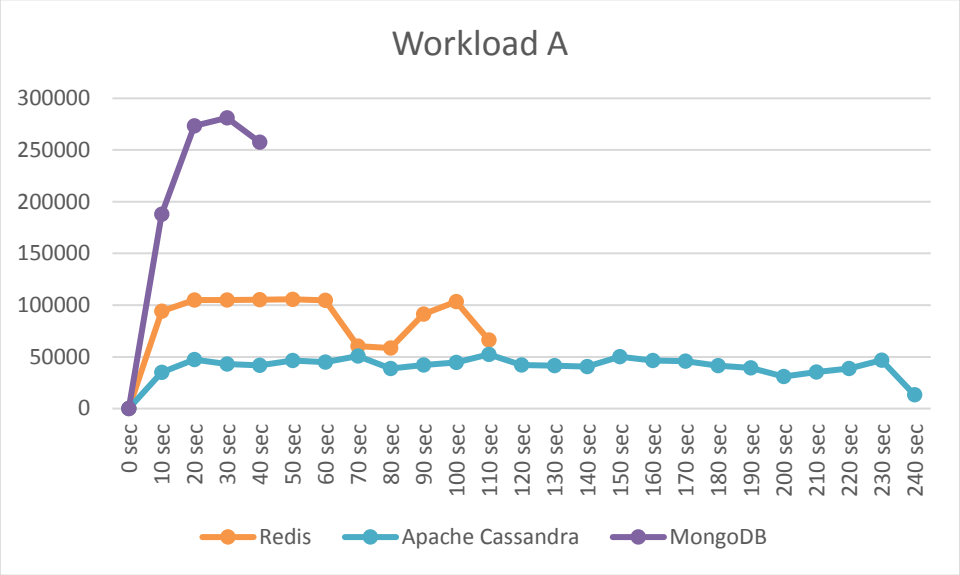
5.2.1 Workload A

Tento test obsahuje operace čtení / zápis v poměru 50 / 50. Uváděným příkladem podle YCSB je úložiště relací zaznamenávající nedávné akce.

Tabulka 4 Workload A

DB	Redis	Apache Cassandra	MongoDB
Doba trvání (ms)	106295	232811	39209
Průměrná propustnost (opt./sekundu)	9407.78	4295.33	25504.35
Minimální latence (µs)	82	146	22
Průměrná latence (µs)	104.48	229.32	37.3
Maximální latence (µs)	8791	77247	48447

Zdroj: Vlastní zpracování



Graf 1 Porovnání NoSQL databází Workload A
Zdroj: Vlastní zpracování

Nejlépe v tomto testu obstála jednoznačně databáze MongoDB, která dokázala milion operací provést v čase 39,2s, což je sedmkrát tolik, než její hlavní konkurent Apache Cassandra.

U databáze Redis si lze povšimnout propadu výkonu, pravděpodobně z důvodu nedostatku paměti v danou dobu.

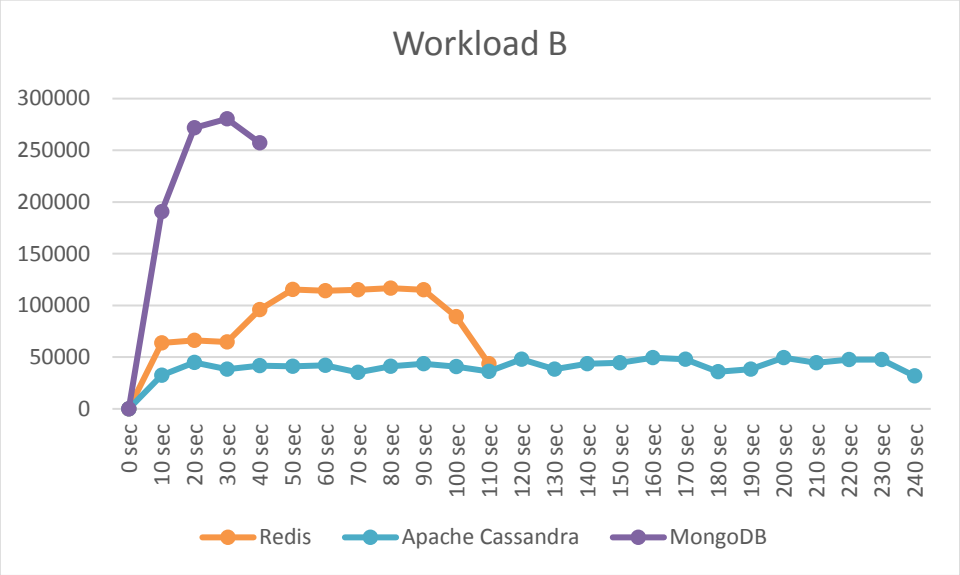
5.2.2 Workload B

Tento test obsahuje operace čtení / zápis v poměru 95 / 5. Uváděným příkladem podle YCSB je správa tagů u fotografií, kde přidání tagu je bráno jako update operace a většina operací je pak právě čtení.

