Pedagogická fakulta
Faculty of Education

Jihočeská univerzita v Českých Budějovicích
University of South Bohemia in České Budějovice

Jihočeská univerzita v Českých Budějovicích
Pedagogická fakulta
Katedra informatiky

Bakalářská práce

NoSQL databáze

Vypracoval: Tomáš Panyko, DiS.
Vedoucí práce: RNDr. Hana Havelková

České Budějovice 2013

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách.

V Českých Budějovicích dne

Poděkování

Rád bych poděkoval RNDr. Haně Havelkové za odborné vedení, připomínky a cenné rady při zpracování této práce.

Anotace

Práce se zabývá méně známými typy databází, tzv. NoSQL databázemi. Nejprve se provede seznámení s problematikou, vysvětlení, co se pod pojmem NoSQL skrývá. Po stručné historii přijde na řadu popis technologie SQL a NoSQL, výčet databázových serverů, se kterými se můžeme setkat a následný výběr tří nejrozšířenějších zástupců NoSQL a jednoho SQL produktu, u kterých proběhne bližší seznámení. NoSQL je nasazeno na velkých serverech, jako je FaceBook a Twitter, proto se s nimi lehce seznámíme. Následuje výběr jednoho zástupce za NoSQL a jednoho za SQL a jejich detailní rozbor. Od instalace a nastavení, práce s nimi, jako je vyhledávání, přidávání záznamů, atd., přes jeho výhody a nevýhody oproti konkurenčním systémům, k porovnání vybraných zástupců podle rychlosti vyhledávání na velkém počtu záznamů s přiloženými grafy.

Abstract

The work is dealing with less known types of databases called NoSQL. At first there is an introduction to the problem, explaining, what NoSQL means. After a short history are descriptions of SQL and NoSQL technologies, enumeration of databases that we can encounter, followed with three of the most common NoSQL databases and one SQL database, which are closer described. NoSQL is deployed on big servers, such as Facebook and Twitter, that is why there is a part about them. Next step is choosing one member for each SQL and NoSQL group and their detailed analysis. From installation and setting, work with them, such as searching, adding records, etc., through their advantages and disadvantages against the competitive systems, to comparing selected members by searching speed, where both of them have a lot of records. The results are in added graphs.

1 Úvod 1
2 Historie 2
3 SQL databáze 4
3.1 Popis 5
3.2 Normální formy 6
3.3 Hlavní představitelé 7
3.4 MySQL 9
3.4.1 Popis 9
3.4.2 Dotazování 11
3.4.3 Datové typy 13
4 NoSQL databáze 14
4.1 Popis 15
4.2 CAP teorém 16
4.3 Škálovatelnost 17
4.4 Sharding 18
4.5 Hlavní představitelé 19
4.5.1 Sloupcově orientované databáze 19
4.5.2 Databáze s klíčem a hodnotou 20
4.5.3 Dokumentové databáze 21
4.5.4 Objektové databáze 22
4.6 HBase 23
4.6.1 Popis 23
4.6.2 Spuštění a připojení k serveru 25
4.6.3 Správa dat 25
4.6.4 Datové typy 29
4.7 MongoDB 30

4.7.1 Popis 30
4.7.2 Spuštění a připojení k serveru 31
4.7.3 Správa dat 33
4.7.4 Datové typy 36
4.8 Redis 37
4.8.1 Popis 37
4.8.2 Spuštění a připojení k serveru 39
4.8.3 Správa dat 39
4.8.4 Datové typy 42
5 NoSQL v praxi 44
5.1 NoSQL a FaceBook 44
5.2 NoSQL a Twitter 45
5.3 NoSQL a Cloud 46
6 Práce s Redisem 47
6.1 Instalace 47
6.2 Nastavení 49
6.3 Práce s daty 51
6.3.1 Persistence dat 51
6.3.2 Pokročilá správa dat 53
6.4 Porovnání s konkurencí 56
6.4.1 Využití 57
6.4.2 Dostupnost materiálů 58
6.4.3 Podpora operačních systému 61
6.4.4 Podpora programovacích jazyků 62
6.4.5 Vyhodnocení 63
7 Práce s MySQL 64

7.1 Instalace ... 65
7.2 Nastavení ... 66
7.3 Práce s daty ... 67
7.4 Porovnání s konkurencí ... 71
8 Porovnání Redisu s MySQL ... 72
8.1 Vytvoření datasetu ... 73
8.2 Naplnění Redisu daty ... 74
8.3 Naplnění MySQL daty ... 76
8.4 Měření ... 78
8.4.1 Faktory ovlivňující výkon Redisu ... 79
8.4.2 Přidávání záznamů s parsováním ... 80
8.4.3 Přidávání záznamů bez parsování ... 82
8.4.4 Update záznamů ... 85
8.4.5 Vyhledávání záznamů ... 86
8.5 Celkové vyhodnocení ... 88
9 Závěr ... 88
10 Terminologický slovník ... 89
11 Použitá literatura ... 90

1 Úvod

Pojem databáze dnes zná prakticky každý. Už od pradávna mají lidé potřebu shromažďovat data, ať už jde o seznam knih, evidenci obyvatel nebo třeba i klientů bank. Počty těchto záznamů leckdy přesahují desítky milionů a jsou tak výzvou pro ukládání a analýzu. Současně s rapidním nárůstem dat se také setkáváme s jevem, kdy data začínají být pouze částečně strukturovaná, např. záznamy se liší v počtu atributů. To má za následek, že tradiční správa dat pomocí schémat a relačních referencí přestává být použitelná. Problém velkého objemu dat a jeho různorodosti vedl k potřebě vzniku nových databázových systémů. Těm se celkově říká NoSQL. Tento termín zastřešuje především sloupcově orientované databáze (column-oriented data stores), databáze s klíčem a hodnotou (key/value pair databases) a dokumentové databáze (document databases).

NoSQL je doslovně kombinací dvou slov: No a SQL. Je to tedy technologie, která stojí proti SQL. Zkratka může být matoucí a mezi lidmi neexistuje jednotný názor na to, co znamená. Nejvíce rozšířený je ovšem ten, který tvrdí, že jde o acronym „Not only SQL.“ Ať už zkratka znamená cokoliv, NoSQL je dnes zastřešující název pro všechny databáze, které nejdou cestou známého RDBMS (relační systém řízení báze dat), ale jdou svojí vlastní, často spojovanou s velkými objemy dat.

Tuto práci jsem si vybral kvůli své zálibě v relačních databázích, ve kterých se již pár let pohybuji. V několika svých projektech jsem tyto databáze využíval, snažil se dotazy co nejlépe optimalizovat a vždy jsem obdivoval, jakým způsobem dokážou tak rychle najít v rozsáhlém počtu záznamů jeden konkrétní. Proto, když jsem se dozvěděl o NoSQL databázích, které jsou navržené pro mnohem větší objemy dat, zaujalo mě to a musel jsem se do problematiky ponořit hlouběji.

Cílem mé práce je seznámit čtenáře, kteří mají základní zkušenosti s relačními databázemi, s možnostmi, které nabízejí NoSQL produkty. Ukázat jim, jak se liší od databází, které již znají, co jim přináší nového a naopak jakou daň za to požadují. NoSQL databází, stejně jako jejich protějšků, SQL, je celá řada a nedá se v práci obsáhnout všechny, proto jsem vybral vhodné zástupce, o kterých se zmíním blíže. To se samozřejmě neobejde bez potřebných ukázek práce s nimi. Není opomenuto ani na čtenáře, kteří s databázemi obecně přišli do styku pouze povrchně, proto si kladu za cíl také poukázat na klíčové vlastnosti SQL produktů, aby si i méně zkušený čtenář mohl udělat obrázek o celkové problematice.

V praktické části práce jsem si vybral jeden NoSQL produkt ze zástupců, který je knižně i co se týče internetu nejhůře zdokumentován a rozhodl se mu věnovat více pozornosti, kterou si bezesporu zaslouží. Zaměřil jsem se především na to, jak server nainstalovat, nakonfigurovat, pracovat s ním a také na jeho silné a slabé stránky vůči konkurenčním produktům. Podobným způsobem jsem přistoupil i k SQL, kde jsem pro změnu vybral patrně nejznámější produkt. Abych podpořil tvrzení, že NoSQL systémy jsou určeny pro databáze s větším objemem dat, na závěr jsem otestoval zástupce za NoSQL a SQL proti sobě na databázi, která má 5 milionů záznamů. Jako testovací metody jsem zvolil přidávání, upravování a vyhledávání záznamů.

2 Historie

Již velmi dlouho panuje potřeba uchovávat informace. Za předchůdce databází by se daly považovat kartotéky, které umožňovaly schraňovat velké objemy dat a třídit je podle různých kriterií. Tehdy veškerý vyhledávací čas ležel na bedrech lidí. Čím lépe byla kartotéka organizovaná, tím rychleji člověk našel patřičný záznam.

Člověk je ale tvor pohodlný, proto se pokusil převést část svojí práce na stroje. Poprvé jsme toho mohli být svědky ve větším měřítku u sčítání lidu v roce

1890, kdy pro uchování informací se používalo děrných štítků a samotné sčítání probíhalo na elektromechanických strojích. Tato technika byla celkem populární, proto se jí dařilo přežívat dalších 50 let.

V padesátých letech 20. století přišel velký boom vývoje počítačů. Ty dali impulz pro další rozvoj databázových systémů. Chyběl ovšem vhodný jazyk, pomocí kterého by se zpracovávaly data. Proto v roce 1960 byla publikována první verze jazyka COBOL, která tuto činnost zvládala bravurně.

V druhé polovině 60. let začala vznikat koncepce databázových systémů, jejichž součástí jsou Systémy Řízení Báze Dat (SŘBD), v originále DataBase Management System (DBMS). Šlo vlastně o programovou vrstvu, která řešila operace nad databází. Když to celé zjednodušíme, dalo by se říct, že databázový systém je spojení databáze, schopné uchovávat informace a SŘBD.

Za první SŘBD by se daly považovat síťové, které byly popsány v roce 1971 a které fungovaly na sálových počítačích.

V téže době vznikaly taky hierarchické databáze. První implementací tohoto typu byl IMS – systém navržený firmou IBM pro program Apollo.

V 70. letech začínají vznikat první relační databáze. To vše díky článku, který napsal Edgar Frank „Ted“ Codd. Ten začal na data pohlížet jako na tabulky. O pár let později vznikl dotazovací jazyk SQL, který umožnil pohodlnou práci s daty v těchto databázích.

V 90. letech začaly vznikat objektově orientované databáze. Šlo o alternativu a rozšíření relačních databází. Ty umožňovaly uchovávat objekty, včetně jejich chování. Původní předpoklady byly, že se jim podaří vytlačit relační databáze. K tomu ale nedošlo a místo toho vznikly kompromisní objektově relační databáze.

Nerelační databáze nejsou vůbec novou záležitostí, ve skutečnosti první z nich pochází z dob, kdy byly sestaveny první počítače. Využity byly pro uložení ověřovacích informací a autorizačních oprávnění. Avšak dnešní NoSQL systémy vznikly ve světě masivně škálovatelných internetových aplikací a sdílí se svými předchůdci pouze základní myšlenky.

Poprvé byl termín NoSQL použit v roce 1998 Carlem Strozzim, který tak pojmenoval svoji souborovou databázi. Počátky NoSQL se ale pojí s vyhledávacím enginem Inktomi, který byl později vytlačen Googlem. Ten musel řešit problém s velkým objemem strukturovaných dat. Převažovaly zde stránky, jejich propojení a textové odkazy. Google publikoval článek, ve kterém představil svůj návrh databáze pro velká data, kterou pojmenoval BigTable. Tímto činem odstartoval tvorbu nových databází, které se dnes nazývají NoSQL. Další podstatná změna byla ta, že databáze již nebyla řádkově orientovaná, jako je to běžné u SQL databází, ale sloupcově. Amazon, inspirovaný projektem BigTable, navrhl pro svoji potřebu ukládání informací o svých produktech databázi Dynamo. Stejně jako BigTable byla založena na systému klíč/hodnota (vysvětleno dále). Spolu se tak stali hlavními vzory NoSQL databází, které přišly po nich.[1]

3 SQL databáze

V této kapitole bych rád představil problematiku SQL databází. Poněvadž ale není toto téma cílem mé práce, je k němu mnoho dostupných informací, ať již v tištěné nebo elektronické formě, není třeba zacházet do detailů. Celé se ale opomenout nedá, protože je potřebné pro pochopení souvislostí a porovnávání s NoSQL technologií. Nejdříve je zmíněno, co to vlastně SQL databáze je, jaké jsou její charakteristické vlastnosti, dále přijdou na řadu normální formy, tj., jak by se měly tabulky správně navrhovat. Dále je uvedeno, jaké SQL produkty jsou v současnosti nejrozšířenější a nejznámější, přičemž je následně vybrán MySQL a provedeno s ním základní seznámení.

3.1 Popis

Na úvod je třeba říci, že ne každá SQL databáze je relační databází. SQL databází se rozumí ta, nad kterou se dají vykonávat SQL příkazy. My se v této kapitole budeme zabývat pouze relačními databázemi, které umožňují využití SQL dotazování.

Relační databáze využívají tabulek, do kterých ukládají data. Jednotlivé tabulky mají mezi sebou předem definované vztahy. Sloupce se nazývají atributy, řádky se nazývají záznamy.

Každý sloupec musí mít jedinečný název, v tabulce se nesmí opakovat. Dále se u něj určuje datový typ. To jest typ hodnoty, který se bude do sloupce ukládat. Těchto typů je mnoho a jejich výčet záleží na použitém databázovém systému. Většina jich ale bývá stejná, proto skoro všude naleznete typy jako Text, Varchar, Date a Integer.

Řádek neboli záznam je řezem tabulkou, přes všechny jeho sloupce a slouží k uložení dat. Každý záznam musí být unikátní, kvůli jeho identifikaci.

Atribut u jednoho záznamu ovšem může obsahovat vždy pouze jednu hodnotu. Pro situace, kdy je třeba do atributu u jednoho záznamu vložit více hodnot, musíme je uložit do nové tabulky a propojit ji s hlavní tabulkou pomocí primárních a cizích klíčů.

Primární klíč je atribut, případně atributy, které mají na starosti určení jedinečnosti každého záznamu. Poněvadž tabulky nemají nijak definované pořadí záznamů, musí se vždy v každé tabulce určit jeden nebo více atributů, které zajistí, že je možné při znalosti hodnoty klíče vždy odkázat pouze na jeden záznam.

Cizí klíč je také atribut, který nese stejnou hodnotu jako primární klíč v druhé tabulce.

Obrázek 1 Vazba mezi tabulkami

Na obrázku 1 je k vidění vztah mezi dvěma tabulkami. Tabulka 1 obsahuje 3 atributy. Prvním je id, druhým je nazev, třetím je autor. Id je zde primární klíč (Primary key - PK) a autor je cizí klíč (Foreign key - FK). Druhá tabulka obsahuje také primární klíč id, jehož hodnoty odpovídají hodnotám cizího klíče v první tabulce. Tímto způsobem se dají tabulky spojit.

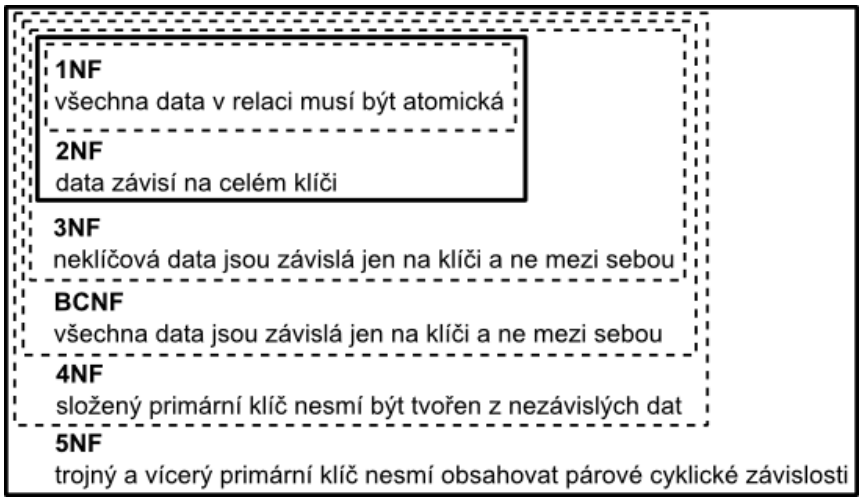
Možná vás napadne, že by to šlo také zapsat vše do jedné tabulky. Tento způsob by byl ale neefektivní, protože by tak narůstal objem databáze, docházelo by zde k redundanci záznamů a celkově by to bylo proti normálním formám.

3.2 Normální formy

Normální formy jsou sada pravidel, podle kterých by měly být navržené tabulky a jejich vztahy mezi nimi. Označují se jako 1NF, 2NF, 3NF, BCNF, 4NF a 5NF. Každá normální forma obsahuje podmínky forem předchozích, plus něco navíc. Minimálně by databáze měla splňovat 1NF, často se ale setkáme i s vyššími stupni.

1NF říká, že by data měla být atomická. To znamená, že by neměla být dále dělitelná. Nejlépe pochopitelné je to na příkladu, kdy bychom chtěli uchovávat adresu člověka. Aby tabulka splňovala 1NF, nesměla by celá adresa být napsaná v jednom atributu (např. Jeremiášova 10, České Budějovice), ale museli bychom vytvořit 3 atributy, z nichž každý by uchovával ulici, číslo popisné a město.

Všechny normální formy a jejich pravidla jsou na obrázku 2.



Obrázek 2 Normální formy [2]

3.3 Hlavní představitelé

V této sekci si vyjmenujeme nejznámější relační SQL databáze a jejich základní informace.

MSSQL (Microsoft SQL Server) je relační SŘBD vyvinutý firmou Microsoft. Pro práci s ním se nejčastěji využívá SQL Server Management Studio, které je součástí instalace MSSQL. To je umožněno díky TDS (Tabular Data Stream). Jde o protokol aplikační vrstvy, který přenáší data mezi databázovým serverem a klientem. Data jsou uložená v databázi s příponou .mdf, soubor s příponou .ndf uchovává metadata a logové údaje jsou zaznamenávány v souboru .ldf.

Verze	Rok vydání	Název	Kódové označení
1.0	1989	SQL Server 1.0 (16bit)	-
1.1	1991	SQL Server 1.1 (16bit)	-
4.21	1993	SQL Server 4.21	SQLNT
6.0	1995	SQL Server 6.0	SQL95
6.5	1996	SQL Server 6.5	Hydra
7.0	1998	SQL Server 7.0	Sphinx
8.0	2000	SQL Server 2000	Shiloh
8.0	2003	SQL Server 2000 64-bit Edition	Liberty
9.0	2005	SQL Server 2005	Yukon
10.0	2008	SQL Server 2008	Katmai
10.25	2010	SQL Azure DB	CloudDatabase
10.5	2010	SQL Server 2008 R2	Kilimanjaro
11.0	2012	SQL Server 2012	Denali

Tabulka 1 Vývojové verze MSSQL Serveru [3]

MySQL je databázový systém, který je distribuovaný pod bezplatnou licencí GPL, tak i pod komerční licencí. Jedná se o velmi oblíbený a často nasazovaný systém a to díky své multiplatformnosti, vysokému výkonu a minimálním finančním nákladům. Více v následující sekci.