Tabulka 5 Workload B

DB	Redis	Apache Cassandra	MongoDB
Doba trvání (ms)	106655	237246	39156
Průměrná propustnost (opt./sekundu)	9376.03	4215.03	25538.87
Minimální latence (µs)	77	148	22
Průměrná latence (µs)	104.75	233.53	37.19
Maximální latence (µs)	12079	116543	47615

Zdroj: Vlastní zpracování



Graf 2 Porovnání NoSQL databází Workload B
Zdroj: Vlastní zpracování

V testu číslo dva, který je určen převážně jako čtení, zvítězila také databáze MongoDB s podobnými výsledky jako v testu číslo jedna.

Redis v tomto testu dopadl, také podobně, avšak bez výpadku paměti.

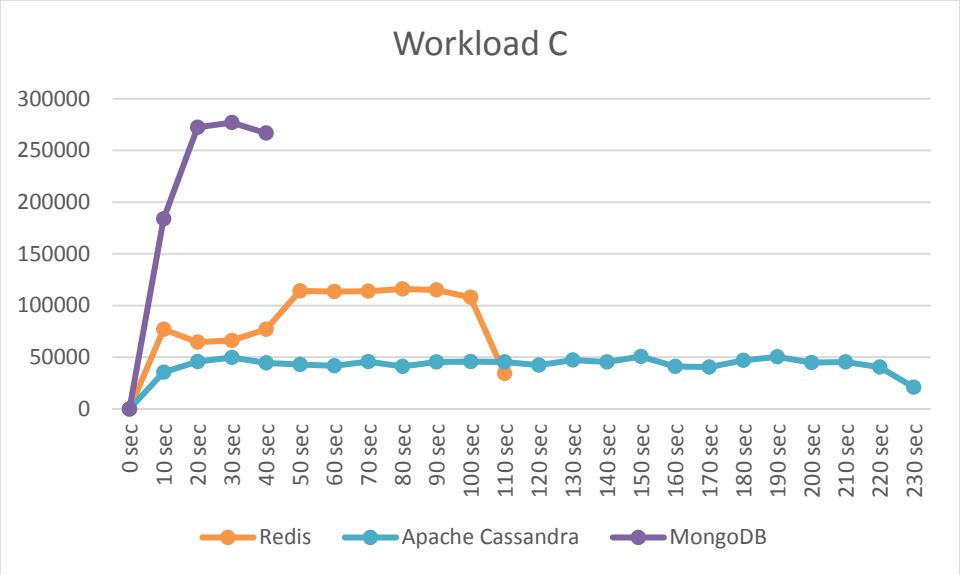
5.2.3 Workload C

Tento test obsahuje pouze operace typu čtení. Uváděným příkladem podle YCSB je cache uživatelských profilů, kde tvorbu profilů zajišťuje jiná technologie.

Tabulka 6 Workload C

DB	Redis	Apache Cassandra	MongoDB
Doba trvání (ms)	105240	224729	39636
Průměrná propustnost (opt./sekundu)	9502.09	4449.8	25229.59
Minimální latence (µs)	78	146	23
Průměrná latence (µs)	103.45	221.41	37.67
Maximální latence (µs)	15527	116095	45247

Zdroj: Vlastní zpracování



Graf 3 Porovnání NoSQL databází Workload C
Zdroj: Vlastní zpracování

Třetí test zaměřený čistě na čtení dopadl podobně jako druhý, pro všechny typy databází, s tím že Redis v danou chvíli provádění testu bojoval trošku víc s pamětí, než v testu předešlém.