Oracle je databázový systém, vyvinutý firmou Oracle Corporation, který podporuje objektově relační databáze. Jejich první databáze vznikla v roce 1979, pod názvem Oracle V2. Pro práci s Oracle databázemi firma dodává grafický nástroj Oracle SQL Developer. Pod záštitou značky Oracle Corporation můžeme nalézt i MySQL, protože předchozí majitel, Sun

Microsystems, byl firmou Oracle Corporation koupen, ale také i Oracle NoSQL Database, což je NoSQL databáze typu klíč-hodnota.[4]

PostgreSQL, často označovaný také jako Postgres, je open source objektově relační systém, který vznikl v roce 1987. Jako rozhraní pro správu databáze se často používá Psql, což je aplikace pro příkazovou řádku, schopná zadávání SQL dotazů s řadou zjednodušujících funkcí a PgAdmin, grafické rozhraní, které práci ještě více zjednodušuje. [5]

SQLite je databázový systém, který je v relativně malé knihovně, napsané v jazyce C. Tento systém není typu klient-server, kdy máme aplikaci komunikující s nainstalovaným SQL serverem, jako je to u většiny serverů, ale jde o součást klientské aplikace. Jinými slovy, nikde Vám neběží žádný speciální server, stačí jen knihovnu připojit k Vaší aplikaci. Toto řešení je velmi oblíbené u malých zařízení s omezenou pamětí, protože knihovna zabere v paměti pouze kolem 25kb.[6]

3.4 MySQL

Jak již bylo zmíněno, MySQL je v dnešní době velmi populární a rozšířený, proto byl vybrán jako zástupce za SQL databáze a bude tak s ním seznámeno podrobněji a později i podstoupí porovnání s konkurenčními SQL systémy a především porovnání s vybraným zástupcem za NoSQL.

3.4.1 Popis

Snad každý, kdo kdy měl něco společného s databázemi, musel narazit i na pojem MySQL. Její začátky se datují k roku 1995, kdy vznikla první verze. Původní myšlenka byla vytvořit rychlou databázi, která neměla obsahovat pokročilejší vlastnosti, jako například cizí klíče. Ovšem díky tomu, že je pod svobodnou licencí a také hlavně proto, že je součástí LAMP (jde o spojení Linuxu, Apache, MySQL a PHP), stala se MySQL nejpoužívanější databází na světě. Takovýto rozmach ovšem způsobil zvýšené požadavky na funkcionalitu

databáze, se kterými se třeba ze začátku vůbec nepočítalo a to může vést například ke zmateným chybovým hlášením nebo nečekaným vlastnostem.

Pro základní práci s MySQL stačí Community Edition, která je zdarma. Za verzi Standard zaplatíte 2000$, Enterprise stojí 5000$ a verze Cluster Carrier Grade vás vyjde na rovných 10000$.[7]

Co tedy v MySQL nalezneme? Například multiplatformní podporu, uložené procedury, triggery, pohledy, transakce, předpřipravené výrazy a nastavení práv uživatel.

Multiplatformní podporou rozumíme plnou funkčnost pod systémem Solaris, Debian, Microsoft XP a výše, Apple, FreeBSD a dalšími. Celý výčet je k dispozici na www.mysql.com/support/supportedplatforms/database.html.[8]

Uložená procedura je sada příkazů, která je uložená v databázi a je zkompilovaná pro rychlejší použití. Proč tomu tak je? Hlavně proto, že uživatelé databáze často provádí operace, které jsou totožné nebo alespoň podobné. Když budou uloženy na serveru, nemusí je uživatel pokaždé vypisovat. Další výhodou je ulehčení správy databázových aplikací. Mějme jako příklad server, který ukládá a zpracovává data. Pokud to celé poběží pomocí uložených procedur, mohou se shodným způsobem vůči databázi chovat klienti pocházející z nejrůznějšího prostředí - desktopová aplikace napsaná v Javě, webový klient v PHP nebo třeba vzdálený server, který s tím "naším" komunikuje. Jestliže je potřeba uloženou proceduru upravit (nebo opravit), může se to udělat na jednom místě a tato změna je ihned použitelná pro všechny aplikace, které s databází komunikují.[9]

Trigger je nástroj, který nám umožňuje provedení určité akce při vložení, smazání nebo změně záznamu v určité tabulce. Příkladem budiž situace, kdy máme databázi se zaměstnanci. Pokud jeden z nich odejde, je třeba smazat kromě samotného zaměstnance i další související záznamy, např. jeho plat

nebo jeho zařazení, které se můžou nacházet v jiné tabulce. Všechny tyto operace je možné provádět i v aplikační vrstvě, ale je snazší a rychlejší je zautomatizovat právě pomocí triggerů.

Pohled je virtuální databázová struktura, která má za cíl jedinou věc. Zjednodušit nám život tím, že z tabulek získáme data, která jsou pro nás relevantní. Jsou to vlastně statické dotazy, které nám filtrují databázi. S pohledem můžeme pracovat jako s tabulkami, takže z nich lze vybírat konkrétní záznamy.

Transakce je skupina úloh, které je třeba vykonat jako jeden funkční celek. Transakce musí splňovat tzv. **ACID** vlastnosti (Atomicity, Consistency, Isolation, Durability) – Atomicita, Konzistence, Izolovanost, Trvalost. Tyto vlastnosti mají garantovat spolehlivost transakce. Atomicitou se rozumí provedení operace jako celku. Konzistence znamená přivedení databáze z jednoho validního stavu do druhého, podle všech definovaných pravidel. Izolovaností se myslí skrytí obsahu transakce před vnějšími operacemi a trvalost znamená, že jakmile je jednou transakce dokončena, zůstanou výsledky uložené i v případě výpadku napájení. Nejlépe pochopitelné je to na příkladě, kdy převádíme peníze z jednoho účtu na jiný. Nejdříve se musí peníze z jednoho účtu odečíst a pak na druhý přičíst. Nesmí dojít k tomu, že by jeden z kroků nebyl vykonán a peníze tak zmizely nebo se naopak z ničeho nic objevily. Proto celý převod musí být brán jako celek, který je buď proveden úspěšně, nebo jsou všechny změny vráceny do předchozího stavu.

3.4.2 Dotazování

Pro dotazování do MySQL databáze můžeme využít jazyk SQL – Structured Query Language (Strukturovaný dotazovací jazyk).

Historie jazyka sahá do 70. a 80. let. V roce 1986 byl přijat první standard, označovaný jako SQL86. V roce 1992 vyšla jeho opravená verze, s označením

SQL92. Následné verze byly pod označením SQL:1999, SQL:2003, SQL:2008 a SQL:2011. [1]

Jde o nástroj pro správu dat uložených v databázi. Název ovšem není dostatečně výstižný, poněvadž SQL není pouze dotazovací jazyk, ale umožňuje toho mnohem více, mimo jiné definovat struktury tabulek, naplňovat tabulky záznamy, definovat mezi nimi vztahy a řídit přístup k datům pomocí oprávnění. Většinou je ale používán k dotazování, u kterého umožňuje získat odpovědi na složité dotazy prakticky ihned. Jde o neprocedurální, standardizovaný a srozumitelný jazyk. Data chápe v podobě tabulek a tím usnadňuje pochopení vztahů uživatelům.

Jazyk SQL se skládá z několika částí. Některé jsou určeny pro návrháře a administrátory databázových systémů, jiné pro uživatele a programátory. První částí je DDL – **Data definition Language.** Jde o jazyk pro vytváření katalogů a schémat. Další částí je SDL – **Storage Definition Language.** Ta definuje, jak by se měly uchovávat tabulky. Třetí částí je VDL – **View Definition Language,** která se věnuje vytváření pohledů. Důležitou částí je pak DML – **Data Manipulation Language,** obsahující základní příkazy, jako je `INSERT, UPDATE, DELETE` a `SELECT`.

Příkazy pro správu databáze

Tato sada příkazů pracuje se strukturou databází. Umí vytvářet nové objekty, upravovat je a mazat.

CREATE – Tento příkaz slouží k vytváření databázových objektů. Můžeme ho využít pro vytvoření databáze, tabulky, pohledu nebo uložené procedury.

```
CREATE <DATABASE | TABLE | VIEW | PROCEDURE>
název_objektu
```

ALTER – Tento příkaz slouží ke změně databázových objektů. Můžeme ho využít pro změnu struktury databáze, tabulky, výběru dat v pohledu nebo kódu uložené procedury.

```
ALTER <DATABASE | TABLE | VIEW | PROCEDURE>
název_objektu
```

DROP – Tento příkaz slouží k odstranění databázových objektů. Můžeme ho využít pro odstranění databáze, tabulky, pohledu nebo uložené procedury.

```
DROP <DATABASE | TABLE | VIEW | PROCEDURE>
název_objektu
```

Příkazy pro správu dat

Tato sada příkazů slouží k vyhledávání a úpravám záznamů.

SELECT – Vrací množinu záznamů z jedné nebo více tabulek.

```
SELECT * FROM Kniha
```

INSERT – Přidává do tabulky nový záznam.

```
INSERT INTO <tabulka> (<sloupec>) VALUES (<hodnota>);
```

UPDATE – Upravuje záznamy v databázi.

```
UPDATE <tabulka> SET <název_sloupečku> = <hodnota>
WHERE <podmínka>
```

DELETE – Maže záznamy v databázi.

```
DELETE FROM <název_tabulky> WHERE <podmínka>
```

3.4.3 Datové typy

Zde si uvedeme nejrozšířenější Datové typy, na které můžeme v MySQL narazit.

INT nebo INTEGER

Rozsah hodnot od -2 147 483 648 do +2 147 483 647, bez znaménka až 4 294 967 295 (zabere 4 bajty).

FLOAT

Rozsah hodnot od -3.402823466E+38 do 3.402823466E+38.

DOUBLE

Rozsah hodnot od -1.7976931348623157E+308 do 1.7976931348623157E+308.

DATE

Datum ve formátu "rok-měsíc-den" respektive "RRRR-MM-DD" a v rozsahu 1000-01-01 až 9999-12-31.

DATETIME

Datum a čas v rozsahu 1000-01-01 00:00:00 až 9999-12-31 23:59:59 (formát je "RRRR-MM-DD HH:MM:SS").

VARCHAR(m)

Délka řetězce "m" může být v rozsahu 0-255.
Pokud je vložený řetězec kratší než nastavíme, chybějící znaky se nedoplňují (má tedy "plovoucí" velikost), ale navíc se ukládá informace o jeho délce.

4 NoSQL databáze

NoSQL databáze jsou tématem této práce, proto je jim věnována největší část. Nejdříve je třeba říct, co to vlastně NoSQL databáze jsou. Dále je kapitola věnována CAP teorému, na kterém je vysvětleno, že jedním z důvodů, proč existuje tolik NoSQL produktů, je to, že každý z nich může zajistit jen určité vlastnosti na úkor jiných a uživatel se pak musí rozhodnout, která kombinace vlastností je pro něj nejdůležitější a podle toho vybrat produkt. Dále se práce věnuje škálovatelnosti, tedy schopnosti navyšování výkonu databázového

serveru, ať již samotným vylepšováním serveru o výkonnější hardware nebo rozprostřením databáze na více serverů. Spolu se škálovatelností je třeba se taky zmínit o shardingu, tedy o technice, která rozděluje databáze na více částí, jež mohou fungovat samostatně a rychleji, než jeden celek. Poté následuje seznámení s nejznámějšími NoSQL produkty, kde jsou následně vybráni tři zástupci, se kterými je provedeno bližší seznámení.

4.1 Popis

Jak již zaznělo v úvodu, NoSQL databáze jsou ty, které nejdou relační cestou řízení dat, ale jinou. Nejsou primárně postavené na tabulkách a nevyužívají SQL pro práci s daty. Jejich doména je vysoká optimalizace pro vyhledávání, na úkor malé funkcionality, která se často omezuje na prosté ukládání dat. Tyto nedostatky, v porovnání s SQL, jsou kompenzovány škálovatelností a výkonem u určitých datových modelů.

NoSQL se hodí na ukládání velkého objemu dat, u kterých není třeba uchovávat vzájemné vztahy. Strukturovaná data jsou ovšem povolena.

U transakcí nemohou NoSQL databáze nabídnout plnou podporu ACID, nýbrž pouze Eventual consistency (Výsledná shoda). Jde o situaci, kdy díky zanedbání ACID získáme větší dostupnost a také lepší škálovatelnost. Tento přístup bez „silné konzistence“ je vhodný pro NoSQL, které ho taky často aplikují.

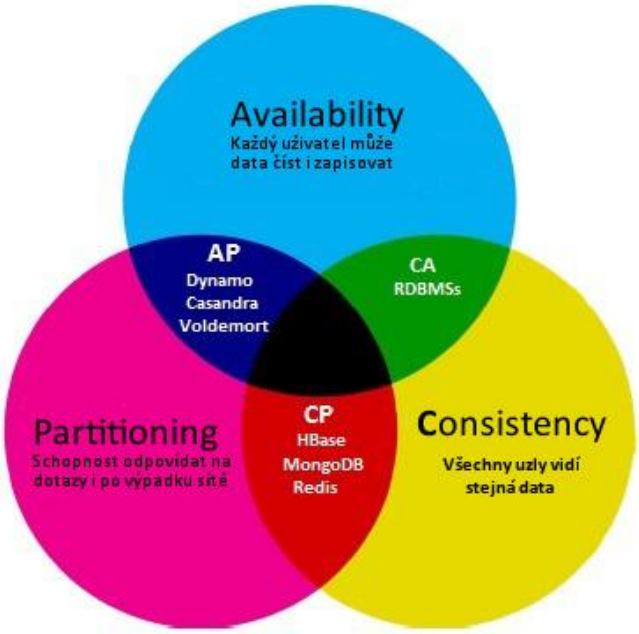
NoSQL má distribuovanou architekturu, která je odolná vůči chybám. Některá data bývají umístěna na několika serverech a tím selhání jednoho se dá tolerovat. Tyto databáze většinou škálují horizontálně a spravují velké objemy dat, u kterých je důležitější výkon než konzistence.

4.2 *CAP teorém*

CAP teorém, někdy nazývaný též jako Brewerův teorém, bývá spojovaný s distribuovanými systémy. Jedná se o vlastnosti, které mají zabezpečit chod. Obsahuje tři požadavky: Konzistenci (Consistency), Dostupnost (Availability) a toleranci k rozdělení (Partitioning tolerance). Zároveň ovšem také dodává, že není možné splnit všechny tři na 100%.

Brewer chápal pod zkratkou CAP ve spojení s NoSQL databázemi následující:

- Consistency znamená, že stejná data jsou k dispozici v určitém čase na všech uzlech distribuovaného systému.
- Availibility znamená, že každý klient, který odešle dotaz, dostane informaci o tom, zda byla operace úspěšná nebo ne
- Partition tolerance hovoří o tom, že pokud dojde k výpadku části sítě, musí být systém stále schopný odpovídat na dotazy.



Obrázek 3 CAP teorém [10]

Z obrázku je vidět, že je možné navrhnout distribuovaný systém, který splňuje maximálně 2 kritéria. Ve skutečnosti se k výsledku dá ještě přičíst část zbývající vlastnosti, ale to je vše. Není možné dosáhnout úplného splnění všech 3 vlastností.

Největší důraz se klade na 2 kombinace. V případě Dynama je větší důraz kladen na to, že každý klient dostane odpověď na svůj dotaz i v případě výpadku napájení. U HBase je hlavní konzistence dat a vrácení odpovědi v případě výpadku napájení. Dynamo i z větší části řeší konzistenci, protože ho využívá společnost Amazon a ta si nemůže dovolit nabídnout produkt více zájemcům, pokud ho má na skladu v nedostatečném množství.

4.3 Škálovatelnost

Objem dat umístěných na internetu se neustále navyšuje, a proto je třeba zajišťovat další prostory. Pokud bychom ukládali pouze data, která se nebudou číst, vystačili bychom si klidně i s pouhým rozšířením diskových kapacit na webových serverech. Poněvadž ale ukládat data, která se nebudou číst, nemá význam, zvyšuje se nám spolu s potřebou větších kapacit i odezva aplikací, které s těmito daty operují (ať už jde o čtení, zápis nebo aktualizaci), protože tyto aplikace musí daty procházet a organizovat je.

André Bondi z AT&T Labs uvádí, že *„Škálovatelnost je žádoucí vlastnost sítě, systému, nebo procesu. Tato koncepce implikuje schopnost systému pojmout rostoucí počet prvků nebo objektů ke zpracování rostoucích objemů práce řádně a / nebo být nachystaný k rozšíření.“* a že škálovatelnost je „*vlastnost nejen fungovat dobře po rozšíření, ale hlavně této skutečnosti plně využít.“*[11]

Navýšení diskové kapacity u relačních i nerelačních systémů nebývá většinou dostačující. Navyšuje se proto výkon celého systému, čehož lze dosáhnout dvěma metodami:

1. **Vertikální škálování (Vertical scaling)**

Navýšení výkonu fyzického serveru se provádí například přidáním paměti a výměnou procesoru za výkonnější. Tento způsob škálování má ale svá omezení a těmi jsou kompatibilita a především finanční náročnost, protože u této metody je třeba mít vysoce výkonný a tudíž drahý hardware a často se ani nevyhneme kompletní výměně serveru.

2. **Horizontální škálování (Horizontal scaling)**

Horizontální škálování naproti tomu jde jinou cestou. Nesnaží se vylepšovat jeden server, ale místo toho rozprostře svá data na více serverů, které spolu vzájemně spolupracují. I když se to na první pohled nemusí zdát, tak tento způsob je finančně méně zatěžující, protože stačí levnější stroje. Nesporná výhoda je taky fakt, že takto můžeme zvyšovat kapacitu teoreticky do nekonečna.

4.4 Sharding

Shard je část databáze, která vznikne, pokud budeme chtít držet určité záznamy zvlášť, místo toho, aby byly všechny pospolu. Každý tento shard může být pak umístěn na odlišném databázovém serveru nebo klidně i geografickém místě.

Tato technika má řadu výhod. Poněvadž je databáze rozdělena a distribuována na více serverech, počet záznamů v každé databázi je menší. To zmenšuje velikost indexace a tím i dobu vyhledávání. Dále pokud by vypadl jeden ze shardů (databázových strojů), tak se stále ke zbytku záznamů dostaneme, protože jsou umístěné zvlášť. Navíc můžeme jeden dotaz spustit paralelně na několika shardech a potom pomocí sjednocení získat jeden výsledek.[1]

Nejčastěji se jako příklad shardingu uvádí situace, kdy budeme mít jednu velkou databázi uživatelů. Pro rychlejší práci je jí ale třeba rozdělit. Mohli bychom tedy rozdělit uživatele geograficky podle toho, kde žijí. Další příklad

rozložení databáze je WordPress Multisite. Ten hostuje stovky tisíc blogů. Možné řešení by bylo rozdělit stovku nejvíce zatížených blogů mezi 2 shardy a zbytek dát na shard třetí.

4.5 Hlavní představitelé

Mezi NoSQL databáze patří mnoho zástupců. Ti se mohou rozdělovat do kategorií podle toho, jaký datový model splňují. Mezi nejznámější datové modely patří sloupcově orientované databáze, databáze s klíčem a hodnotou, dokumentové databáze a objektové databáze. Modelů existuje samozřejmě více, např. grafové databáze a XML databáze, my se ale budeme věnovat pouze těm nejznámějším.

4.5.1 Sloupcově orientované databáze

Sloupcově orientované databáze ukládají data ve sloupcové formě a ne v řádkové, jak to známe z relačních databází. Nejlépe pochopitelné to bude na následující ukázce.

Mějme dvourozměrnou tabulku, která uchovává informace o produktech.

id	kategorie	kód	cena
1	5	118741	12241
2	8	168123	2420
3	2	247351	1748

Aby systém mohl s tabulkou pracovat, jde třeba ji převést na jednorozměrnou podobu. Řádkově orientované databáze spojí všechny hodnoty v řádce dohromady.

```
1,5,118741,12241;
2,8,168123,2420;
3,2,247351,1748;
```

Sloupcově orientované databáze ale spojí všechny hodnoty ve sloupcích dohromady.

```
1,2,3;
5,8,2;
118741,168123,247351;
12241,2420,1748;
```

Sloupcově orientované databáze jsou výkonnější, když potřebujeme provést nějakou agregační funkci na mnoha řádcích, ale na omezeném počtu sloupců. Výkonnost se také projeví, pokud chceme změnit hodnoty sloupců u všech řádek najednou.

Mezi nejznámější zástupce patří:[1]

HBase

- **Oficiální stránky –** http://hbase.apache.org.
- **Historie –** Vytvořené firmou Powerset v roce 2007. Projekt se později stal součástí Apache Software Foundation.
- **Jazyk –** Napsáno v jazyce Java.
- **Použití –** Facebook a další.

4.5.2 Databáze s klíčem a hodnotou

Asociativní pole nebo HashMapa jsou nejjednodušší datové struktury, které dokážou uchovávat klíč s hodnotou. Klíčem je zde jedinečná hodnota, která může být jednoduše využita pro vyhledání dat.

Databáze s klíčem a hodnotou mohou být různých typů. Některé z nich ukládají data v paměti a jiné umožňují zaznamenání dat na disk.

Nejznámějšími databázemi na systému klíč-hodnota jsou:[1]

Redis

- **Oficiální stránky –** http://redis.io/
- **Historie –** Projekt byl zahájen v roce 2009 jako nezávislý projekt. Sponzorem Redisu se stal VMware.
- **Přístupové metody –** K Redisu lze přistoupit přes jeho příkazovou řádku nebo přes knihovny k jazykům jako jsou Java, C, C++ a další.
- **Jazyk –** Napsáno v jazyce C.

Voldemort

- **Oficiální stránky –** http://project-voldemort.com/
- **Historie –** Vytvořené LinkedIn v roce 2008.
- **Jazyk –** Napsáno v jazyce Java.
- **Použití –** Využití u LinkedIn.

Cassandra

- **Oficiální stránky –** http://cassandra.apache.org/
- **Historie –** Vyvinuto Facebookem. V roce 2008 se z projektu stal open-source. Apache Cassandra byla poté darována společnosti Apache Software Foundation.
- **Přístupové metody –** Ke Cassandře lze přistoupit přes příkazovou řádku nebo přes knihovny k jazykům jako jsou Java, PHP, .NET.
- **Jazyk –** Napsáno v jazyce Java.
- **Použití –** Využití u Facebooku, Twitteru a dalších.

4.5.3 Dokumentové databáze

Dokumentové databáze nejsou systémy pro správu dokumentů. Slovo „dokument“ v dokumentových databázích míní volně strukturovanou sadu

klíčů-hodnot v dokumentech. Základním prvkem je zde dokument, do kterého se ukládají data a je označen jedinečným identifikátorem a množinou klíčů-hodnot. Pro ukládání dat se využívá většinou JSON.

Mezi dokumentovými databázemi jsou nejznámější:[1]

MongoDB

- **Oficiální stránky –** www.mongodb.org.
- **Historie –** Vytvořené společností 10gen, která jej vyvíjí jako open-source.
- **Jazyk –** Napsáno v jazyce C++.
- **Přístupové metody –** Přes JavaScriptové rozhraní, případně využití rozsáhlé podpory jazyků, jako jsou C, C#, C++, PHP, atd.
- **Použití –** FourSquare a další.

CouchDB

- **Oficiální stránky –** http://couchdb.apache.org a www.couchbase.com.
- **Historie –** V roce 2005 ho začal psát Damien Katz, který se ale po dvou letech rozhodl ho uvolnit jako open-source a v roce 2008 byl projekt začleněn pod Apache Software Foundation.
- **Jazyk –** Napsáno v jazyce Erlang.
- **Použití –** BBC, Apple, CERN a další.

4.5.4 Objektové databáze

V těchto databázích jsou místo tabulek uloženy přímo objekty, včetně svých vlastností a místo řádků tu jsou samotné instance objektů. Každý objekt je identifikován jedinečným OID, což je ukazatel do virtuální paměti. Není tedy třeba vytvářet žádné primární klíče. Objektové databáze také přejímají některé

vlastnosti z objektového přístupu, jako je dědičnost, zapouzdření a polymorfismus.

Asi nejznámější objektovou databází je:[1]

Db4o

- **Oficiální stránky –** www.db4o.com.
- **Historie –** Vývoj zahájen v roce 2000, oficiálně vydaný v roce 2004. V roce 2008 projekt koupen společností Versant corporation.
- **Jazyk –** Napsán v jazyce Java a C#.

4.6 HBase

HBase je první z vybraných NoSQL databází, kterým se práce věnuje blíže. Zaměřuje se na popis databáze, který se zabývá architekturou a síťovou topologií. Následně je ukázáno, jak se s databází pracuje, konkrétně jde o spuštění serveru a připojení k němu. Na příkladech je pak vidět, jak se s daty pracuje a jaké datové typy můžeme použít.

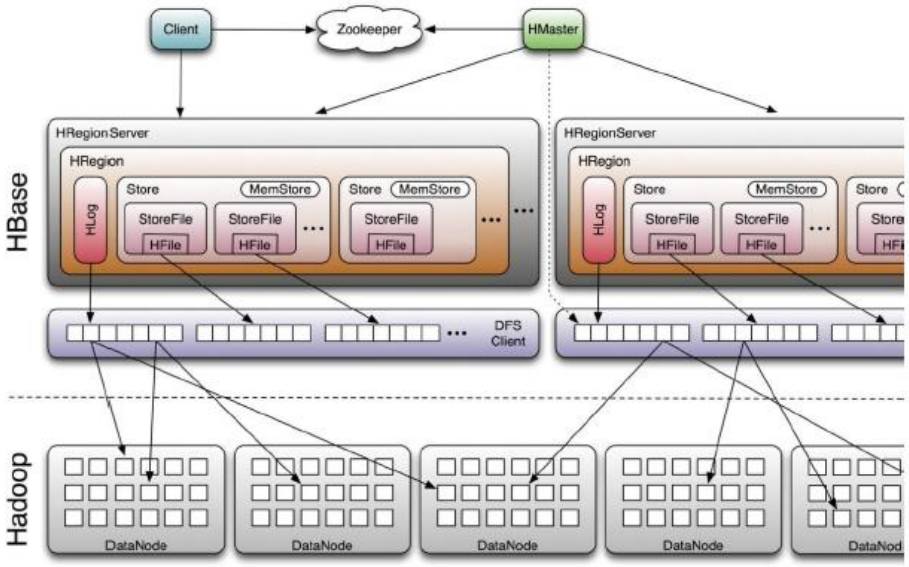
4.6.1 Popis

HBase byl vyvinut jako součást takzvaného Apache Hadoop projektu. Jde o open-source framework, který je určen pro zpracování velkého množství nestrukturovaných a distribuovaných dat v řádek petabytů a exabytů. Obsahuje také vlastní distribuovaný souborový systém HDFS (Hadoop Distributed File System).

Architektura HBase obsahuje více částí než jen HBase samotný. Především jde o centralizovanou a distribuovanou službu, která slouží k synchronizaci a konfiguraci.

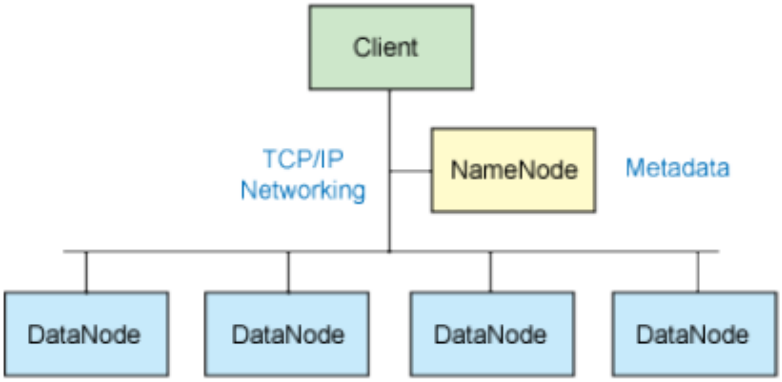
U HBase má každý prvek svou roli v síťové topologii. Pokud chceme nasadit HBase a využít distribuovaného ukládání souborů, můžeme nasadit i Apache

Hadoop. Základním kamenem, který se stará o požadavky klientů, je zde tzv. Region Server, který spravuje jednotlivé regiony (tabulka má více regionů). O spolupráci mezi jednotlivými Region Servery se stará Master Server, který běží nad nimi. Nad tím vším běží ještě Zookeeper, který udržuje celý cluster pohromadě. Na obrázku 4 je vidět zjednodušená architektura HBase.



Obrázek 4 HBase topologie[11]

Ukládání dat zajišťuje Hadoop Cluster, ke kterému je připojený každý z Region Serverů. Cluster obsahuje jeden NameNod a několik DataNodes. Na obrázku 6 je vidět síťová architektura Hadoop Clusteru.



Obrázek 5 Hadoop Cluster[11]

Při pokusu o vložení dat do clusteru naváže Region Server (Client) komunikaci s NameNode a sdělí mu, že chce uložit data. Ten mu sdělí adresu DataNode a číslo bloku, pod kterým bude data ukládat. Současně sdělí DataNode, že se do něj bude ukládat. Client pak může rovnou zaslat očekávaná data DataNode. Po uložení dat se ještě NameNode postará o vytvoření kopií uložených dat na jiné DataNody (buď ve stejném racku, nebo i mimo něj), aby v situaci, kdy dojde k výpadku jednoho z DataNode, byly data stále dostupná.[11]

4.6.2 Spuštění a připojení k serveru

Abychom mohli začít pracovat s HBase, musíme nejdříve spustit HBase server a připojit se k němu pomocí HBase shellu. Pokud jste si stáhli a extrahovali poslední verzi HBase, spusťte server pomocí `bin/start-hbase.sh`

Jakmile bude server běžet, můžete se k němu připojit pomocí `bin/hbase shell`

Po připojení můžeme vytvořit kolekci *knihy* s dvěma column-families, *kniha* a *obálka*.

```
hbase(main):001:0> CREATE 'knihy', 'kniha', 'obalka'
```

4.6.3 Správa dat

Správu dat můžeme provádět buď pomocí Hbase shellu nebo pomocí klienta. Ukážeme si obě techniky, přičemž klienta budeme mít napsaného v jazyce java.

Správa dat pomocí HBase shellu

Pro ukázku práce s databází můžeme využít příklad s knihami. Řekněme, že budeme chtít uchovávat:

- Název knihy
- Autora knihy

- Popis knihy
- Přední obálku knihy
- Zadní obálku knihy

Abychom mohli tyto informace uchovávat, vytvoříme si kolekci nazvanou *Knihy* a budeme data ukládat do dvou kategorií, *kniha* a *obálka.* Možný vstup ve formátu JSON by pak vypadal následovně:

```
{
     "kniha": {
          "nazev": "Robinson Crusoe",
          "autor": "D. Defoe",
          "popis": "Dobrodružný román",
     },
     "obalka": {
     "predni": front.png,
     "zadni": back.png,
     },
}
```

Nebo by to mohlo vypadat následovně:

```
{
     "kniha": {
          "nazev": "Robinson Crusoe",
          "autor": "D. Defoe",
          "popis": "Dobrodružný román",
     },
     "obalka": {
     "logo": logo.png,
     "barva": #EE9A49,
     },
}
```

Pokud se podíváte na předchozí 2 příklady, zjistíte, že oba sdílejí kategorie *kniha* a *obálka,* ale nemusí mít stejná pole. V HBase se tomu říká, že sdílejí stejné Column-families (kniha a obálka), ale mají rozdílné sloupce. Ve skutečnosti má *obálka* 4 sloupce: *přední, zadní, logo, barva* a v některých případech tyto sloupce nemají hodnotu (jsou null). Pokud bychom něco

takového chtěli provést v relačních databázích, museli bychom vytvořit všechny 4 sloupce u každého záznamu a vyplnit null na patřičných místech. Sloupcové databáze, a tudíž i HBase, nemusí ukládat hodnoty, pokud jsou null.

Přidání záznamů probíhá následujícím příkazem:

```
hbase(main):001:0> PUT 'knihy', 'kniha1',
'kniha:popis',
hbase(main):002:0* 'Dobrodruzny roman'
hbase(main):003:0> PUT 'knihy', 'kniha1',
'kniha:autor', 'D. Defoe'
hbase(main):004:0> PUT 'knihy', 'kniha1', 'kniha:
nazev', 'Robinson Crusoe'
hbase(main):005:0> PUT 'knihy', 'kniha1',
'obalka:predni', 'predni.png'
hbase(main):006:0> PUT 'knihy', 'kniha1',
'obalka:zadni', 'zadni.png'
```

Vytvořili jsme si záznam s jednou knihou. Pokud bychom udělali chybu a chtěli ji opravit novým zadáním, např.

```
hbase(main):003:0> PUT 'knihy', 'kniha1',
'kniha:autor', 'W. Shakespeare'
hbase(main):004:0> PUT 'knihy', 'kniha1',
'kniha:autor', 'D. Defoe'
```

tak ve světě relačních databází by došlo k updatu a záznam se upravil. V HBase jsou ovšem data neměnná a proto opakované vložení nevede ke změně existujícího záznamu, ale k vytvoření nové verze data setu. Tento přístup má 2 výhody. Vyhneme se problémům s atomicitou u aktualizace záznamů a vzniká nám zabudovaný verzovací systém v databázi.

Nyní, když máme záznam v databázi, můžeme se na něj dotázat. Pokud jsme se mezitím odhlásili, je třeba se opět přihlásit pomocí `bin/hbase shell`.

Když budeme chtít získat data vztahující se ke `kniha1`, dotážeme se následovně:

```
hbase(main):07:0> GET 'knihy', 'kniha1'
```

Tento dotaz nám vrátí odpovídající informace:

```
COLUMN                    CELL
obalka:predni             timestamp=1302139666802,
value=predni.png
obalka:zadni              timestamp=1302139638880,
value=zadni.png
kniha:autor               timestamp=1302139570361,
value=D. Defoe
kniha:popis               timestamp=1302139604738,
value=Dobrodruzny roman
kniha:nazev               timestamp=1302139434638,
value=Robinson Crusoe
```

Pokud by nás zajímala pouze jedna informace, například autor knížky, dotaz trochu modifikujeme:

```
hbase(main):08:0> GET 'knihy', 'kniha1', {
COLUMN=>'kniha:autor' }

COLUMN                    CELL
kniha:autor               timestamp=1302139434638,
value=D. Defoe
```

Tento dotaz nám vrátí poslední zadanou informaci o autorovi dané knihy. Pokud jsme v situaci, kdy jsme provedli opakované vložení autora (viz. výše) a chtěli bychom vidět obě verze, dotaz bude vypadat následovně:

```
hbase(main):027:0> GET 'knihy', 'kniha1', {
COLUMN=>'kniha:autor', VERSIONS=>2 }

COLUMN                    CELL
kniha:autor               timestamp=1302139851203,
value=W. Shakespeare
kniha:autor               timestamp=1302139819904,
value=D. Defoe
```

Správa dat pomocí jazyka Java

Abychom se mohli dotazovat do databáze HBase, musíme si stáhnout následující JAR soubory a potom je přidat do importů v projektu.

- commons-logging-x.x.x.jar
- hadoop-core-x.x-append-rx.jar
- hbase-x.x.x.jar
- log4j-x.x.x.jar

Pokud bychom chtěli znát autora a název knížky u `kniha1`, zdrojový kód v javě by mohl vypadat následovně:

```
8
9  public class HBase
10 {
11     public static Map ziskejKnihu(String knihaID) throws IOException
12     {
13         HTable table = new HTable(new HBaseConfiguration(), "knihy");
14         Map kniha = new HashMap();
15         RowResult vysledek = table.getRow(knihaID);
16         for (byte[] sloupec : vysledek.keySet())
17         {
18             kniha.put(new String(sloupec),
19             new String(vysledek.get(sloupec).getValue()));
20         }
21         return kniha;
22     }
23     public static void main(String[] args) throws IOException
24     {
25         Map knihy = HBase.ziskejKnihu("kniha1");
26         System.out.println(knihy.get("kniha:nazev"));
27         System.out.println(knihy.get("kniha:autor"));
28     }
29 }
```

Obrázek 6 Dotazování do HBase v jazyce Java

Vytvoříme si metodu `ziskejKnihu`, která na základě daného parametru získá všechny jeho hodnoty sloupců. V metodě `main` si již pouze vybereme, které sloupce nás zajímají.

4.6.4 Datové typy

HBase podporuje interface, který umožňuje uložit cokoliv, co může být konvertováno na pole bytů. Vstupem tedy mohou být řetězce znaků, čísla, komplexní objekty nebo dokonce obrázky, pokud se dají převést do pole bytů.

Existují praktické limity vztahující se na velikosti ukládaných hodnot (například s ukládáním 10-50MB objektů v HBase byste pravděpodobně moc neuspěli).

4.7 MongoDB

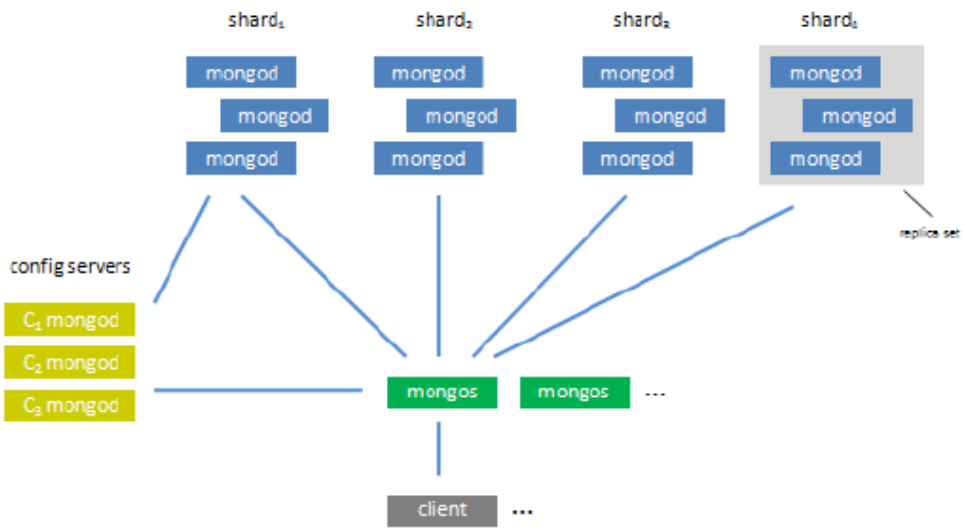
U MongoDB se nejdříve budeme věnovat jeho popisu, kde se opět seznámíme s tím, jak jsou v něm data uložená a jak funguje v síti. Pro práci s databází je také si nejdříve ukázat, jak vůbec server spustit a připojit se k němu. Nakonec následuje kapitola o datových typech, které lze v MongoDB využít.

4.7.1 Popis

MongoDB je dokumentová databáze, kde jsou dokumenty spojovány do kolekcí. Kolekce si můžeme představit jako relační tabulky s tím rozdílem, že kolekce netrpí povinností dodržovat schéma databáze, které je u relačních tabulek podmínkou. Libovolné dokumenty mohou být spojeny dohromady v jednu kolekci, měly by ovšem být podobné, aby mohlo být dosaženo efektivního indexování.

Každý dokument je uložen ve formátu BSON. Jde o binárně kódovanou reprezentaci JSON formátu, kde jsou vnořené sady klíčů a hodnot. BSON je nadmnožinou JSON a podporuje navíc například regulární výrazy, binární data a datum. Každý dokument obsahuje jedinečný identifikátor, který můžeme zadat ručně. Pokud tak neučiníme, MongoDB ho vygeneruje sám.

Komponenty a topologie clusteru MongaDB je vidět na Obrázku 7. Uložená data zde drží tzv. *mongod*, který se také spouští v případě single-node clusteru, tedy v situaci, kdy se všechny data ukládají na lokální disk. Instancí mongod může být spuštěno teoreticky neomezené množství a spojují se do tzv. shardů, kde všechny mongod instance v jednom shardu obsahují tytéž data. Vedle shardů můžeme vidět několik instancí *config serverů*, které udržují informace o tom, jaká data udržují shardy a také informace o clusteru samotném.