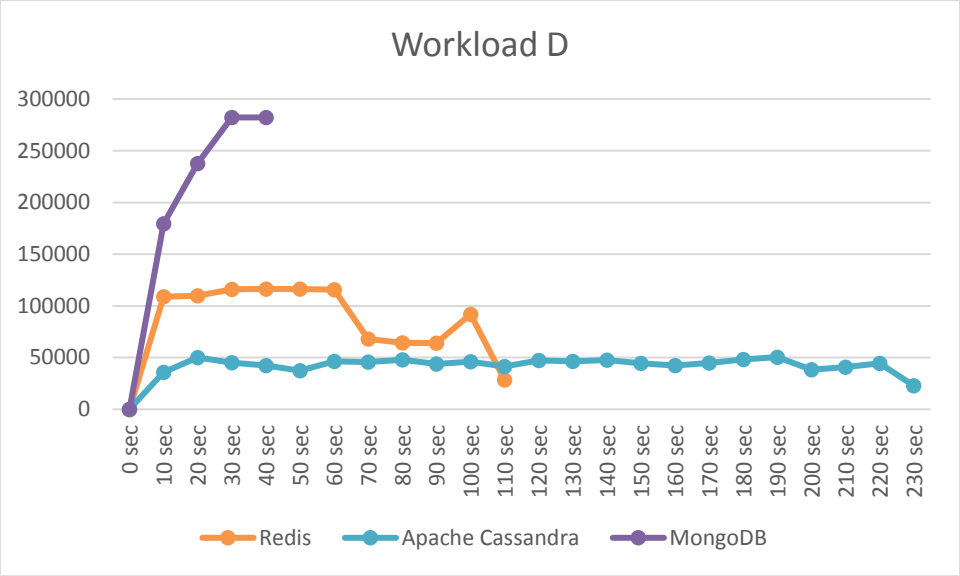
5.2.4 Workload D

Tento test simuluje čtení posledně vložených dat. Vloží se několik záznamů a nejvíce čtené jsou ty poslední vložené. Typickým použitím můžou být uživatelské statusy, kde nejchtěnější je ten poslední.

Tabulka 7 Workload D

DB	Redis	Apache Cassandra	MongoDB
Doba trvání (ms)	102442	227335	40694
Průměrná propustnost (opt./sekundu)	9761.62	4398.79	24573.65
Minimální latence (µs)	76	145	22
Průměrná latence (µs)	100.67	222.82	38.67
Maximální latence (µs)	11775	67007	52543

Zdroj: Vlastní zpracování



Graf 4 Porovnání NoSQL databází Workload D
Zdroj: Vlastní zpracování

Čtvrtý test simulující čtení posledních dat dopadl nejlépe pro databázi MongoDB, která dokázala test dokončit za 40,7s. Databázi Apache Cassandra opět trvalo takřka šestkrát tak dlouho test dokončit.

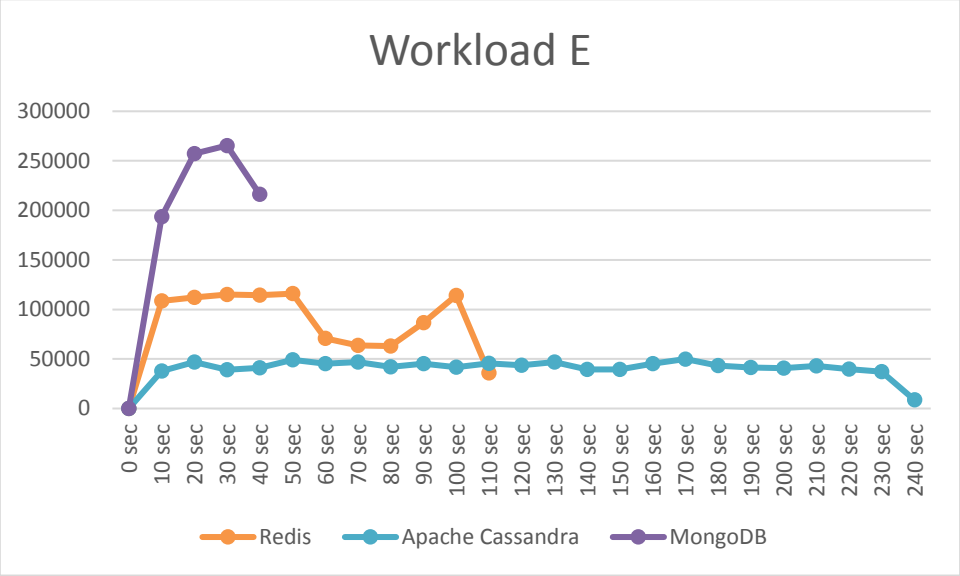
5.2.5 Workload E

Tento test simuluje krátké rozsahy, které jsou dotazovány společně na místo individuálních dotazů. Příkladem můžou být vlákna konverzací, kde jsou dohledávány odpovědi k danému vláknu konverzace.