Obrázek 7 MongoDB cluster[11]

Jako prostředník zde slouží Mongo Router, neboli *mongos.* Ten, když obdrží požadavek od nějaké klientské aplikace, tak se nejdříve zeptá *config serveru,* kde jsou požadovaná data uložena. Když dostane odpověď, provede další dotaz už na patřičný *mongod,* který obsahuje hledaná data a následně přepošle data klientské aplikaci. Pokud by data byly rozesety na více shardech, tak nejdříve provede spojení odpovědí z *mongod*. Rozdělení záznamů v clusteru můžeme vidět na Obrázku 8.

collection	minkey	maxkey	location
users	{ name : 'Miller' }	{ name : 'Nessman' }	shard2
users	{ name : 'Nessman' }	{ name : 'Ogden' }	shard4
...			

Obrázek 8 Záznam v MongoDB clusteru[11]

4.7.2 Spuštění a připojení k serveru

Abychom mohli začít spravovat MongoDB databázi, musíme se k ní nejdříve připojit. Spuštění MongoDB serveru probíhá přes program *mongod*, který se nachází v adresáři *bin,* umístěném v distribuci.

Nejjednodušší způsob, jak se připojit k MongoDB serveru, je využití JavaScriptového shellu, dodávaného jako součást distribuce. Stačí spustit přes příkazovou řádku program *mongo,* který se nachází taktéž v adresáři *bin.*

Při spuštění MongoDB serveru byste v terminálu měli vidět podobný výpis:

```
Feb 22 15:30:03 [initandlisten] MongoDB starting :
pid=3645 port=27017 dbpath=/data/db/ 64-bit
host=striker-MS-7532

Feb 22 15:30:03 [initandlisten] db version v2.2.3,
pdfile version 4.5

Feb 22 15:30:03 [initandlisten] build info: Linux ip-
10-2-29-40 2.6.21.7-2.ec2.v1.2.fc8xen #1 SMP Fri Nov
20 17:48:28 EST 2009 x86_64 BOOST_LIB_VERSION=1_49

Feb 22 15:30:03 [websvr] admin web console waiting
for connections on port 28017

Feb 22 15:30:03 [initandlisten] waiting for
connections on port 27017
```

Výpis nám toho sděluje poměrně dost. Je zde ID procesu, port, na kterém server naslouchá příchozím požadavkům, místo, kam se ukládá databáze, verze distribuce MongoDB a na jakém systému distribuci provozujeme.

Když nám už server běží, můžeme se k němu připojit pomocí programu *mongo.* Ten spustíme v novém terminálu. Měl by se nám objevit podobný výpis:

```
MongoDB shell version: 2.2.3
connecting to: test
Welcome to the MongoDB shell.
>
```

Výpis nám říká, jakou verzi MongoDB shellu používáme a k jaké databázi se připojujeme (standardně je to databáze *test*, která je k dispozici na localhostu). Dále už program čeká na naše příkazy. Pokud napíšeme příkaz `help`, objeví se nám základní sada příkazů a další `help`, který obsahuje příkazy na konkrétní oblasti.

db.help()	help s metodami k databázi
db.mycoll.help()	help s metodami ke kolekcím
help connect	help s připojením k databázi
help admin	help pro správu databáze
help misc	mix věcí, které je dobré znát
show dbs	ukáže názvy databází
show collections	ukáže kolekce v aktuální databázi
use <db_name>	nastaví aktuálně používanou databázi
exit	ukončení mongoDB shellu

Tabulka 2 Zkrácená verze výsledku příkazu help

4.7.3 Správa dat

Správa dat je opět možná dvěma způsoby. Buď můžeme využít shell, který se dodává spolu se serverem v balíčku nebo můžeme využít některý z klientů, napsaných v různých programovacích jazycích. V tomto případě jde o jazyk java.

Správa dat pomocí MongoDB shellu

Abychom mohli začít spravovat data, vytvoříme si nejdříve databázi, pojmenovanou *knihovna*. Po vytvoření uložíme páry *jmen autorů* a *názvů knížek* do kolekce *autoři*. Tento proces se provede ve 3 krocích:

1. Změníme databázi na *knihovnu*.
2. Vytvoříme si data, která budeme chtít uchovávat.
3. Vytvořená data uložíme do kolekce.

Vykonání těchto kroků je jednoduché. Následující řádky zapíšeme v MongoDB shellu:

```
USE knihovna
a={jmeno: "Alois Jirasek", kniha: "Stare povesti
ceske"};
b={jmeno: "Agatha Christie", kniha: "Deset malych
cernousku"};
c={jmeno: "Miloslav Ponkrac", kniha: "PHP a MySQL"};
d={jmeno: "Karolina Svetla", kniha: "Kriz u potoka"};
db.autori.save(a);
db.autori.save(b);
db.autori.save(c);
db.autori.save(d);
```

Logickou otázkou by zde mohlo být, jak je možné, že mohu používat databázi *knihovna* a kolekci *autoři*, pokud jsem si je nikde předtím nevytvořil. Odpověď je jednoduchá. V MongoDB se databáze a kolekce vytvoří při ukládání dat do nich. V našem případě tedy dojde k vytvoření databáze a kolekce, když ukládáme záznam s Aloisem Jiráskem.

Pokud se budeme chtít dotázat na všechny uložené položky v kolekci *autoři*, uděláme to pomocí `db.autori.find()`. Tento příkaz nám vrátí:

```
{ "_id" : ObjectId("51278653f1de2f0eefbe5109"),
"jmeno" : "Alois Jirasek", "kniha" : "Stare povesti
ceske" }
{ "_id" : ObjectId("5127865cf1de2f0eefbe510a"),
"jmeno" : "Agatha Christie", "kniha" : "Deset malych
cernousku" }
{ "_id" : ObjectId("51278664f1de2f0eefbe510b"),
"jmeno" : "Miloslav Ponkrac", "kniha" : "PHP a MySQL"
}
{ "_id" : ObjectId("5127866ef1de2f0eefbe510c"),
"jmeno" : "Karolina Svetla", "kniha" : "Kriz u
potoka" }
```

Kromě *jména* a *knihy* zde máme i *_id*. Jde o jedinečný identifikátor pro každý záznam nebo dokument.

Metoda `find()` bez parametru nám vrací všechny uložené záznamy v kolekci. Pokud bychom chtěli výpis více filtrovat, můžeme jako parametr metodě `find()` předat to, co hledáme. Aby byl ale výpis zajímavější, přidáme si ještě další 2 záznamy:

```
a={jmeno: "A. Christie", kniha: "Deset malych
cernousku"};
b={jmeno: "Agatha Christie", kniha: "Potize v zalivu
Pollensa"};
db.autori.save(a);
db.autori.save(b);
```

Pokud bychom teď chtěli najít, kdo napsal knihu „Deset malých černoušků“, Dotaz s výsledkem by vypadal takto:

```
db.autori.find({kniha: "Deset malych cernousku"});
{ "_id" : ObjectId("5127865cf1de2f0eefbe510a"),
"jmeno" : "Agatha Christie", "kniha" : "Deset malych
cernousku" }
{ "_id" : ObjectId("51278872f1de2f0eefbe510f"),
"jmeno" : "A. Christie", "kniha" : "Deset malych
cernousku" }
```

A pokud bychom chtěli vědět, jaké knihy napsala „Agatha Christie“, dotaz by byl obdobný:

```
db.autori.find({jmeno:"Agatha Christie"});
{ "_id" : ObjectId("5127865cf1de2f0eefbe510a"),
"jmeno" : "Agatha Christie", "kniha" : "Deset malych
cernousku" }
{ "_id" : ObjectId("512787f5f1de2f0eefbe510e"),
"jmeno" : "Agatha Christie", "kniha" : "Potize v
zalivu Pollensa" }
```

MongoDB podporuje řadu pokročilých vyhledávacích mechanismů, včetně zástupných znaků a regulárních výrazů, takže nemusí zůstat pouze u jednoduchých filtrů.

Správa dat pomocí jazyka Java

Abychom mohli propojit MongoDB s Javou, potřebujeme si stáhnout knihovny, které následně připojíme k projektu. Můžeme je nalézt na http://github.com/mongodb.

Následně spustíme MongoDB server pomocí programu *mongod*, který se nachází v adresáři *bin* v distribuci MongoDB, aby se měla naše aplikace k čemu přihlásit.

```
7
8  public class MongoDB
9  {
10     Mongo m = null;
11     DB db;
12     public void connect() {
13         try
14             m = new Mongo("localhost", 27017 );
15         catch (Exception e)
16             e.printStackTrace();
17     }
18     public void vypisZaznamy() {
19         if(m!=null) {
20             db = m.getDB("knihovna");
21             DBCollection collection = db.getCollection("autori");
22             DBCursor cur = collection.find();
23             while(cur.hasNext())
24                 System.out.println(cur.next());
25         }
26         else
27             System.out.println("Nepripojeno");
28     }
29     public static void main(String[] args) {
30         MongoDB conn = new MongoDB();
31         conn.connect();
32         conn.vypisZaznamy();
33     }
34 }
```

Obrázek 9 Připojení databáze a vypsání záznamů

4.7.4 Datové typy

MongoDB může uchovávat libovolné kolekce dat za předpokladu, že je lze vyjádřit za použití JSON. Podporuje všechny jeho datové typy, konkrétně

string, integer, boolean, double, null, array a object. Díky podpoře BSON si MongoDB může mezi své datové typy ještě přidat date, object id, binární data a regulární výrazy.

4.8 Redis

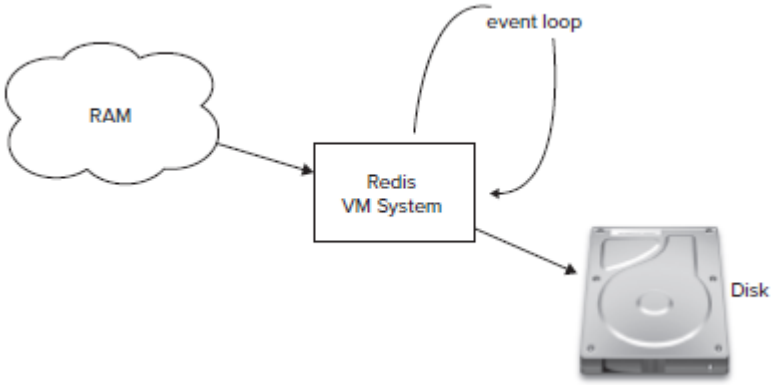
Redis je poslední z vybraných NoSQL produktů, kterými se práce blíže zabývá. Jako u předchozích se nejdříve práce zabývá tím, jak Redis data uchovává a jak pracuje v síti. Spolu se spuštěním serveru a připojením k němu, se práce zmiňuje také o tom, jak pracovat s daty, která jsou v něm uložená. Redis umožňuje uchovávat celkem 5 typů hodnot, proto se jim poslední část věnuje. Kapitola 6 navazuje na problematiku Redisu a věnuje se ji ve větší hloubce.

4.8.1 Popis

Všechno v Redisu je v podstatě reprezentováno jako řetězec. Dokonce i kolekce, jakou je list, set, sorted set a mapy, jsou složeny z řetězců. Redis definuje speciální strukturu, kterou nazývá *Simple dynamic string (SDS)*. Tato struktura se skládá ze tří částí:

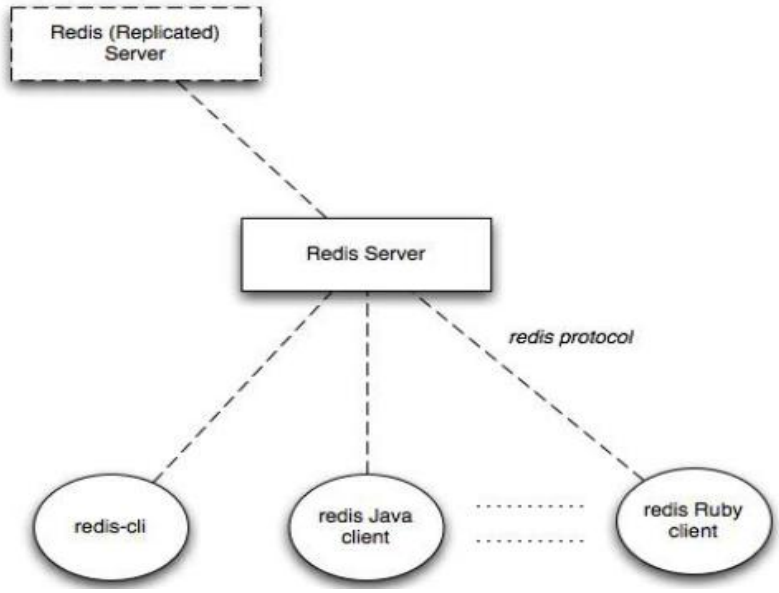
1. **buff –** Pole znaků, které uchovává řetězec
2. **len –** Číslo typu long, které uchovává délku pole buff
3. **free –** Počet bytů, které jsou ještě k dispozici

Redis uchovává svá data v paměti, ukládání na disk je na vyžádání. Na rozdíl od MongoDB nevyužívá mapování paměti souborů, ale má svůj vlastní virtuální paměťový subsystém. Když je uložená hodnota na disk, ukazatel na toto místo na disku je uložen spolu s klíčem.



Obrázek 10 Redis architektura[1]

Redis není určen pro ukládání velkého množství dat a proto také neumožňuje rozdělení svých dat mezi několik nodů v clusteru. Využívá tzv. master-slave architektury, kdy je v clusteru jeden master server a na něj se připojují slave servery. Na ty se ale mohou připojit další slave servery, takže vůči nim se chová jako master server. Jednou z vlastností Redisu je, že replikace dat probíhá pouze z master serveru na slave server. Proto také musí být všechny aplikace připojeny k master serveru, pokud chtějí měnit data.



Obrázek 11 Síťová topologie Redisu[11]

4.8.2 Spuštění a připojení k serveru

Pro práci s Redisem je třeba mít běžící Redis server a klienta, který bude se serverem komunikovat. Spuštění serveru můžeme provést pomocí příkazu `redis-server`. Měli bychom vidět podobný výpis:

```
striker@striker-MS-7532:~/redis-2.6.10/src# redis-
server[8597] 23 Feb 17:59:47 # Warning: no config
file specified, using the default config. In order to
specify a config file use 'redis-server
/path/to/redis.conf'
[8597] 23 Feb 17:59:47 * Server started, Redis
version 2.4.15
[8597] 23 Feb 17:59:47 * The server is now ready to
accept connections on port 6379
[8597] 23 Feb 17:59:47 - 0 clients connected (0
slaves), 790840 bytes in use
```

Nyní již můžeme spustit klienta pomocí příkazu `redis-cli`.

```
striker@striker-MS-7532:~/redis-2.6.10/src$ redis-cli
redis 127.0.0.1:6379>
```

Server standardně naslouchá na portu 6379. Abychom si ověřili, že jsme skutečně připojení, můžeme zkusit zadat příkaz `SET klic "hodnota"`. Pokud uvidíme text `OK`, jako odpověď na náš příkaz, pak jsme připojeni. Více k této problematice v kapitole 6.

4.8.3 Správa dat

I Redis se dodává se shellem, který je součástí balíčku a umožňuje se připojit k serveru a pracovat s ním. Zároveň má taky bohatou podporu klientů, ze kterých si ukážeme ten, který je napsaný v javě a práci s ním.

Správa dat pomocí Redis shellu

Správu dat provedeme opět pomocí ukázky. Řekněme, že chceme uchovávat články. Redis podporuje linked list, set, sorted set, hashe a řetězce. Pro uchovávání článků využijeme v našem případě set, protože nás nezajímá,

v jakém pořadí články budou, a pojmenujeme si ho *články*. U každého článku budeme evidovat *id, titulek, text, autora* a *tag*, který bude obsahovat *id* a *název*. Vytvoření *id, titulku, textu* a *autora* proběhne následovně:

```
>INCR next.clanky.id
(integer) 1
>SADD clanky:1:titulek “Nehoda v Praze“
(integer) 1
> SADD clanky:1:autor“Petr Laciny“
> SADD clanky:1:text “Dnes rano v 8 hodin se stala nehoda“
```

Redis nabízí mnoho příkazů, které jsou všechny uvedeny na http://redis.io/commands. Příkaz INCR generuje číselnou posloupnost postupným navyšováním o jedničku. Nezačíná se tu ale od nuly, nýbrž od čísla jedna. Další dva příkazy nám vytváří *titulek a autora* u článku, který je reprezentován svým *id*. Nyní si přidejme ke článku několik *tagů*.

```
>SADD clanky:1:tag 1
>SADD clanky:1:tag 2
>SADD clanky:1:tag 3
>SADD clanky:1:tag 4
```

Právě jsme přidali k článku s *id* 1 *id* několika tagů. Tagy ale zatím nebyly nijak definovány. To můžeme udělat následovně:

```
>SADD tag:1:nazev “Z domova“
>SADD tag:2:nazev “Nehody“
>SADD tag:3:nazev “Automobil“
>SADD tag:4:nazev “Spatne pocasi“
```

Nyní je třeba zajistit vzájemné propojení, aby článek měl odpovídající tagy. První článek má všechny tagy, takže přidáme každý tag následovně:

```
>SADD tag:1:clanky 1
>SADD tag:2:clanky 1
>SADD tag:3:clanky 1
>SADD tag:4:clanky 1
```

První záznam je tedy kompletní. Pro potřeby vyhledávání si ale ještě vytvořme jeden.

```
>INCR next.clanky.id
>SADD clanky:2:titulek “Mrazy poleví“
>SADD clanky:2:autor “Michal Churavý“
>SADD clanky:2:text “Během příštího týdne se oteplí“
>SADD clanky:2:tag 1
>SADD clanky:2:tag 4
>SADD tag:1:clanky 2
>SADD tag:4:clanky 2
```

Pokud nás bude zajímat název článku číslo 2 a jeho tagy, dotaz můžeme zformulovat následovně:

```
>SMEMBERS clanky:2:titulek
1. “Mrazy poleví“
>SMEMBERS clanky:2:tags
1. “1“
2. “4“
```

Víme, že článek číslo 1 má všechny tagy 1-4 a článek 2 má tagy pouze 1 a 4. Redis disponuje řadou funkcí, mezi něž se řadí i funkce průnik. Následující kód demonstruje, jak zjistit, které články mají společné tagy 1 a 4.

```
>SINTER tag:1:clanky tag:4:clanky
1. “1“
2. “2“
```

Stejný princip funguje pro sjednocení a rozdíl. Pro sjednocení se používá klíčové slovo `SUNION a` pro rozdíl `SDIFF.`

Správa dat pomocí jazyka Java

Pro práci s Redisem v Javě se často využívá projekt Jredis nebo Jedis. Jredis podporuje verzi Redisu 2.6 a proto je vhodnějším kandidátem. Ke stáhnutí je zdarma na adrese https://github.com/alphazero/jredis, kde je součástí projektu i řada ukázkových příkladů použití. Na následujícím obrázku je vidět příklad práce s Redisem v Javě.

```
7
8  public class Redis {
9
10 private  void run ()
11 {
12     JRedis jredis = new JRedisClient();
13     int pocetObjektu = 100;
14     System.out.format ("Vytvareni a ukladani %d Java objektu do Redisu ...",
15     pocetObjektu);
16
17     for(int i=1; i<pocetObjektu; i++)
18     {
19         SimpleBean obj = new SimpleBean ("bean #" + i);
20         int id = jredis.incr("SimpleBean::next_id");
21         String key = "objects::SimpleBean::" + id;
22         jredis.set(key, obj);
23         jredis.sadd("object_set", obj);
24     }
25     List<SimpleBean>  members = decode (jredis.smembers("object_set"));
26     for(SimpleBean obj : members)
27     {
28         System.out.format(obj.toString() + "\n");
29     }
30 }
31 }
```

Obrázek 12 Přidání záznamů a jejich vypsání v jazyce Java

Projekt zapíše do databáze 100 záznamů a následně je vypíše. Pro připojení k databázi se využívá `JRedisClient()`, který bez parametrů provede defaultní připojení na portu 6379. Vytvoříme si objekt `SimpleBean`, který spolu s klíčem budeme uchovávat v databázi. `Id` se zde inkrementuje o jedničku v každém cyklu. Nakonec si vypíšeme z databáze všechny záznamy, které jsou v `object_set`.

4.8.4 Datové typy

Redis podporuje celkem 5 datových typů. Je to řetězec (string), seznam (list), set, sorted set (řazený set) a hash. Každý z těchto typů se hodí na uchovávání jiné struktury.

1. **Řetězec (String)**

String je nejčastějším typem v Redisu. Může obsahovat jakákoliv data, například i JPEG obrázek. Jeho velikost je omezena na 512MB. String se dá

využít pro mnoho věcí, například se používá jako počítadlo, jehož hodnota se mění pomocí příkazů `INCR`, `DECR`, `INCRBY`. Můžeme k němu přidat jiný řetězec pomocí příkazu `APPEND`.

2. **Seznam (List)**

Seznam je složen z řetězců, které jsou seřazené dle přidání. Je možné přidat nový záznam do redisu buď na začátek, nebo na konec. Příkaz `LPUSH` přidává záznam na začátek (zleva), `RPUSH` na konec (zprava). Seznam dokáže velmi rychle přistupovat k prvkům, které jsou na začátku nebo na konci, pomalý je v případě přístupu k prvkům uprostřed kolekce.

3. **Set**

Set je neseřazená kolekce stringů. Je možné přidávat a mazat v konstantním čase bez ohledu na počet záznamů. Set neumožňuje přidání záznamu se stejnou hodnotou vícekrát. Maximální počet záznamů v Setu je 2^{32}.

4. **Hash**

Hashe jsou mapy mezi atributem a hodnotou, takže jsou ideální pro reprezentaci objektů, například uživatel s atributy jméno, příjmení, věk a tak dále. Pokud budeme v Hash typu ukládat do stovky atributů, potřebný prostor pro uložení bude velmi malý, takže je možné uložit mnoho záznamů v relativně malé instanci Redisu.

5. **Sorted set**

Sorted Set, podobně jako Set, neukládá opakující se záznamy v kolekci. Rozdíl mezi nimi je v tom, že každý člen Sorted Set je spojen s určitým score (námi definovaná hodnota), který slouží k seřazení záznamů od nejmenší po největší hodnotu. Mezitím co členi Sorted Set musí být jedineční, score se může opakovat. Se seřazeným Setem je možné přidávat, odebírat a aktualizovat záznamy velmi rychle a to i uprostřed, což z něj činí jeden z nejpokročilejších datových typů u Redisu.

5 NoSQL v praxi

NoSQL není žádný mýtus, není to technologie, která by neměla využití. Naopak se čím dál více začíná objevovat na serverech, které se potýkají s velkým objemem dat. Jako důkaz může být využití na Facebooku, Twitteru a také v cloudu.

5.1 NoSQL a FaceBook

V roce 2010 společnost Facebook spustila novou verzi svých zpráv, které kombinují chat, email, SMS a další do konverzace v reálném čase. Potřebovali něco, co by zvládlo jejich službu zajišťující pro 350 milionů uživatelů odesílání 15 miliard zpráv jiným uživatelům měsíčně a podporovala by více jak 300 milionů uživatelů v chatu, kteří by odeslali měsíčně přes 120 miliard zpráv. Když společnost Facebook sledovala využívání své služby, našla 2 vzory, které se často opakovaly:

1. Pozměňováno bylo pouze malé množství dat
2. K velkému množství dat bylo přistupováno jen zřídka

Tyto dva body se měly zohlednit, když hledali náhradu za existující infrastrukturu služby zpráv. V roce 2008 zveřejnili svůj projekt Cassandra, NoSQL databázi typu klíč-hodnota, která již tehdy zajišťovala vyhledávání ve zprávách. Většina ostatních funkcí ale stále běžela na MySQL, ve kterém jejich databázové týmy měly největší zkušenosti. Změna technologie by byl proto problém, museli by buď upustit od svého vývoje Cassandry nebo vyškolit zaměstnance v používání nového a většího systému.

Rozhodli se tedy, že provedou test. Vytvořili framework, který měl za úkol vyhodnotit clustery u MySQL, Cassandry, HBase a pár dalších systémů. Nejlépe z testovaných dopadl HBase. Cassandry datový model se ukázal být nevhodným pro infrastrukturu zpráv a MySQL měl problémy se zvládnutím objemných dat, kdy výkon se razantně snížil. HBase byl vybrán také proto, že

technici z Facebook mají již mnoho zkušeností s HDFS, což je souborový systém, který využívá HBase.

Poněvadž funkce Zprávy dokáže přijímat data z různých zdrojů, jako je email a SMS, rozhodl se Facebook napsat vlastní aplikační server. Komunikuje s řadou služeb, např. s Haystack, který uchovává přílohy a obrázky, a dalšími službami, jako jsou facebook vztahy, nastavení soukromí a způsob doručení zpráv (zda má zpráva být poslána přes chat nebo SMS).[12]

5.2 NoSQL a Twitter

Twitter je velmi závislý na MySQL, přesto nasazuje NoSQL řešení na řadu problémů, pro které není MySQL ideální. Podle dostupných informací, se v roce 2011 generovalo 12 terabytů dat za den. Toto množství se ještě násobí každým rokem. Twitter používá několik produktů pro své potřeby. Je to Scribe, Hadoop, Pig, HBase, FlockDB a Cassandra.

Scribe je framework pro zaznamenávání logů, který byl vytvořen a uvolněn společností Facebook. Twitter používá Scribe pro zapisování logů do Hadoop. Tento open-source nástroj velmi zjednodušil práci a proto si teď taky Twitter může dovolit uchovávat 80 různých kategorií dat.

Twitter ukládá velké množství dat, mnohem větší, než které by byl schopný ukládat na jeden disk, proto potřebuje ukládat data do clusterů. K tomu využívá distribuci Hadoop zvanou Cloudera. Využívá se k analýze dat.

Pig je vyšší programovací jazyk, který nahradil Javu pro komunikaci s Hadoop, kvůli její složitosti. Společnost Yahoo vytvořila Pig kvůli velkému rozvoji Hadoop projektu. Potřebovala totiž jazyk, který by byl velmi lehký k naučení a pochopení.

Twitter používá HBase, do nějž se importují informace o uživatelích a umožňuje tak vyhledávací funkci lidí na Twitteru.

FlockDB je realtime distribuovaná databáze, která byla vytvořena a uvolněna Twitterem. FlockDB je v jádru MySQL, ale je značně upravený a velmi rychlý. Slouží třeba k určení, jakému uživateli se mají zobrazit jaké tweety. Pokud například @Microsoft odešle tweet na @Linux, tak by se měl zobrazit pouze lidem, kteří sledují zároveň @Microsoft i @Linux.

Cassandra se u Twitteru využívá pro geolokaci, kde se uchovává a dotazuje databáze bodů zájmu, a analýzu dat, kde se uchovávají výsledky dolování dat z databází uživatelů.

Twitter využívá svá data pro různé účely. Provádí s nimi jak jednoduché operace, jako např. kolik dotazů je provedeno denně, jaký je průměrný čas provedení dotazů, atd., tak i složitější operace, jako je porovnání různých typů uživatelů. Twitter zajímá také to, zda uživatelé na mobilních telefonech nebo uživatelé, kteří mají komunikačního klienta, využívají twitter jinak, než ti klasičtí. Nebo také, zda přidání nějaké nové funkce přiláká více uživatelů.[13]

5.3 NoSQL a Cloud

Většina dnešních populárních webových aplikací, jako je Google a Amazon, dosáhla díky horizontálnímu škálování vysoké dostupnosti a schopnosti ji zajistit i pro miliony uživatelů. Tento úspěch ukázal, že při horizontálním škálování, jako je u Google a Amazon, se do popředí dostávají NoSQL databáze a zastiňují tak relační protějšky. Jestli je škálovatelnost a dostupnost prioritou, pak NoSQL v cloudu může být ideálním řešením.

Existuje mnoho poskytovatelů cloudové služby. Ve většině případů máte dokonce na výběr, jaký NoSQL produkt chcete používat. Příkladem může být Amazon EC2 (Elastic Compute Cloud). Někteří poskytovatelé již nabízí nainstalované a nakonfigurované NoSQL databáze, které jsou připravené k používání. Asi nejznámějšími zástupci jsou Google App Engine a Amazon SimpleDB.

Google způsobil převrat ve světě cloudu, když vypustil svoji na použití jednoduchou infrastrukturu. Nebyl ale první, kdo přišel se svým řešením. Amazon EC2 již v té době byl zaběhlý. Byl to ale model Googlu, který mu zajistil úspěch a tak jeho cloudové řešení, Google App Engine, se rapidně rozšířilo za poměrně krátkou dobu.

Google App Engine poskytuje sandboxové prostředí pro aplikace, které jsou napsané v Javě, Pythonu nebo Go. Poskytuje vývojářům také bohatou škálu API a SDK pro tvorbu aplikací, běžící na Google app enginu.

Amazon SimpleDB je alternativou Google App Engine. Je součástí cloudu AWS, nemusíme ale využívat jiných služeb, můžeme používat pouze SimpleDB. Je to zástupce databází typu klíč-hodnota, kam spadá i Redis.[1]

6 Práce s Redisem

V kapitole 4.8 proběhlo první seznámení s Redisem. Tato kapitola si klade za cíl rozšíření vědomostí o tomto systému, o jeho instalaci, nastavení, podrobnější práci s daty a také srovnání s konkurenčními NoSQL produkty.

6.1 Instalace

Nainstalování Redisu můžeme provést buď stáhnutím z oficiálních stránek nebo práci nechat na package manageru naší Linuxové distribuce. Druhý zmíněný přístup má ale občas problém v tom, že neodkazuje na nejaktuálnější verzi. Proto se doporučuje využít odkaz

`http://download.redis.io/redis-stable.tar.gz`,
který obsahuje vždy nejaktuálnější verzi.

1. Z terminálu provedeme stáhnutí Redisu pomocí
   `wget http://download.redis.io/redis-stable.tar.gz`.
   Po vykonání příkazu se nám stáhne soubor do našeho domovského adresáře.

2. Soubor rozbalíme pomocí příkazu

`tar xvzf redis-stable.tar.gz`

3. Do adresáře se přesuneme pomocí příkazu `cd redis-stable`

4. Napíšeme příkaz `make`, abychom soubory zkompilovali.

Pokud vše proběhlo bez chyby, Redis je připraven k používání. Pro pohodlnější práci bych ale ještě doporučil zkopírování Redis serveru a rozhraní pro komunikaci s ním do adresáře `/usr/local/bin/`, abychom je mohli volat přímo bez nutnosti psaní cesty. To uděláme následovně:

5. `sudo cp redis-server /usr/local/bin/`. Při zadání toho příkazu se nás to pravděpodobně zeptá na heslo. Je třeba zadat heslo administrátora.

6. `sudo cp redis-cli /usr/local/bin/.`

Pro možnost spouštění aplikací přímo bez nutnosti zadávání cesty se předpokládá, že v proměnné `PATH` se nachází řetězec s cestou `/usr/local/bin`. Pokud tomu tak není, je třeba ho ještě doplnit.

7. Zadejte příkaz `gedit ~/.profile`.

8. Najděte řádku `PATH="$HOME/bin:$PATH"` a změňte ji na `PATH="$HOME/bin:$PATH:/usr/local/bin"`.

Pravě jsme úspěšně nainstalovali Redis a připravili ho k používání. Abychom si to ověřili, zadejme příkaz `redis-server` bez jakýchkoliv argumentů. Měla by se nám mimo jiné vypsat hláška o tom, že server je připraven přijmout připojení na portu 6379. Otevřeme si další terminál a v něm spustíme nástroj pro komunikaci se serverem pomocí `redis-cli`. Ten by se měl sám připojit

k Redis serveru. Ověřit si to můžeme tak, že do spuštěného nástroje zadáme příkaz `ping`. Pokud se nám jako odpověď dostane `PONG`, vše je v pořádku. Pokud budeme chtít Redis server ukončit, provedeme to v okně s nástrojem pomocí příkazu `shutdown`. Nástroj se ukončí příkazem `exit`.

6.2 Nastavení

V předchozí kapitole jsme pracovali s Redisem v defaultním, výchozím stavu. Tento stav má svoje výhody. Nemusíme nic nastavovat, aplikace se nám spustí ve funkční podobě. Je určen především pro testování a seznamování s prostředím. Má to ale taky svá omezení. Pokud bychom narazili na problém v předchozí části, nemohli bychom se podívat do logu, co chybu způsobilo, protože žádný takový soubor neexistuje. Defaultně tedy Redis žádné logy nesbírá. Možností nastavení je více, proto se této problematice věnuje vlastní sekce.

Začneme tím, že si vytvoříme adresáře pro konfigurační soubory a data následujícími příkazy:

`sudo mkdir /etc/redis` a `sudo mkdir /var/redis`.

V dalším kroku si zkopírujeme soubor `redis_init_script` v adresáři utils, který se nachází ve stažené distribuci Redisu do adresáře `/etc/init.d` a pojmenujeme ho podle portu, na kterém běží. Toho můžeme dosáhnout příkazem

```
sudo cp utils/redis_init_script /etc/init.d/redis_6379
```

Pokud bychom chtěli změnit číslo portu, na kterém budeme komunikovat, v souboru si najdeme řádku, kde se definuje proměnná `REDISPORT`. Tato hodnota ale může zůstat nezměněná.

Dále zkopírujeme konfigurační soubor `redis.conf`, který se nachází v kořenovém adresáři distribuce Redisu do adresáře `/etc/redis` a pojmenujeme si ho podle čísla portu. Provedeme to následovně: `sudo cp redis.conf /etc/redis/6379.conf.`

Následně si vytvoříme adresář uvnitř `/var/redis`, který bude sloužit jako pracovní adresář pro instanci Redisu a pojmenujeme ho opět podle čísla portu. `sudo mkdir /var/redis/6379`

Nyní přišel okamžik na samotné nastavování Redisu. To se provádí v `/etc/redis/6379.conf`. Začněme základním nastavením:

1. Najdeme v souboru „`daemonize`“ a změníme hodnotu na „`yes`“
2. Najdeme v souboru „`pidfile`“ a změníme jeho hodnotu na „`/var/run/redis_6379.pid`“
3. Pokud jsme měnili port, najdeme v souboru „`port`“ a změníme jeho hodnotu podle potřeby
4. Najdeme v souboru „`loglevel`“. Zde je defaultně hodnota „`notice`“. Pokud bychom chtěli provádět testování databází, můžeme změnit hodnotu na „`debug`“. V tomto stavu se nám do logu bude vypisovat velké množství údajů, které můžou být ale v řadě případů až nadbytečné a matoucí. Můžeme zvážit změnu na „`verbose`“, který není tolik nepřehledný jako „`debug`“ a přesto generuje řadu informací. Ponecháním hodnoty „`notice`“ ovšem nic nezkazíme.
5. Nalezneme v souboru „`logfile`“, který slouží jako cesta pro soubor s logovými údaji. Jeho hodnotu změníme na „`/var/log/redis_6379.log`“.
6. Najdeme „`dir`“ a jeho hodnotu změníme na „`/var/redis/6379`“. Tato proměnná slouží jako cesta k pracovnímu adresáři.

To by bylo k základnímu nastavení. Veškeré změny v souboru uložíme a soubor zavřeme. Na závěr provedeme následující příkaz: `sudo update-rc.d redis_6379 defaults`.[14]

6.3 Práce s daty

V Kapitole 4.8.3 jsme se již trochu seznámili s prací s daty v Redisu. V této kapitole se zmíním o tom, jakým způsobem dochází k ukládání dat, a blíže se seznámíme s příkazy dalších datových struktur, které Redis využívá.

6.3.1 Persistence dat

Jak již bylo zmíněno, Redis pracuje s databází v paměti a na disk je ukládáno na požádání. Existují dvě techniky, které se o ukládání starají. Jde o tzv. Snapshotting a AOF.

Snapshotting ukládá obraz databáze v určených intervalech. Například pokud předchozí obraz bych vytvořen před více jak 2 minutami a bylo zapisováno do databáze více než 100x, tak se vytvoří nový obraz. Toto nastavení může být změněno bez nutnosti restartování serveru. Každý obraz je vytvořen jako soubor s příponou .rdb, který obsahuje celou databázi. Tímto způsobem můžeme provádět pravidelné zálohování, které dokáže odvrátit případnou pohromu. Tato technika má ale taky své nedostatky, které vychází z logiky fungování. Pokud budeme ukládat data řekněme každé 2 minuty, tak můžeme v případě pohromy přijít až o 2 minutovou práci s databází.

AOF (Append only file) jde na to jiným způsobem. Pokaždé, když jsou data jakkoliv modifikována v paměti, příkaz na operaci je zaznamenán. Tento log je ve stejném formátu, ve kterém ho zadal uživatel při změně dat. AOF má výhodu v tom, že jde pouze o append-only, takže nemůže dojít k poškození souboru třeba při výpadku napájení. Dokonce, i když je zaznamenán jen napůl kompletní příkaz, ať už z jakéhokoliv důvodu, tak nástroj redisu na kontrolu AOF souboru dokáže problém snadno opravit. Poněvadž tento způsob

zaznamenává všechny příkazy, které mění data, do jednoho souboru, může se stát, že soubor se stane příliš velkým. V tomto případě dojde k úpravě stávajícího AOF souboru tak, aby obsahoval minimum příkazů, nutných pro vytvoření aktuálních dat a další logy začne zaznamenávat do nového AOF souboru. Zapnutí AOF proběhne příkazem `appendonly yes.`

Pro lepší představu, jak AOF zaznamenává příkazy, uvedu příklad:

```
redis 127.0.0.1:6379> SET key1 Hello
OK
redis 127.0.0.1:6379> APPEND key1 " World!"
(integer) 12
redis 127.0.0.1:6379> DEL key1
(integer) 1
```

Tabulka 3 Příkazy v Redis shellu

```
*2
$6
SELECT
$1
0
*3
$3
SET
$4
key1
$5
Hello
*3
$6
APPEND
$4
key1
$7
 World!
*2
$3
DEL
$4
key1
```

Tabulka 4 Výpis AOF souboru

Obecně se doporučuje používat obě metody najednou, pokud vám záleží na tom, aby data byla v naprostém bezpečí. Pokud vám sice záleží na datech, ale dokážete žít s pár minutovým výpadkem dat při nějakém neštěstí, pak je ideální použít samotný RDB snapshoot. Řada uživatelů používá samotný AOF, ale to nemůžu doporučit, protože se tak ochudíte o zálohy dat a o rychlejší restartování serveru.

6.3.2 Pokročilá správa dat

Již jsme si ukázali, jak se pracuje s daty, které jsme si uložili do setu. Jsou to příkazy `SADD` pro přidání záznamu a `SMEMBERS` pro vrácení všech hodnot, které jsou uložené pod klíčem. Rozšíříme si obzory ještě o `SREM`, který odstraní určitý člen daného setu.

```
redis>  SADD myset "Hello"
(integer) 1
redis>  SREM myset "Hello"
(integer) 1
```

V tomto příkladu jsme vytvořili hodnotu „Hello“, která je v setu myset a v dalším kroku jsme ji odstranili.

Pokud budeme uchovávat v naší databázi klíč-hodnota pouze jednoduché řetězce, které neexpandují do kolekcí, využívá se pro ukládání dat `SET` a pro získání `GET`. Pokud klíč již nějakou hodnotu uchovává, je přepsána, bez ohledu na její typ. Odstranění pak proběhne pomocí příkazu `DEL`.

```
redis>  SET mykey "Hello"
OK
redis>  GET mykey
"Hello"
redis>  DEL mykey
(integer) 1
```

Tyto příkazy `SET` a `GET` se ale nedají použít u jiných typů, pouze u řetězců. V případě, že naše data budeme uchovávat v seznamu (list), použijeme pro to

příkaz LPUSH a RPUSH. Ty přidávají nové záznamy buď zleva (LPUSH) nebo zprava (RPUSH). Spolu s tím, že k nim můžeme přistupovat také z obou stran pomocí LRANGE, dá se list využít jako stack. Odstranění pak proběhne pomocí LREM.

```
redis>  LPUSH mylist "one"
(integer) 1
redis>  RPUSH mylist "two"
(integer) 1
redis>  LRANGE mylist 0 0
1) "one"
redis>  LREM mylist -2 "two"
(integer) 1
```

Za příkazy LRANGE a LREM se uvádí čísla. U LRANGE čísla 0 0 znamenají začáteční a končící index prvku. Můžeme tedy tímto příkazem vypsat více prvků najednou. Pokud bychom k listu nechtěli přistupovat zleva, ale zprava, můžeme využít záporných čísel, kdy -1 značí poslední prvek, -2 předposlední, atd. U LREM je situace podobná. Zde se uvádí pouze jedno číslo a to znamená, kolik výskytů „two“ se má smazat. Pokud bude číslo kladné, začne se list prohledávat z leva, pokud záporné, tak zprava. V případě, že by číslo bylo 0, odstraní se všechny odpovídající prvky.