Tabulka 8 Workload E

DB	Redis	Apache Cassandra	MongoDB
Doba trvání (ms)	103106	232168	43161
Průměrná propustnost (opt./sekundu)	9698.76	4307.23	23169.06
Minimální latence (µs)	77	149	23
Průměrná latence (µs)	101.34	228.88	41.04
Maximální latence (µs)	16703	93759	53343

Zdroj: Vlastní zpracování



Graf 5 Porovnání NoSQL databází Workload E
Zdroj: Vlastní zpracování

Při práci s krátkými rozsahy namísto individuálních dotazů také vyhrává databáze MongoDB, která test dokončila za 43,1s.

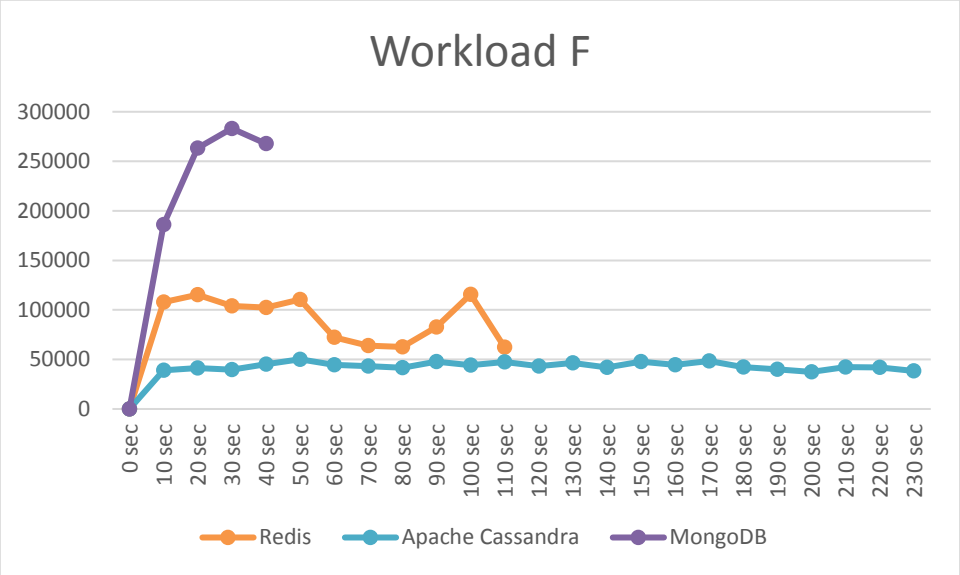
5.2.6 Workload F

V tomto testu se klient dotáže na záznam, upraví ho a zapíše změny. Typickým příkladem je uživatelská databáze, kde jsou uživatelé a jejich aktivity dotazovány i upravovány.

Tabulka 9 Workload F

DB	Redis	Apache Cassandra	MongoDB
Doba trvání (ms)	105537	229774	39474
Průměrná propustnost (opt./sekundu)	9475.35	4352.1	25333.13
Minimální latence (µs)	77	148	22
Průměrná latence (µs)	103.71	226.45	37.41
Maximální latence (µs)	13295	65503	53151

Zdroj: Vlastní zpracování



Graf 6 Porovnání NoSQL databází Workload F
Zdroj: Vlastní zpracování

V posledním testu zaměřeném na uživatelské databáze vítězí databáze MongoDB s časem 39,4s.

6 Závěry a doporučení

Na základě výše získaných výsledků se jeví databáze MongoDB jako zcela jasný vítěz, což také nesporně je. Ve všech testech dosáhla tato databáze nejlepších výsledků, které několikanásobně předčily výsledky z konkurenční databáze Apache Cassandra i odlehčené databáze Redis.