U sorted set se záznamy přidávají následovně:

```
redis> ZADD azset 1 "sset member 1"
```

Číslo 1 za azset znamená pozici nebo skóre. Abych to lépe vysvětlil, přidáme více prvků.

```
redis> ZADD azset 4 "sset member 2"
redis> ZADD azset 3 "sset member 3"
```

Pokud budeme nyní chtít prvky vypsat, uděláme to pomocí:

```
redis> ZRANGE azset 0 4
```

Čísla 0 4 opět značí počáteční a konečný index se stejnými pravidly, jako má výše zmíněný list. Výpis přidaných prvků v sorted set bude vypadat takto:

```
1. "SSET member 1"
2. "SSET member 3"
3. "SSET member 2"
```

Neřadí se tedy podle toho, jak byly záznamy přidány, ale podle jejich skóre.

Smazání záznamu je opět velmi podobné listu, s tím rozdílem, že místo `LREM` se používá příkaz `ZREM` a nepoužívá se čísel, značících počet výskytů.

```
redis>  ZREM azset "sset member 1"
```

Poslední kolekcí je hash. Uložení a získání hodnoty probíhá pomocí:

```
redis>  HSET myhash field1 "foo"
(integer) 1
redis>  HGET myhash field1
"foo"
```

Pokud bychom chtěli vypsat všechny hodnoty všech polí, můžeme využít příkaz `HGETALL`. Smazání pak proběhne pomocí příkazu `HDEL`.

```
redis>  HGETALL myhash
1) "field1"
2) "foo"
redis>  HDEL myhash field1
(integer) 1
```

Na následující tabulce jsem se pokusil shrnout všechny důležité příkazy pro správu dat u Redisu

	Čtení	**Zápis**	**Mazání**

String	`GET`
	
	`např. GET mykey`
		`SET`
		
		`např. SET mykey "Hello"`
			`DEL`
			
			`např. DEL mykey`

Set	SMEMBERS např. SMEMBERS myset	SADD např. SADD myset "Hello"	SREM např. SREM myset "Hello"
Sorted set	ZRANGE např. ZRANGE azset 0 4	ZADD např. ZADD azset 1 "sset member 1"	ZREM např. ZREM azset "sset member 1"
List	LRANGE např. LRANGE mylist 0 0	LPUSH, RPUSH např. LPUSH mylist "one" RPUSH mylist "two"	LREM např. LREM mylist -2 "two"
Hash	HGET např. HGET myhash field1	HSET např. HSET myhash field1 "foo"	HDEL např. HDEL myhash field1

Tabulka 5 Příkazy pro správu dat v Redisu[15]

6.4 Porovnání s konkurencí

Porovnávat NoSQL databáze bývá problém, protože každá je jiná, funguje na jiném principu a hodí se do jiných podmínek. Proto jsem se rozhodl vybrat ty porovnávací kritéria, která mají nějakou vypovídající hodnotu a určují, jak se bude uživateli s databází pracovat. Konkrétně jde o využití databází pro různé situace, dostupnost materiálů, ať už na internetu nebo v podobě knih, podpora operačních systémů, to jest, kde můžeme databáze oficiálně využívat, kde je to nedoporučeno a kde to nelze vůbec a posledním měřítkem je možnost využití různých programovacích jazyků pro práci s databází.

6.4.1 Využití

Každá databáze, ať již mluvíme o SQL nebo NoSQL technologiích, má své výhody a nevýhody. Proto se také hodí na nasazení v odlišných situacích. V této kapitole se pokusím nastínit, o jaké situace jde.

MongoDB

MongoDB se využívá v situacích, pokud potřebujeme dynamické dotazování. Je také vhodným kandidátem, pokud chceme definovat indexy a ne MapReduce funkce. MongoDB také dosahuje vysokého výkonu u velkých databází. Doporučil bych ho tedy do míst, kde by se za normálních okolností použilo MySQL, ale nechtěli bychom být omezováni nutností mít předem definované atributy.

Redis

Redis se používá v situacích, kdy dochází k časté změně velkého množství dat, ale databáze je rozumné velikosti, protože by se měla vejít z velké většiny do operační paměti. Případný zbytek by byl pak uložen ve swap oblasti. Příkladem použití může být vývoj cen akcií, sběr dat v reálném čase, komunikace v reálném čase.

HBase

Hbase je asi nejlepší možností pro situace, kdy potřebujeme spouštět MapReduce funkce na velkém množství dat. Doporučuje se využívat Hadoop s HDFS souborovým systémem. Nejlepší využití najde u vyhledávacích enginů a analyzování logových údajů. Obecně je vhodný pro scanování velkých, dvourozměrných tabulek, které nemají joiny.

Cassandra

Cassandra je ideální, pokud potřebujeme více psát, než číst. Typicky jde o situaci logování údajů. Proto tento systém je často k nalezení v bankovnictví, finančním průmyslu, protože je zde rychlejší zapisování, než čtení.

	Využití v situaci
MongoDB	Dynamické dotazování, nestrukturovaná data
Redis	Častá změna velkého množství dat
HBase	Čtení logů
Cassandra	Zapisování logů

Tabulka 6 Využití NoSQL produktů

Shrnutí

Při přechodu z SQL na NoSQL je třeba vědět, co od naší databáze očekáváme. Pokud nám vadilo omezení tabulek, kdy jsme museli mít předem definovanou strukturu, můžeme využít MongoDB. Tento systém je v současnosti taky jedním z nejpoužívanějších. Redis použijeme na rychlé databáze s častou změnou dat. Cassandru využijeme pro logování údajů a HBase na jejich analýzu.

6.4.2 Dostupnost materiálů

Řádná dokumentace je neodmyslitelnou součástí každé aplikace. To platí dvojnásob u těch, které jsou určeny pro běh na serveru, protože bývají často velmi konfigurovatelné, a také proto, že komunikují s řadou jiných aplikací. Tuto součinnost je třeba zaznamenat, protože bývá klíčová pro pochopení vzájemné komunikace.

MongoDB

MongoDB má na svých stránkách velmi kvalitní dokumentaci, která zahrnuje instalace na různých operačních systémech, první kroky s databází, seznam příkazů nad databází, řadu tutoriálů a spoustu dalších užitečných informací, které začínající uživatel může potřebovat. Poněvadž je MongoDB asi nejrozšířenější NoSQL databáze, má taky největší základnu uživatelů a tudíž se dá narazit na internetu na různé články, které se zabývají zmíněnou databází.

Většina z nich je ovšem samozřejmě v angličtině. Knih, které se ať přímo nebo pouze okrajově zabývají touto databází, je kolem 30[1]. Společnost 10gen, která spravuje MongoDB, má na svých stránkách dostupný online kurz zdarma, po jehož absolvování dostanete certifikát. Máte na výběr, zda chcete pracovat s MongoDB jako vývojář v javě, pythonu nebo jako databázový administrátor.

Redis

Dokumentace Redisu je značně tenčí, než v případě MongoDB, a trochu nepřehledná. Obsahuje ale základní průvodce instalací, nastavením, články o vlastnostech Redisu, jak optimalizovat instance a taky odkazy na pár knih, které se Redisem zabývají. Na svých stránkách ovšem mají zajímavý projekt. Jde o Try Redis, což je online výukový tutoriál, kterým procházíte zadáváním požadovaných databázových příkazů. Existuje pár článků o Redisu v češtině, zbývajících pár je v angličtině. Počet knih, které jsou dostupné s touto problematikou, je kolem 11.

HBase

Dokumentace HBase je na oficiálních stránkách poměrně rozsáhlá a dobře se v ní orientuje, protože je dělená formou kapitol. Dokumentace začíná počátečním stažením projektu a jeho instalací, spuštěním, konfigurací a řadou dalších kapitol, ve kterých jsou příklady pro snazší pochopení problematiky. Na internetu je více článků, než v případě Redisu a to také díky tomu, že často bývá objektem srovnávání s jinou databází. Tomu odpovídá i počet knih, které se v nějaké míře zabývají touto problematikou, jejichž počet se pohybuje okolo 21.

[1] Vyhledáváno pomocí Google books

Cassandra

Oficiální stránky Cassandry, co se dokumentace týče, jsou velmi strohé a omezují se pouze na úvodní nainstalování, spuštění, základní používání a v pár řádcích i na konfiguraci. Mnohem lépe je na tom situace na stránkách DataStax, což je partnerská společnost, kde je přehlednější dokumentace, doplněná instruktážními videi. Poněvadž se Cassandra využívá na známých sociálních sítích, píše se také o ní poměrně často v různých internetových článcích a elektronických publikacích. Tématem Cassandry se zabývá přibližně 14 knih.

	Dostupnost materiálů
MongoDB	Kvalitní dokumentace, řada článků, okolo 30 knih
Redis	Nekvalitní dokumentace, pár článků, okolo 11 knih
HBase	Kvalitní dokumentace, řada článků, okolo 21 knih
Cassandra	Průměrná dokumentace, řada článků, okolo 14 knih

Tabulka 7 Dostupnost materiálů

Shrnutí

U všech zmíněných databází je k dispozici dokumentace, ať již oficiální nebo pod záštitou jiné společnosti. V případě MongoDB, HBase a Cassandry, je dokumentace kvalitní a poměrně rozsáhlá, takže se dá využít jako hlavní zdroj informací při seznamování s danou technologií. Kromě dokumentací je u těchto tří zmíněných technologií i dostatek článků a knih, ze kterých může uživatel čerpat další informace. Redis ve srovnání s ostatními technologiemi v oblasti dokumentace zaostává. Rozsahem je kratší než ostatní a celkově se v ní hůře orientuje. Článků a elektronických publikací není na internetu moc, proto uživateli často nezbude nic jiného, než si koupit jednu z mála knih, která se problematikou zabývá. Navíc pouze v angličtině.

6.4.3 Podpora operačních systému

Ne všechny databázové systémy můžeme provozovat na libovolném operačním systému. Většinou to bývá tak, že tvůrce databázového systému má jednu platformu, pro kterou je plná podpora a pro ostatní platformy vzniká bokem. Poněvadž je většina NoSQL produktů open-source, jejich hlavní platforma je Linux. Ty nejrozšířenější produkty jdou ovšem dál a poskytují podporu většině operačních systémů.

MongoDB

Dle stránek výrobce je MongoDB připraveno k používání jak na Linuxu, OS X, Solarisu, tak i na Windows. Vše oficiálně, takže by verze měly být stabilní.

Redis

Redis je určený podle výrobce primárně pro Linux. Na oficiálních stránkách je sice ke stáhnutí verze pro Windows, ale jde o projekt třetí strany, konkrétně Microsoft Open Tech. Tato verze je pouze experimentální a nedosahuje potřebných kvalit pro plné nasazení. Proto se doporučuje pouze pro seznámení s technologií a základní vývoj.

HBase

HBase poskytuje primárně podporu pro Linux. Nicméně poněvadž je projekt napsán v javě, projekt je plně použitelný napříč platformami, včetně Windows. Pro použití na Windows se využívá Cygwin, který emuluje Linuxové prostředí.

Cassandra

Cassandra podporuje Linux, Windows 7, Windows server 2008 a OS X.

Na následující tabulce je shrnuto, jaká je podpora jednotlivých NoSQL produktů na různých operačních systémech.

	Linux	OS X	Windows	Solaris
MongoDB	Podporuje	Podporuje	Podporuje	Podporuje
Redis	Podporuje	Nepodporuje	Nedoporučeno	Nepodporuje
HBase	Podporuje	Nedoporučeno	Nedoporučeno	Nepodporuje
Cassandra	Podporuje	Podporuje	Podporuje	Nepodporuje

Tabulka 8 Podpora NoSQL produktů na OS

Shrnutí

Z tabulky je vidět, že na Linuxu můžeme bez problémů provozovat všechny zmíněné NoSQL databáze. Vývoj jde ale dál a čím dál větší množství produktů je portováno pro větší pohodlí vývojářů na ostatní systémy. Nejdále je v tomto ohledu MongoDB, které pracuje na všech zmíněných operačních systémech.

6.4.4 Podpora programovacích jazyků

NoSQL databáze jsou určeny pro velké objemy dat. Zadávat proto miliony záznamů v shellu databáze, by bylo na dlouhé lokte. Pro tyto situace se tedy využívá klientů, kteří se připojí k databázi, a přes ně ji můžeme naplnit. Každé rozhraní databáze je ovšem jiné a proto se taky musí použít jiných ovladačů pro komunikaci s databází.

Řada tvůrců NoSQL databází poskytuje ovladače, které vytvořili sami. Ty bývají taky jimi spravované a dosahují tak nejlepší spolehlivosti. Existuje ale i řada ovladačů, které nejsou vytvořeny přímo vývojáři databázových systémů, ale komunitou uživatelů. Tímto se rozšiřuje možnost použít databázové klienty, které využívají různých programovacích jazyků.

V následující tabulce je znázorněno, jaké nejznámější programovací jazyky je možné použít pro práci s NoSQL databázemi. Seznam je ve skutečnosti větší, pro kompletní výpis podporovaných jazyků navštivte stránky výrobce.

	MongoDB	Redis	HBase	Cassandra
C	Podporuje	Podporuje	Nepodporuje	Nepodporuje
C++	Podporuje	Komunita	Nepodporuje	Komunita
C#	Podporuje	Komunita	Nepodporuje	Komunita
Java	Podporuje	Komunita	Podporuje	Podporuje
Javascript	Podporuje	Nepodporuje	Nepodporuje	Nepodporuje
Perl	Podporuje	Komunita	Nepodporuje	Komunita
PHP	Podporuje	Komunita	Nepodporuje	Komunita
Python	Podporuje	Komunita	Komunita	Komunita
Ruby	Podporuje	Komunita	Nepodporuje	Komunita

Tabulka 9 Podpora programovacích jazyků u NoSQL databází

Shrnutí

Z tabulky je vidět, že opět nejlépe je na tom MongoDB, které podporuje všechny nejrozšířenější programovací jazyky přímo od vývojářů. Velkou komunitní podporu má ovšem Redis, který tak nabízí snadné používání své databáze skrz řadu jazyků. Obdobně je na tom i Cassandra. Oproti ostatním v tomto ohledu ovšem zaostává HBase, který se to ale snaží vynahradit podporou Thriftu, tj. rozhraní, které umožňuje vytvořit ovladač v jakémkoliv programovacím jazyce.

6.4.5 Vyhodnocení

Každá ze zmíněných databází má své výhody a nevýhody. V kapitole 6.4 jsem se pokusil nastínit problematiku využití databází, dostupnosti materiálů k nim, podpoře operačních systému a možnosti použití různých programovacích jazyků pro práci s nimi. Za vítěze by se dalo považovat MongoDB, které má dobře zpracovanou dokumentaci, podporuje většinu dnešních operačních systémů a již od výrobce má podporu velkého rozsahu programovacích jazyků. Druhé místo bych přiřknul Cassandře, která má sice od výrobce spíše podprůměrnou dokumentaci, ale jejich partnerská firma tento neduh dohnala.

Zároveň taky podporuje většinu operačních systémů a komunita okolo Cassandry poskytuje podporu programovacím jazykům. O třetí a čtvrté místo se dělí Redis s HBase. Podpora operačních systémů u těchto dvou produktů je velmi podobná. Redis má velkou podporu komunitních ovladačů pro práci s databází při využití programovacích jazyků, kdežto HBase je zase lépe zdokumentován a snáze se k němu shání potřebné informace.

V následující tabulce jsem se pokusil zachytit výše zmíněné vlastnosti. Hodnocení každé z vlastností proběhlo vzájemným porovnáním všech produktů a vytvořením stupnice kvality, kde nejlepšího hodnocení dosáhne produkt, který v dané kategorii dopadl nejlépe.

	Využití	Dokumentace	Podpora OS	Podpora progr. jazyků
MongoDB	Dynamické dotazování, nestrukturovaná data	1	1	1
Redis	Častá změna velkého množství dat	4	3	2
HBase	Čtení logů	2	3	4
Cassandra	Zapisování logů	3	2	2

Tabulka 10 Porovnání NoSQL produktů. Hodnocení jako ve škole (1 - nejlepší, 5 - nejhorší)

7 Práce s MySQL

V této části navážu na kapitolu 4.3., kde bylo provedeno základní seznámení s MySQL databází. Rozšířím obsah o instalaci, konfiguraci, správu dat a porovnání s konkurencí. Ani zde nebudu zabíhat příliš do detailů, protože

návodů na práci s MySQL je na internetu dostatek a není to hlavním cílem práce.

7.1 Instalace

Pro práci s MySQL je třeba mít nainstalovány 2 základní části. Jde o samotný MySQL server a MySQL klient. Toho můžeme jednoduše dosáhnout tak, že napíšeme:

```
sudo su
apt-get install mysql-server mysql-client
```

prvním příkazem získáme administrátorská práva (musíme zadat heslo administrátora) a druhým příkazem stáhneme a začneme instalovat MySQL.

V průběhu instalace se nás program zeptá pouze na jednu věc a tou je heslo k MySQL. Můžeme zde heslo zadat nebo se k nastavení vrátit později pomocí příkazu:

```
mysqladmin -u root password novéheslo
```

Po doinstalování si můžeme její úspěšnost ověřit příkazem:

```
mysql -u root -p
```

Ten vás požádá o heslo, které jste zadali při instalaci, a následně se připojí k serveru.

Tento přístup je nejjednodušší. Často ale takto nezískáme nejaktuálnější verzi MySQL. Tu bychom museli stáhnout ze stránek http://dev.mysql.com/downloads/mysql/, například pomocí programu wget. Nejdříve bychom si ale museli doinstalovat `libaio1`, což je knihovna, umožňující asynchronní I/O systémová volání, která jsou důležitá pro výkon databází. Série příkazů pro stáhnutí této knihovny a MySQL vypadá následovně:

```
apt-get install libaio1
cd /usr/local
wget -O mysql-5.6.10-linux2.6-x86_64.tar.gz
http://dev.mysql.com/get/Downloads/MySQL-5.6/mysql-
5.6.10-linux-glibc2.5-
x86_64.tar.gz/from/http://cdn.mysql.com/
```

Nyní máme v adresáři `/usr/local` stažený archiv s MySQL. Předtím, než ho rozbalíme, vytvořme si skupinu a uživatele pojmenovaného „mysql“ kvůli bezpečnosti.

```
groupadd mysql
useradd -r -g mysql mysql
```

Teď je na řadě rozbalení archivu, přejmenování adresáře na výstižnější a především kratší verzi a nastavení práv vlastníka a skupiny rozbaleného archivu na „mysql“.

```
tar xvfz mysql-5.6.10-linux2.6-x86_64.tar.gz
mv mysql-5.6.10-linux2.6-x86_64 mysql
cd mysql
chown -R mysql .
chgrp -R mysql .
```

Už zbývá jen vytvoření nezbytných databází (jako je mysql) a změna pár vlastnických práv.

```
scripts/mysql_install_db –user=mysql
chown -R root .
chown -R mysql data
```

MySQL je nyní připraveno k používání. Poněvadž jsou ale příkazy jako `mysql` a `mysql_secure_installation` umístěny v `/usr/local/mysql/bin`, která není součástí `PATH`, museli bychom je volat se zadáním celé cesty. To se dá obejít třeba vytvořením symlinku:

```
ln -s /usr/local/mysql/bin/* /usr/local/bin/
```

7.2 Nastavení

MySQL používá jako svůj konfigurační soubor „`my.cnf`“, který se nachází v adresáři `/usr/local/mysql` a také v `/usr/mysql`.

Pokud bychom chtěli logovat obecné údaje, musíme tuto funkci nejdříve zapnout, protože ve výchozím stavu je tato řádka zakomentovaná. Otevřeme si proto soubor `/usr/mysql/my.cnf` a najdeme řádku začínající `general_log_file` a odkomentujeme jí. Důležitá poznámka: Zapnutím této funkce snižujeme výkon databáze, proto by se měla využívat s rozvahou. Logování chyb je standardně zapnuto (řádka začínající `log_error`), takže o důležité informace nejsme ochuzeni. Pro seznam všech systémových proměnných a jejich význam doporučuji navštívit stránku http://dev.mysql.com/doc/refman/5.5/en/server-system-variables.html.
Nastavení je vyladěno na klasické používání, takže není třeba měnit žádné možnosti.

V případě, že bychom MySQL provozovali na velkém systému, museli bychom změnit některé hodnoty pamětí (`key_buffer_size`, `read_buffer_size`, `read_rnd_buffer_size`) na vyšší, jinak vše ostatní může zůstat nezměněné.

7.3 Práce s daty

V kapitole 3.4.2. jsem uvedl základní příkazy pro práci s daty a databází. V Linuxu můžeme pro spravování dat i databází u MySQL použít jak konzolový nástroj, tak i jiného, pohodlnějšího klienta. Často se ve spojitosti s MySQL mluví o PHPMyAdmin. Jde o nástroj napsaný v jazyce PHP, který umožňuje jednoduchou správu databáze pomocí webového rozhraní. Patří mezi jeden z nejpopulárnějších nástrojů pro správu databáze.

Stejně ale jako u NoSQL jsme pracovali v terminálu, tak ani v tomto případě neuděláme výjimku. Spuštění klienta provedeme pomocí příkazu

```
mysql -u root -p
```

Po zadání hesla bychom se úspěšně měli přihlásit k MySQL serveru. Pokud se nám to nepodařilo, tak je třeba zkontrolovat, zda server běží. Pokud běží, zkusíme ho restartovat pomocí příkazu

```
service mysql restart
```

Pokud server neběží, spustíme ho pomocí příkazu

```
service mysql start
```

V případě, že nepomůže ani toto, můžeme prozkoumat soubor s logovými záznamy chyb. Řekněme ale, že se nám podařilo připojit a chtěli bychom vytvořit novou databázi, která bude uchovávat naší sbírku knih. To provedeme pomocí

```
mysql> CREATE DATABASE knihovna;
```

Nesmíme zapomenout na ukončující středník, jinak náš příkaz bude pokračovat po stisknutí ENTER na další řádce. Jako odpověď je nám text, který oznamuje úspěšnost příkazu.

```
Query OK, 1 row affected (0.00 sec)
```

Po tom, co jsme si vytvořili databázi, musíme říct MySQL, aby ji začal používat. K tomu slouží příkaz

```
mysql> USE knihovna;
```

Abychom mohli ukládat data do databáze, musíme si pro ně vytvořit tabulku. Tu si pojmenujeme „knihy“. Zároveň si také definujeme, které atributy budeme chtít uchovávat. U knihy nás bude zajímat její název, autor, rok vydání, nakladatelství a ISBN. Vytvoření takovéto tabulky zapíšeme pomocí

```
mysql> CREATE TABLE knihy (id INT, nazev VARCHAR(50),
autor INT, rokVydani INT, nakladatelstvi INT, ISBN
VARCHAR(13));
```

Tento příkaz si zaslouží bližší rozebrání. Prvním atributem je zde id. To nám bude sloužit jako primární klíč, to znamená, že bude určovat jedinečnost celého záznamu. Místo id by se taky jako primární klíč dalo použít ISBN, za předpokladu, že v naší knihovně bude stejná kniha vícekrát. Dalším atributem je název knihy. Tomu jsme nastavili maximální velikost na 50 znaků. Následující atribut je o něco zajímavější. Je jím autor knihy, ale není zde typu VARCHAR, jak by se mohlo čekat, nýbrž INT. V kapitole 3.1 byla zmíněna problematika cizích klíčů a toto je ten případ. Dá se totiž předpokládat, že v naší databázi budeme mít více knih od stejného autora. V tom případě bychom měli databázi zbytečně objemnější. Je výhodnější vystrčit autory do samostatné tabulky, kde bude id autora a jeho jméno s příjmením a následně při dotazování tyto 2 tabulky pomocí klíčů spojovat. Stejná situace je v případě nakladatelství. Rok vydání je INT a ISBN 13ti znakový řetězec.

Následně musíme vytvořit tabulku pro autory a nakladatelství, což bude obdobné jako u předchozího příkazu.

```
mysql> CREATE TABLE autori (id INT, jmeno
VARCHAR(50), prijmeni VARCHAR(50));

mysql> CREATE TABLE nakladatelstvi (id INT, nazev
VARCHAR(50));
```

Struktura databáze je hotová, teď ji zbývá naplnit daty. To se provádí pomocí příkazu `INSERT INTO`, který v našem případě vypadá

```
mysql> INSERT INTO autori (id,jmeno,prijmeni)
VALUES(1,"Agatha","Christie"); INSERT INTO autori
(id,jmeno,prijmeni) VALUES(2,"Alois", "Jirásek");

mysql> INSERT INTO nakladatelstvi (id,nazev)
VALUES(1,"Euromedia Group"); INSERT INTO
nakladatelstvi (id,nazev) VALUES(2,"Knižní expres");
```

```
mysql> INSERT INTO knihy (id, nazev, autor,
rokVydani, nakladatelstvi, ISBN) VALUES(1,"Staré
pověsti české",2,2001,2,80-86132-59-5); INSERT INTO
knihy (id, nazev, autor, rokVydani, nakladatelstvi,
ISBN) VALUES(2,"Tajemný protivník",1,2004,1,80-242-
1173-4); INSERT INTO knihy (id, nazev, autor,
rokVydani, nakladatelstvi, ISBN) VALUES(3,"Plavý
kůň",1,2006,1,80-242-1604-3);
```

Právě jsme do všech našich třech tabulek vložili záznamy. Knížky máme 3, přičemž 2 z nich napsala stejná autorka a byly vydány stejným vydavatelstvím. Na příkladu je vidět, že není třeba pro uložení záznamu stisknout pokaždé enter, rozhodující je zde středník, který odděluje jednotlivé příkazy. Zřetězováním příkazů se ale vystavujeme vyššímu riziku překlepu, kvůli nepřehlednosti.

Data uložena máme, ale byly by nám k ničemu, pokud bychom se na ně nemohli podívat a pracovat s nimi. K tomu slouží dotazovací jazyk SQL. Při správném formulování dotazu získáme snadno z databáze potřebná data.

Pokud bychom chtěli například vypsat tabulku `knihy`, uděláme to jednoduše takto

```
mysql> SELECT * FROM knihy;
```

Výsledek ale nebude asi takový, jaký bychom si přáli ho mít. U autora a nakladatelství se nám totiž vypíšou pouze čísla, nikoliv jména a názvy. K tomu slouží klíčové slovo `JOIN`, které dokáže spojovat záznamy v jednotlivých tabulkách podle daného kritéria. `JOIN` má několik variant. Jde o `LEFT JOIN`, `RIGHT JOIN` a `INNER JOIN`. Liší se v tom, že `LEFT JOIN` a `RIGHT JOIN` zobrazí z jedné tabulky všechny záznamy a z druhé pouze související data, kdežto `INNER JOIN` zobrazí pouze data, která mají mezitabulkový vztah.

Abychom tedy vypsali všechny knihy, které máme v naší databázi, spolu s autory a nakladatelstvím, modifikovali bychom dotaz do podoby

```
mysql> SELECT knihy.nazev,autor.jmeno,
autor.prijmeni, knihy.rokVydani,
nakladatelstvi.nazev, knihy.ISBN FROM knihy INNER
JOIN autori ON knihy.autor = autori.id INNER JOIN
nakladatelstvi ON knihy.nakladatelstvi =
nakladatelstvi.id;
```

Tímto dotazem docílíme, že se nám vypíše název knihy, jméno autora, rok vydání, nakladatelství a ISBN. `INNER JOIN` se tedy použije tak, že na obou jeho stranách jsou tabulky, mezi kterými existuje nějaká závislost a za klíčovým slovem `ON` se tato závislost mezi atributy vyjádří.

Pokud bychom chtěli navíc tento výpis omezit nějakým kritériem, můžeme využít klíčové slovo `WHERE`, které slouží jako filtr výsledků a píše se na konec dotazu. Klauzule `WHERE`, která by ponechala pouze knihy, které napsal „Jirásek“, by vypadala

```
WHERE autori.prijmeni = "Jirásek"
```

7.4 Porovnání s konkurencí

V poslední době se čím dál více začínají objevovat informace o projektu MariaDB. Jedná se o fork[2] MySQL, který oslavil první velký úspěch tím, že část Wikipedie nyní využívá tuto technologii. Jde o menší, hubenější a rychlejší verzi MySQL, která vznikla v důsledku nespokojenosti se spravováním MySQL jejím vlastníkem, společností Oracle. Tato databáze je kompatibilní s MySQL, to znamená, že případný přechod na MariaDB je bezproblémový. Stačí smazat MySQL a nainstalovat MariaDB. Zatímco

[2] Alternativní větev programu, která je často vyvíjena nezávisle a jinými lidmi

MySQL je zdarma pouze za určitých podmínek, MariaDB je v současné době plně zdarma.[16]

Rozdílů mezi MySQL a MSSQL už je více. Za prvé je třeba zmínit, že MSSQL je v použitelných verzích placené. Zdarma se dá vyzkoušet pouze verze Express, která má ale řadu omezení, proto se nevyužívá ve větším měřítku. MSSQL je produktem Microsoftu, který vyvíjí své aplikace především pro svoji platformu Windows, proto ho naplno nevyužijete pod systémem Linux. Nicméně v roce 2011 Microsoft vypustil klienta pro práci s MSSQL pod RedHat Enterprise Linux, aby si tak alespoň částečně rozšířil pole své působnosti. MySQL je především zaměřeno na internetové servery a open-source projekty. Spolu s tím, že je pro běžného uživatele zdarma, se stal hlavním databázovým systémem, který se používá na internetu. MSSQL je spíše zaměřeno na podniky, kterým nabízí větší funkcionalitu, velmi dobře zpracovaný systém uživatelských práv a dobrou spolupráci s Visual Studiem[3].

Pokud bychom chtěli porovnávat MySQL a Oracle, zjistíme, že situace je v lecčems podobná porovnání s MSSQL. Oracle je sice primárně určen pro Linux, ale má dobrou podporu i pro Windows. Oracle je podobně jako MSSQL určen spíše pro větší společnosti. Zdarma nabízí pouze verzi Express, která je taky značně omezená.

8 Porovnání Redisu s MySQL

Pro porovnání SQL a NoSQL technologií jsem u každé z nich vybral jednoho zástupce, naplnil ho daty a provedl na něm testy, které se skládají z měření potřebné časy pro přidání, změnu a odebrání záznamů. Aby testy byly co nejvíce vypovídající, bylo třeba zvolit prostředí, které bude oběma produktům nejpřirozenější. Poněvadž NoSQL databáze jsou určeny především pro Linuxové prostředí a MySQL má v něm také kořeny, operačním systémem

[3] Vývojové prostředí od společnosti Microsoft

byla zvolena distribuce Linuxu, zvaná Ubuntu, ve verzi 12.10. Celé testování probíhalo na jedné stanici, jejíž parametry jsou:

procesor: Intel quad-core 9400 2.66 GHz

operační paměť: 2x2 GB 1600 Mhz

pevný disk: 40GB IDE

8.1 Vytvoření datasetu

Aby bylo možné porovnávat databáze z hlediska rychlostí, je třeba mít testovací data. Pro tyto potřeby jsem vytvořil v javě program, který do souboru .txt vytvoří 5 milionů záznamů, přičemž každý z nich má 6 atributů a id. Jde o smyšlená, náhodně generovaná data, která simulují logový soubor a slouží pouze pro potřeby testu.

```
private static void vytvorDataSet()
{
    try
    {
        FileWriter fw = new FileWriter("dataSet.txt");
        BufferedWriter bf = new BufferedWriter(fw);
        Random r = new Random();

        for(int i=0; i<5000000; i++)
        {
            bf.write(String.valueOf(i+1)+";"+
            String.valueOf(r.nextInt(20000))+ ";"+
            String.valueOf(r.nextDouble())+ ";"+
            String.valueOf(r.nextFloat())+";"+
            String.valueOf(r.nextInt(2))+";"+
            String.valueOf(r.nextInt())+";"+
            String.valueOf(r.nextDouble())+"\n");
        }
        bf.flush();
        bf.close();
    }
    catch(Exception ex){}
}
```

Obrázek 13 Vytvoření datasetu

Záznamy se do souboru zapisují v cyklu, který proběhne přesně 5 milionkrát. Vyskytují se tu hodnoty typu integer, double a float. Vytvoření takovéhoto množství dat je otázkou méně než minuty.

8.2 Naplnění Redisu daty

Naplnění daty se u redisu je možné třemi způsoby. Prvním je klasické zadávání příkazů řádek za řádkem v shellu redisu. Tento přístup je použitelný ale pouze u přidávání pár záznamů, protože by jinak byl časově velmi náročný. Druhým přístupem je využití klienta jako náhrady za shell a vkládání záznamů díky nějakému jazyku, například javy. Třetím a nejrychlejším způsobem je využití nové funkce v redisu, která je dostupná v posledních verzích. Jde o vytvoření si textového souboru s již předpřipravenými příkazy, který potom pomocí roury přesměrujeme na vstup shellu redisu.

První přístup už byl zmíněn v kapitole 6.3.1. Jeho syntaxe pro přidání záznamu by vypadala

```
redis> ZADD z:1 1  9781
```

kde „`z:1`“ značí kolekci a klíč („z“ je kolekce, „1“ je klíč), „1“ je score, které určuje, že hodnota bude v setu jako první a „9781“ je hodnota, která odpovídá klíči.

Podobný příkaz bychom ale museli udělat 30000000x, abychom uložili všech 5000000 záznamů, přičemž každý z nich má 6 atributů (5000000 x 6). Proto je lepší si celý proces zautomatizovat pomocí cyklu. K tomu se dá využít Jedis, který si připojíme k naší aplikaci.

```
static public void pridejPrvky()
 {
     JedisPool  pool=null;
     Jedis jedis=null;
     try
     {
         pool= new JedisPool(new JedisPoolConfig(), "localhost");
         jedis = pool.getResource();
         FileReader fr = new FileReader("dataSet.txt");
         BufferedReader br = new BufferedReader(fr);
         String s;

         while((s = br.readLine()) != null)
          {
             String[] zaznam = s.split(";");
             for(int i =1; i<zaznam.length;i++)
             {
                  jedis.zadd("z:"+zaznam[0],i,zaznam[i]);
             }
          }
             fr.close();
      }
     catch (Exception ex) { }
     finally
     {
         pool.returnResource(jedis);
         pool.destroy();
     }
 }
```

Obrázek 14 Přidávání prvků do Redisu pomocí jazyka Java

Poněvadž jsou jednotlivé atributy v souboru oddělené středníkem a záznamy novou řádkou, je třeba si každou řádku rozdělit pomocí metody split na jednotlivé atributy. Každou tuto hodnotu pak následně uložíme pomocí metody `ZADD`, která slouží pro přidávání nových položek do sorted set. Parametr je ve formátu klíč, score, hodnota. Na testovaném počítači bylo dosáhnuto výsledku 240 tisíc přidaných záznamů za minutu. Je třeba si ale uvědomit, že většina potřebného času byla využita na parsování dokumentu. Jakých výsledků je schopen dosáhnout Redis bez potřeb parsování, tedy prostého přidávání záznamů, se věnuje kapitola 8.4.

Třetí možností je využití Redis protokolu, dostupného ve verzi Redis 2.6-RC4 a 2.4.14. Tento přístup je nejméně časově náročný, protože předkládáme Redisu již předem připravená data, která jsou ve formátu

```
*<args><cr><lf>
$<len><cr><lf>
<arg0><cr><lf>
<arg1><cr><lf>
...
<argN><cr><lf>
```

Kde <cr> znamená „\r“ a <lf> znamená „\n“, <args> značí počet argumentů, <len> jejich délku a <argX> jejich názvy. Tento vzor může být matoucí, proto je lepší uvést příklad. Takto by vypadalo přidání našeho záznamu

```
*3\r\n$4\r\nZADD\r\n$3\r\nz:1\r\n$4\r\n9781\r\n
```

První číslo za znakem * značí, jaký počet prvků bude následovat (ZADD, z:1, 9781). Číslo za znakem $ říká, kolik znaků bude mít následující prvek (ZADD se skládá ze 4 znaků).

Soubor s těmito příkazy uložíme a pak pomocí „roury“ předáme Redisu ke zpracování.

```
cat soubor.txt | redis-cli –pipe
```

Je třeba ale dodat, že tento třetí způsob mi ze začátku nešel. Problémem se ukázalo být špatné interpretování „\r“ a „\n“ bashem. Vyřešil jsem to až přidáním znaku „$“ před celý výraz. Dalším možným řešením by mohla být změna shellu na „zsh“.

8.3 Naplnění MySQL daty

Naplnění MySQL je možné také třemi způsoby. Můžeme si buď spustit MySQL shell, který je součástí instalačního balíčku. Tento přístup je ale pomalý, protože by vyžadoval zadávání jednotlivých příkazů zvlášť. Dalším způsobem by bylo využití externího klienta, který se k databázi připojí

a vykoná sérii příkazů rychleji. Nejrychlejším způsobem je třetí metoda, která využívá funkce `LOAD DATA INFILE`, schopná zpracovat soubor a vytvořit z jeho obsahu záznamy v tabulkách.

První přístup, tedy klasické přidávání záznamů ze shellu MySQL, vypadá takto

```
mysql> INSERT INTO Log (id,param1,param2,
param3,param4,param5,param6) VALUES(1,9781,
0.758994068846388, 0.3977307, 1, 1961598753,
0.26655828612050203);
```

Tento způsob by ale vyžadoval, aby uživatel zadal na vstup ručně celkem 5000000 záznamů. Pro naplnění daty tedy použitelný není, pouze pro přidávání pár záznamů.

Druhým způsobem je využití klienta pro zpracování dat. Ten nám umožní se připojit k databázi a v cyklu vykonat potřebné příkazy.

```
public static void vytvorZaznamy()
{

    String url = "jdbc:mysql://localhost:3306/mySqlTest";
    String user = "root";
    String password = "root";

    try {
       Connection  con = DriverManager.getConnection(url, user, password);
       Statement pst = con.createStatement();
       String query;
       query = "CREATE TABLE IF NOT EXISTS Log(Id INT PRIMARY KEY "
               + "AUTO_INCREMENT, param1 INT, param2 DOUBLE, param3 DOUBLE,"
               + " param4 TINYINT, param5 INT, param6 DOUBLE ) "
               + "ENGINE=InnoDB";
       pst.execute(query);

        FileReader fr = new FileReader("dataSet.txt");
        BufferedReader br = new BufferedReader(fr);
        String s;
        while((s = br.readLine()) != null)
         {
            String[] zaznam = s.split(";");
            query="INSERT INTO Log(Id, param1, param2,param3,param4,"
                    + "param5,param6) VALUES("+zaznam[0]+","+zaznam[1]+","
                    +zaznam[2]+","+zaznam[3]+","+zaznam[4]+","+zaznam[5]+","
                    +zaznam[6]+")";
            pst.execute(query);
         }
       pst.close();
       con.close();
    }
    catch(Exception ex){}
}
```

Obrázek 15 Vytvoření záznamů v MySQL pomocí jazyka Java

Nejdříve je třeba se k databázi připojit. To proběhne pomocí třídy `Connection`. Dále se vytvoří tabulka Log s potřebnými atributy, pokud neexistuje. Následuje načtení dokumentu a jeho parsování. Pro tyto potřeby je použit identický soubor s daty jako u Redisu, proto jeho zpracování je velmi podobné. Jednotlivé hodnoty ze souboru jsou uloženy do pole a následně vloženy do příkazu na vytvoření záznamu.