Ačkoliv čísla mluví jasně, nelze brát výsledky testu jako rozhodující. A už vůbec nelze s jistotou říci, že databáze MongoDB je lepší, než Apache Cassandra. Stále platí, že MongoDB je dokumentová databáze ukládající data ve formátu JSON, která se hodí v systémech, kde pracujeme právě s formátem JSON. Kde si na příklad jednotlivé moduly systému mezi sebou posílají zprávy definované v tomto formátu a je tedy přirozenější takový formát zpráv ukládat a následně číst, po případě modifikovat. MongoDB se hodí na místech, kde je počet dat, tak velký, že tradiční SQL databáze přestávají stačit. Dalším faktorem při výběru této databáze je potřeba volného přidávání sloupců a tím dodržení přístupu bez pevné struktury. V případě tedy, že je pro systém potřeba databáze, která bude uchovávat velké množství dat, systém komunikuje pomocí zpráv ve formátu JSON, bude nad ní provozováno dynamické dotazování a je zapotřebí dynamicky přidávat sloupce, pak se MongoDB jeví jako ideální kandidát.

V případě potřeby aktualizace stávajícího systému, kde funguje nějaká z tradičních SQL databází, na systém bez pevné struktury dat, se může ukázat jako ideální přístup zvolit databázi PostgreSQL s jejím datovým typem JSONB, která umožní hladký přechod z důvodu zachování kompatibility SQL a zároveň poskytne možnost dynamicky, nejen přidávat, ale i dotazovat nově vzniklé sloupce. Tím dojde k ušetření času a zdrojů při implementaci do existujícího systému, protože systém bude zpětně kompatibilní, zvláště při využití některé z ORM technologií, v které pak v podstatě stačí změnit řadič dané databáze.

Oproti tomu databáze Apache Cassandra je robustní sloupcová databáze, která sice taky nemá pevně danou strukturu, ale hodí se na obří systémy replikované do několika desítek uzlů, fungující po celém světě, které převládají svým velkým počtem zápisů, po případě čtení. U Apache Cassandra je operace typu modifikace tabu, z důvodu možné silné nekonzistence popsané v kapitole Apache Cassandra.

Tato databáze se tedy hodí především na systémy, do kterého je potřeba ukládat záznamy v reálném čase, jako je třeba online analýza zátěže v reálném čase. Dalším ukazatelem indikujícím potřebu databáze Apache Cassandra je potřeba technologie Apache Hadoop. V případě, že je v systému potřeba dělat distribuované paralelní výpočty za pomocí MapReduce z Apache Hadoop, pak se většinou ukáže, že Apache Cassandra je správnou volbou.

Databáze Redis se s výše zmíněnými nedá moc porovnávat z důvodu, že Redis je koncipován jako odlehčená malá databáze typu klíč-hodnota sloužící nejčastěji jako fronta, žebříček, cache celých stránek nebo třeba správce konfigurací pro jednotlivé systémy na daném serveru. Z těchto důvodů je tedy patrné, že nebude moc často docházet k rozhodování mezi Redisem a Apache Cassandrou nebo Redisem a MongoDB, ale spíše právě mezi Apache Cassandra a MongoDB. Častěji se tedy stane, že v jednom systému uvidíme použitý jak Redis, tak Apache Cassandra nebo MongoDB.

Neo4j je typickým představitelem kategorie grafových databází a nedává žádný smysl ji porovnávat se třemi výše zmíněnými databázovými systémy, protože je navržena pro systémy, které potřebují pracovat s algoritmy spadajícími do teorie grafů, jako je Dijkstrův algoritmus, algoritmus pro výpočet nejkratší cesty, problém čínského pošťáka, a další. Neo4j je nedistribuovaná databáze, čímž se také liší od výše zmíněných databází. Možnost distribuce by v tomto případě nedávala ani smysl. Grafová databáze Neo4j se tedy hodí do systémů jako je právě Jízdomat zmíněný v kapitole o Neo4j, kde se provádí operace jako je zjištění počtu vazeb, přes které se lze dostat od osoby A k osobě B nebo právě algoritmus nejkratší cesty.

Je nanejvýš rozumné dbát správnému výběru databáze, protože přechod na jiný druh databázového systému v aplikaci, která běží v produkčním prostředí, může být až nereálný problém. Při výběru je tedy dobré uvědomit si dva základní pilíře z CAP teorému, které jsou pro cílový systém hlavní. Sepsat seznam nejčastěji prováděných dotazů, uvědomit si strukturu dat ukládaných do databáze a povahu dotazů, ověřit podporu operačního systému a programovacího jazyka a v neposlední řadě ověřit, jestli už někdo danou databázi tímto stylem používá

a jaké ohlasy na ni má. Jako další faktor se může jevit podpora cloudu pro danou databázi a cena na kterou provoz takové databáze přijde.

Diplomová práce měla za cíl zanalyzovat kategorie NoSQL databází, a to konkrétně databáze typu klíč-hodnota, sloupcové databáze, dokumentové databáze a grafové databáze. Dále provést jejich porovnání a následné doporučení. Z každé kategorie byl vybrán jeden zástupce, na němž byl demonstrován postup instalace a praktická práce s databází. V závěru byly jednotlivé databáze porovnány za pomoci technologie YCSB, kde bylo provedeno šest zátěžových testů simulujících různé situace.

Prvním možným rozšířením zkoumaného problému se jeví možnost přidání dalších databází. Konkrétně pak přidání několika databází spadajících do jedné kategorie, následné provedení zátěžových testů a vyhodnocení výsledků. Druhým možným rozšířením je analýza jednotlivých databází na hlubší úrovni a provádění pokročilých dotazů

7 Seznam použité literatury

[1] DANEL, Roman. Databázové systémy [online]. 2016, 40 [cit. 2017-07-12]. Dostupné z: http://slideplayer.cz/slide/11730182

[2] Základy relačních databází: jejich využití v programování webu. Grafická a multimediální laboratoř VŠE [online]. Praha: Grafická a multimediální laboratoř VŠE, 2014 [cit. 2017-07-12]. Dostupné z: http://gml.vse.cz/data/oppa-webdesign/zaklady-db.html

[3] BATKO, Michal. Relační vs. objektově-relační vs. objektové databáze. Výkonnost objektových databází [online]. 2014, (1), 2 [cit. 2017-07-12]. Dostupné z: http://www.fi.muni.cz/~xbatko/oracle/compare.html

[4] PROCHÁZKA, Jaroslav. Objektově orientované databáze. Databázový svět [online]. 2013, 2 [cit. 2017-07-12]. Dostupné z: http://www.dbsvet.cz/view.php?cisloclanku=2004030301/

[5] ŽÁK, Karel. Historie relačních databází. Software [online]. 2001, (1), 3 [cit. 2017-07-12]. Dostupné z: https://www.root.cz/clanky/historie-relacnich-databazi/

[6] DUBEN, Stanislav. Základy jazyka SQL a databází. Základy jazyka SQL a databází [online]. 2007, (1), 3 [cit. 2017-07-12]. Dostupné z: http://duben.org/zaklady-sql/zaklady-jazyka-sql-a-databazi-i-dil

[7] BERRY, Douglas. ACID Properties. Database concepts and standards [online]. 2010, (1), 1 [cit. 2017-07-12]. Dostupné z: http://www.service-architecture.com/articles/database/acid_properties.html

[8] TONIGHT, Study. Normalization of Database. Database Concept [online]. 2015, (1), 5 [cit. 2017-07-12]. Dostupné z: http://www.studytonight.com/dbms/database-normalization.php

[9] Database - Normal Forms. SQL - RDBMS Concepts [online]. Tutorialspoint., (1), 5 [cit. 2017-07-12]. Dostupné z: https://www.tutorialspoint.com/sql/first-normal-form.htm

[10] STROZZI, Carlo. NoSQL: A Relational Database Management System. NoSQL [online]. 2007, (1), 2 [cit. 2017-07-1s2]. Dostupné z: http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page

[11] FAZAL, Talha. Relational Data vs Non Relational Database [online]. 2015, 2 [cit. 2017-07-12]. Dostupné z: https://github.com/talhafazal/DataBase/wiki/Home-Work-%23-3-Relational-Data-vs-Non-Relational-Database..

[12] HOLUBOVÁ, Irena, Jiří KOSEK, Karel MINAŘÍK a David NOVÁK. Big Data a NoSQL databáze. Praha: Grada, 2015. Profesionál. ISBN 978-80-247-5466-6.

[13] ROUSE, Margaret a Stephen BIGELOW. What is Big Data? Big Data [online]. 2016, 4 [cit. 2017-07-12]. Dostupné z: http://searchcloudcomputing.techtarget.com/definition/big-data-Big-Data

[14] Big data. Gartner [online]. USA: Gartner [cit. 2017-07-12]. Dostupné z: http://www.gartner.com/it-glossary/big-data

[15] 4 V's of big data. In: IBM big data hub [online]. USA: IBM, 2010 [cit. 2017-07-12]. Dostupné z: http://www.ibmbigdatahub.com/sites/default/files/infographic_file/4-Vs-of-big-data.jpg

[16] MAYER-SCHÖNBERGER, Viktor a Kenneth CUKIER. Big Data. Brno: Computer Press, 2014. ISBN 978-80-251-4119-9.

[17] BUGHIN, Jacques. Clouds, big data, and smart assets: Ten tech-enabled business trends to watch [online]. 2010, 14 [cit. 2017-07-12]. Dostupné z: http://www.itglobal-services.de/files/100810_McK_Clouds_big_data_and%20smart%20assets.pdf

[18] BEAL, Vangie. ETL - Extract, Transform, Load. In: Webopedia [online]. [cit. 2017-07-12]. Dostupné z: http://www.webopedia.com/TERM/E/ETL.html

[19] BEAL, Vangie. Data mart. In: Webopedia [online]. [cit. 2017-07-12]. Dostupné z: http://www.webopedia.com/TERM/D/data_mart.html

[20] TICHÝ, Jan. WEBINÁŘ KEBOOLA A GOODDATA V KOSTCE [online]. 2015, 1(1), 17 [cit. 2017-07-13]. Dostupné z: https://www.medio.cz/keboola

[21] JIC. Firmy dokážou předvídat budoucnost: Pomůžou s tím Big Data [online]. 2015, 2 [cit. 2017-07-13]. Dostupné z: https://www.jic.cz/magazin/firmy-dokazou-predvidat-budoucnost-pomuzou-s-tim-big-data

[22] BRUIN, Simone. Big Data Brotherhood. In: TheMETISfiles [online]. 2011 [cit. 2017-07-13]. Dostupné z: http://www.themetisfiles.com/2013/04/big-data-brotherhood

[23] BROWNE, Julian. Brewer's CAP Theorem [online]. 2009, 10 [cit. 2017-07-13]. Dostupné z: http://www.julianbrowne.com/article/viewer/brewers-cap-theorem

[24] BREWER, Eric. Towards Robust Towards Robust Distributed Systems [online]. 2000, 45 [cit. 2017-07-13]. Dostupné z: https://people.eecs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

[25] GANESAN, Lakshminarayanan. What is the difference between replication, partitioning, clustering, and sharding? [online]. 2014, 1 [cit. 2017-07-13]. Dostupné z: https://www.quora.com/What-is-the-difference-between-replication-partitioning-clustering-and-sharding

[26] Ian. What is a Key-Value Database? [online]. Database.Guide, 5 [cit. 2017-07-13]. Dostupné z: http://database.guide/what-is-a-key-value-database

[27] Aerospike. What is a Key-Value Store? [online]. Database.Guide, 2017,4 [cit. 2017-07-13]. Dostupné z: http://www.aerospike.com/what-is-a-key-value-store/

[28] Redis. Documentation [online]. Database.Guide, 2017,60 [cit. 2017-07-13]. Dostupné z: https://redis.io/documentation

[29] JOSHI, Leena. How Redis is Used in Practice [online]. 2015,4 [cit. 2017-07-13]. Dostupné z: https://dzone.com/articles/how-redis-is-used-in-practice

[30] ŠTRAUCH, Adam. Redis: key-value databáze v paměti i na disku. Databáze [online]. 2010, 8 [cit. 2017-07-13]. Dostupné z: https://www.zdrojak.cz/clanky/redis-key-value-databaze-v-pameti-i-na-disku

[31] MONTROSE, Kevin. Does Stack Exchange use caching and if so, how? [online]. 2016, 1 [cit. 2017-07-13]. Dostupné z: https://meta.stackexchange.com/questions/69164/does-stack-exchange-use-caching-and-if-so-how/69172#69172

[32] GIOR, Enrico. Redis on Windows [online]. 2015, 1 [cit. 2017-07-13]. Dostupné z: https://github.com/MSOpenTech/redis

[33] Apache Cassandra [online]. Apache, 2012 [cit. 2017-07-14]. Dostupné z: http://cassandra.apache.org

[34] Netflix: Netflix Personalizes Viewing for Over 50 Million Customers with DataStax. [online]. 2017, 2 [cit. 2017-07-14]. Dostupné z: http://www.datastax.com/personalization/netflix

[35] Thoughtworks. Cassandra carefully [online]. 2017,3 [cit. 2017-07-14]. Dostupné z: https://www.thoughtworks.com/radar/platforms/cassandra-carefully

[36] DataStax. DataStax documentation: CREATE KEYSPACE [online]. 2016, 10 [cit. 2017-07-14]. Dostupné z: http://docs.datastax.com/en/cql/3.3/cql/cql_reference/cqlCreateKeyspace.html

[37] DataStax. DataStax documentation: CREATE AND RUN QUERIES WITH DATASTAX DEVCENTER [online]. 2014, 2 [cit. 2017-07-14]. Dostupné z: http://www.datastax.com/wp-content/themes/datastax-2014-08/getting-started/DataStax-Getting-Started-Guide-DevCenter.pdf

[38] Apache. Welcome to Apache™ Hadoop®!: What Is Apache Hadoop? [online]. 2009, 3 [cit. 2017-07-14]. Dostupné z: http://hadoop.apache.org/

[39] GILMORE, ERIC. Hadoop MapReduce in the Cassandra Cluster [online]. 2011, 3 [cit. 2017-07-14]. Dostupné z: https://www.datastax.com/dev/blog/hadoop-mapreduce-in-the-cassandra-cluster

[40] THON, Ladislav. Lehký úvod do MongoDB. *NoSQL databáze* [online]. 2010, 6 [cit. 2017-08-02]. Dostupné z: http://www.abclinuxu.cz/clanky/programovani/lehky-uvod-do-mongodb

[41] SENTAN. MongoDB - závislost za 30 minut!. *Nestrukturovaná data - NoSql experimenty* [online]. 2015,4 [cit. 2017-08-02]. Dostupné z: https://blog.root.cz/sentan/mongodb-zavislost-za-30-minut/

[42] *Mongo DB* [online]. USA: DoubleClick, 2007 [cit. 2017-08-02]. Dostupné z: https://www.mongodb.com

[43] KUBICA, Tomáš. NoSQL opravdu snadno. *Praktický úvod do MongoDB* [online]. 2015, 10 [cit. 2017-08-02]. Dostupné z: http://www.cloudsvet.cz/?p=249

[44] *Big Data: Algorithms for Data Preprocessing, Computational Intelligence, and Imbalanced Classes* [online]. 2003 - 2017, 2017 [cit. 2017-08-08]. Dostupné z: http://sci2s.ugr.es/BigData

[45] *Business Intelligence* [online]. 2004 [cit. 2017-08-08]. Dostupné z: http://slideplayer.cz/slide/5645907

[46] DANEL, Roman. Databázové systémy [online]. 2016, 40 [cit. 2017-07-12]. Dostupné z: http://slideplayer.cz/slide/11730182

[47] Základy relačních databází: jejich využití v programování webu. Grafická a multimediální laboratoř VŠE [online]. Praha: Grafická a multimediální laboratoř VŠE, 2014 [cit. 2017-07-12]. Dostupné z: http://gml.vse.cz/data/oppa-webdesign/zaklady-db.html

[48] BATKO, Michal. Relační vs. objektově-relační vs. objektové databáze. Výkonnost objektových databází [online]. 2014, (1), 2 [cit. 2017-07-12]. Dostupné z: http://www.fi.muni.cz/~xbatko/oracle/compare.html

[49] Object database. In: *Wikipedia* [online]. 2017 [cit. 2017-07-05]. Dostupné z: https://en.wikipedia.org/wiki/Object_database

[50] *Zdroják.cz: Grafová terminologie a dostupné technologie* [online]. 2013 [cit. 2017-08-10]. Dostupné z: https://www.zdrojak.cz/clanky/grafova-terminologie-a-dostupne-technologie/

[51] *SlideShare: Neo4j* [online]. 2013 [cit. 2017-08-10]. Dostupné z: https://www.slideshare.net/bachmanm/neo4j-introcz

[52] *Neo4j* [online]. 2017 [cit. 2017-08-10]. Dostupné z: www.neo4j.com

[53] *Informatický večer — Grafové databáze (první oficiální Neo4j Meetup)* [online]. 2014 [cit. 2017-08-10]. Dostupné z: https://www.youtube.com/watch?v=YL1IyWzDvK0&t

[54] Yahoo! Cloud System Benchmark (YCSB). *Github*[online]. [cit. 2017-08-14]. Dostupné z: https://github.com/brianfrankcooper/YCSB/