Třetí možností je výše zmíněné funkce `LOAD DATA INFILE`. Pro její využití není třeba upravovat soubor s daty, dokáže načíst jak CSV soubor, tak i TXT. Zápis vypadá následovně:

```
mysql> LOAD DATA INFILE 'dataSet.txt' INTO TABLE Log
FIELDS TERMINATED BY ';';
```

Při zadání tohoto příkazu jsem se ovšem potýkal s hláškou, která oznamovala, že soubor neexistuje, přestože cesta byla správná. Problém se mi až podařilo vyřešit tím, že přihlašování k MySQL serveru probíhalo pomocí

```
mysql -u root -p --local-infile
```

Tedy rozšířeno o přepínač `local-infile`, který povoluje načítání ze souboru. Následně bylo ještě třeba rozšířit samotný příkaz na načtení záznamů o klíčové slovo `LOCAL`, takže celý zápis vypadal

```
mysql> LOAD DATA LOCAL INFILE 'dataSet.txt' INTO
TABLE Log FIELDS TERMINATED BY ';';
```

8.4 Měření

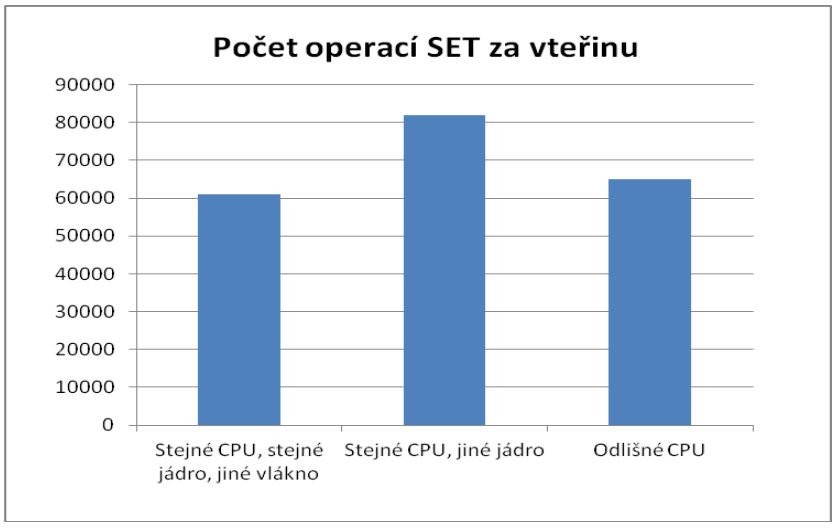
Po ukázkách, jaké jsou rozdíly mezi MySQL a Redisem, jak se s nimi pracuje, jak se skrz ně přistupuje k datům, přišel čas na jejich reálné otestování z hlediska výkonnosti. Do těchto testů jsem zahrnul přidávání záznamů, kdy je třeba data nejdříve parsovat v cyklu z dokumentu, dále přidávání záznamů při využití funkcí samotných technologií, které by měly rychlost navýšit, pak je třeba taky otestovat rychlost updatování většího množství záznamů a ve finále samotné vyhledávání. Pro férové hodnocení jsem se rozhodl nepozměňovat

konfigurační soubory, ale nechat nastavení od výrobce, aby se ukázala síla, popřípadě slabost, jednotlivých systémů, se kterou se bude muset potýkat uživatel, který si systém nainstaluje a bude chtít hned používat, aniž by si musel lámat hlavu se složitým nastavováním. U MySQL ještě dodám, že pro testy budu využívat engine InnoDB.

8.4.1 Faktory ovlivňující výkon Redisu

Existuje hned několik faktorů, které mají přímý dopad na výkon Redisu. Pokud neprovozujeme klienta a server na stejném stroji, bývá často kritickým prvkem šířka pásma a zpoždění v síti. Proto se doporučuje si nejdříve spočítat přibližnou náročnost Redisu na přenos dat a následně porovnat s možnostmi síťových prvků.

Procesor je dalším z důležitých faktorů. Poněvadž je Redis jednovláknová aplikace, nejvíce těží z rychlých procesorů, které mají velké cache paměti. Více jader můžeme zapojit do procesu tak, že spustíme Redis ve více instancích. Nejvýhodnější je vždy dát klienta i server na stejný procesor, ale jiné jádro. Mohou tak těžit z L3 cache. Na následujícím grafu lze vypozorovat rozdíly ve výkonu podle toho, jak rozmístíme klienta a server v procesoru.



Graf 1 Výkon Redisu v závislosti na rozmístění klienta a serveru (procesor Westmare)[17]

Pokud pracujeme s většími objekty v databázích (více jak 10KB), tak se vyplatí mít rychlé RAM paměti. Pokud ovšem pracujeme s menšími daty, má výkon RAM jen malý dopad na celkovou rychlost Redisu.

V případě, že bychom Redis testovali ve virtualizačním nástroji, můžeme narazit na nižší výkon. Testy ukazují, že u VMWaru je výkon téměř poloviční.

Počet připojených uživatelů je také důležitý faktor. V situaci, kdy je k němu připojených 60000 uživatelů, zvládne stále vykonávat 50000 příkazů za vteřinu. Čím méně uživatelů ale, tím rychleji dokáže Redis pracovat. Pokud budeme mít připojeno k serveru 100 uživatelů, bude přibližně o polovinu rychlejší, než když jich bude připojených 30000.[17]

8.4.2 Přidávání záznamů s parsováním

První měření se zaměřilo na přidávání záznamů, kdy je třeba před samotným přidáním ještě získat data pomocí parsování z dokumentu. Tento přístup byl popsán v kapitolách 8.2 a 8.3.

Redis

U Redisu jsem testoval 2 způsoby přidávání dat při parsování. První způsob je popsán v kapitole 8.2 na obrázku 15. Tento způsob má ovšem za následek to, že je třeba provést na našich 5000000 záznamů celkem 30000000 příkazů `ZADD`, který přidává záznam do sorted set, protože v našich datech neodpovídá jednomu klíči pouze jedna hodnota, ale rovnou 6. To ve výsledku prodlužuje dobu vykonávání příkazu. Proto jsem zkusil ještě jiný přístup. Jedis umožňuje kromě přidání samotné hodnoty i přidání mapy, která obsahuje hodnoty typu <Double, String>, kde Double je score a String je hodnota. Pro využití tohoto způsobu je tedy nejdříve nutné si data získaná z dokumentu, která odpovídají určitému klíči, uložit do mapy a následně je možné je jako celek uložit pomocí příkazu `ZADD` do databáze. Tento druhý přístup zkrátil dobu potřebnou pro vykonání příkazu čtyřnásobně. První způsob, ve kterém jsem přidával každou

hodnotu samostatně, zabral 17 minut. Ve druhém způsobu, který ukládal už pouze 5000000 záznamů, protože hodnoty odpovídající stejnému klíči byly uloženy společně v jedné mapě, bylo třeba jen 4 minut k vykonání příkazu.

MySQL

U MySQL jsem testoval způsob, který je uvedený v kapitole 8.3, na obrázku 16. Tento přístup je velmi podobný přístupu u Redisu. Zpracovává se stejný soubor, který se v cyklu parsuje a jednotlivé položky z dokumentu se vkládají do příkazu na vytvoření záznamu. Vykonávání příkazů ovšem muselo být v průběhu ručně zastaveno, protože se ukázalo, že průměrná rychlost vytváření je 27 záznamů za vteřinu, takže 5000000 záznamů by zabralo přibližně 51 hodin. Tento test jsem raději zopakoval, ale výsledek byl dokonce ještě horší, odpovídající 53 hodinám. Proto jsem se rozhodl test provést na jiném stroji. Výsledky byly ale velmi podobné. Poslední možností se ukázalo být přenastavení konfiguračního souboru. Ve výchozím nastavení nebyl totiž vhodný pro potřeby testování. Proti původnímu plánu jsem tedy musel změnit konfiguraci, abych dosáhl lepších výsledků. Konkrétně šlo o změny

`innodb_buffer_pool_size` na `2G`

`innodb_flush_log_at_trx_commit` na `2`

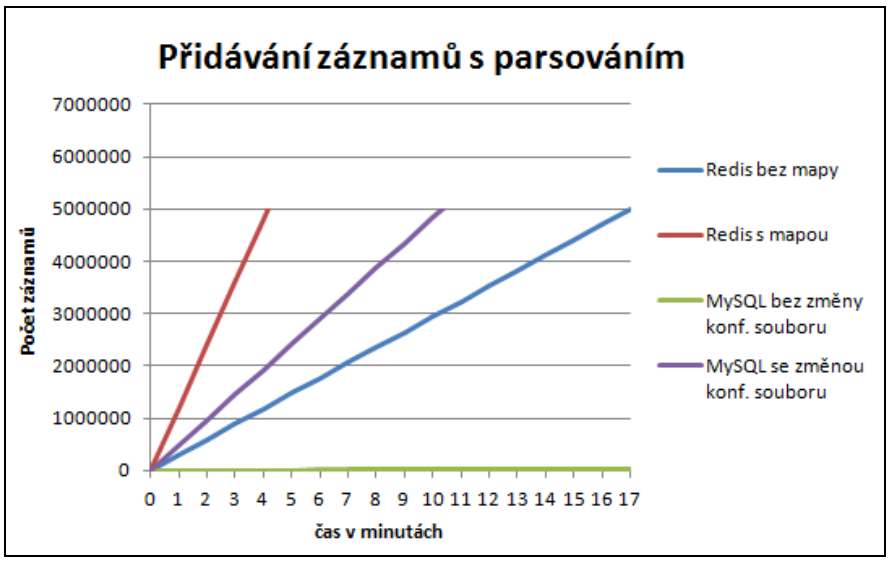
`innodb_thread_concurrency` na `8`

`key_buffer` na `384M`

`thread_cache_size` na `256`

Se změnou nastavení rapidně narostl výkon. Původních 51 hodin se tak zkrátilo na 10 minut a 36 vteřin.

Vyhodnocení



Graf 2 Rychlosti vytváření 5 milionů záznamů při parsování dokumentu

Z grafu vyplývají 2 věci. Za prvé, snížením počtu příkazů `ZADD` tím, že všechny hodnoty odpovídající určitému klíči uložíme do mapy, místo toho, abychom hodnoty ukládali do databáze jednotlivě, snížíme výrazně i dobu potřebnou pro vykonání příkazů. Za druhé, výkon příkazu `INSERT INTO`, který využívá MySQL pro přidávání záznamů, je velmi závislý na nastavení konfiguračního souboru. Pokud bych ho neupravil, MySQL by přidalo 5 milionů záznamů při parsování za 51 hodin, což je 180x pomalejší, než Redis. Při vhodné úpravě jsem ale byl schopný dosáhnout času 10 minut a 36 vteřin.

8.4.3 Přidávání záznamů bez parsování

Druhým testem bylo také přidávání záznamů, tentokrát ale stylem, kdy jsme předem nemuseli parsovat dokument, nýbrž jsme použili alternativních metod, které byly zmíněny v kapitolách 8.2 a 8.3.

Redis

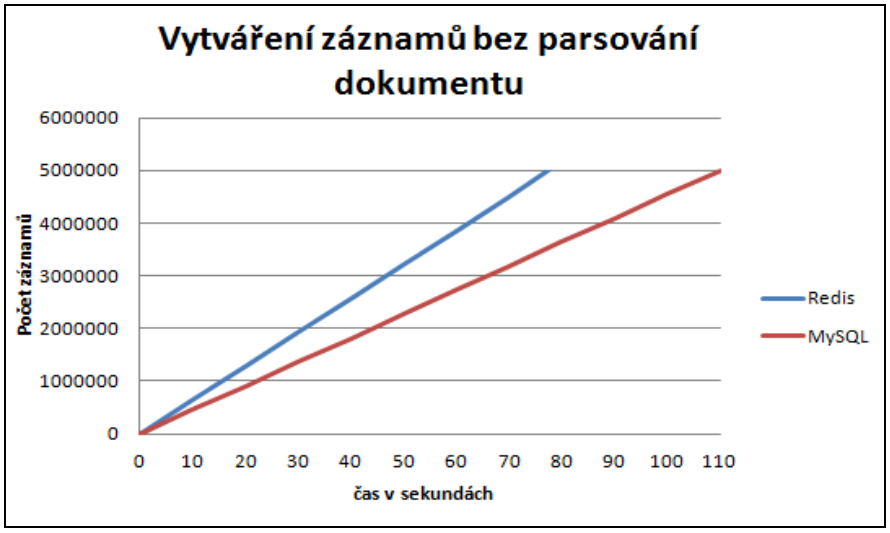
V případě Redisu jde o použití Redis protokolu, který umožňuje pomocí roury předat obsah souboru programu, který ho zpracuje. Obsah musí být ve

zvláštním formátu, který byl zmíněn v kapitole 8.2. Bylo proto třeba napsat novou metodu, která vytvořila požadovanou strukturu souboru, protože klasická verze, kde jsou hodnoty oddělené středníky, zde není použitelná. Výsledný textový soubor kvůli speciálním požadavkům Redis protokolu nabyl značně na objemu, z původní velikosti 350MB na 1.6GB a proto je taky působivé, že výsledná doba, potřebná pro zpracování příkazů, se zkrátila oproti předchozí rychlejší verzi 3.5x, na 1 minutu a 19 vteřin a oproti pomalejší až 14x.

MySQL

MySQL v tomto testu využívalo příkazu `LOAD DATA LOCAL INFILE`, který zpracovával výchozí verzi datasetu, bez potřeb úprav. Vzhledem k výsledkům v předchozím testu jsem k němu choval jistou skepsi, ale výsledek mě samotného překvapil. Dosažená 1 minuta a 50 vteřin bez upraveného konfiguračního souboru naznačuje, že `LOAD DATA LOCAL INFILE` je mocný nástroj, jak do MySQL databáze dostat větší množství dat. V tomto testu jsem zkusil aplikovat opět upravený konfigurační soubor, tentokrát ale bez známek rozdílu, proto není zanesen v následujícím grafu.

Vyhodnocení



Graf 3 Rychlost vytváření 5 milionů záznamů bez parsování dokumentu

Z grafu výše je vidět, že rozdíl v rychlosti vytváření nových záznamů, pokud použijeme funkcí na přidávání masivních objemů dat, je přijatelný. 1.19 u Redisu, při velikosti souboru 1.6GB, poukazuje na to, že je schopen velmi dobře pracovat se svým protokolem. MySQL, které vnitřně zpracovávalo soubor, kde jsou hodnoty oddělené středníky, dosáhlo 1 minuty a 50 vteřin. Vzhledem k tomu, že `LOAD DATA LOCAL INFILE` ještě umožňuje blíže nastavit, v jakém formátu se data v souboru nachází, což Redis protokol neumožňuje, poskytuje tak lepší načítání záznamů ze souboru.

Doteď jsem celou dobu mluvil u Redisu o příkazech `ZADD`, protože ukládáme data do sorted set. V následující tabulce jsem proto chtěl ukázat i rychlosti jiných příkazů, které se v Redisu využívají. Je třeba mít na paměti, že tyto čísla neznamenají přesnou rychlost, které bychom v naší aplikaci dosáhli. V průběhu testování se hodnoty lišily i o 20%. Význam jednotlivých příkazů můžete nalézt v tabulce 5 v kapitole 6.3.2. [17]

Příkaz	Rychlost
`SET`	106382.98 za vteřinu
`GET`	108695.65 za vteřinu
`INCR`	109890.11 za vteřinu
`LPUSH`	109890.11 za vteřinu
`SADD`	109890.11 za vteřinu
`LRANGE_100` (Prvních 100 prvků)	45454.55 za vteřinu
`LRANGE_300` (Prvních 300 prvků)	16750.42 za vteřinu
`LRANGE_500` (Prvních 500 prvků)	10775.86 za vteřinu
`LRANGE_600` (Prvních 600 prvků)	8156.61 za vteřinu

Tabulka 11 Rychlosti příkazů na testovaném stroji. Testované nástrojem Redis-benchmark

8.4.4 Update záznamů

Třetí měření se zaměřilo na schopnosti produktů měnit již stávající položky ve svých záznamech. Test spočíval v co nejrychlejší změně 100000 položek, přičemž se index záznamu i samotná hodnota záznamu náhodně generovaly, takže se v databázi nenacházely pod sebou. Jediné, co se v průběhu testu neměnilo, byl sloupec, který zůstával konstantní.

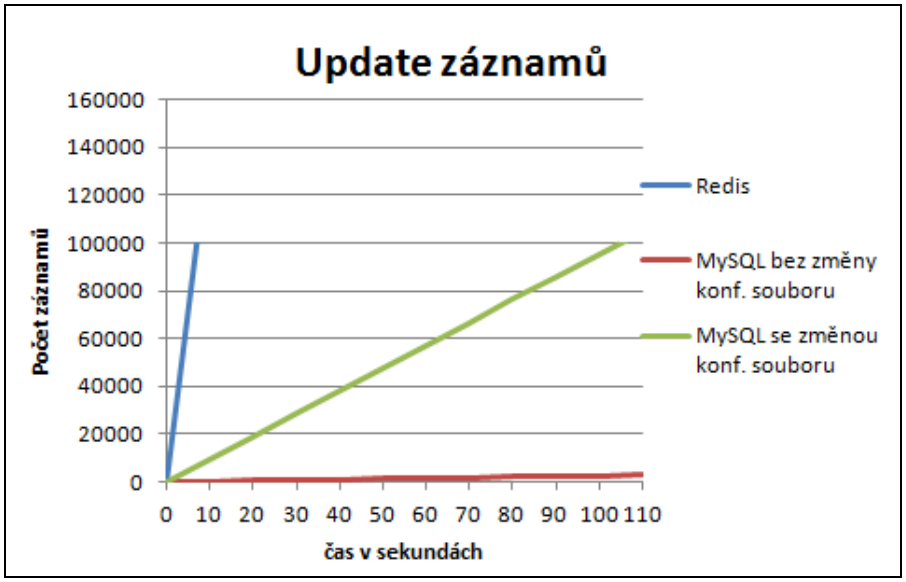
Redis

Poněvadž Redis neumožňuje přímočarou cestu pro změnu hodnoty, jak to můžeme znát z SQL systémů, bylo třeba zvolit jinou cestu. Standardně se tato operace změny provádí tak, že se přidá nová hodnota, která bude mít vyšší score než stávající. Tímto způsobem ale stará hodnota v záznamech zůstane i nadále uložená. Pokud bychom se chtěli přiblížit k updatu, tak jak ho známe odjinud, musíme starou hodnotu smazat. Pro tento způsob jsem se v testu rozhodl. Skládá se tedy ze dvou kroků: Smazání staré hodnoty a vytvoření nové. Tento přístup trval Redisu u 100000 cyklů 7 vteřin.

MySQL

U MySQL jsem zvolil typický způsob pro modifikaci záznamu, tj. `UPDATE Log SET param3=hodnota WHERE Id=hodnota`. Ten změní hodnotu záznamu, aniž by tam starou hodnotu nechal. Ukázalo se ale, že rychlost vykonávání tohoto příkazu je stejná jako v testu v kapitole 8.4.2, tj. 27 příkazů za vteřinu. Poněvadž jsem v testu modifikoval 100000 hodnot, vykonání tohoto požadavku by zabralo hodinu. Proto musel být předčasně opět ukončen. Následně jsem zkusil aplikovat upravený konfigurační soubor, zmíněný v kapitole 8.4.2. Výsledek se opět dostavil a 100 tisíc hodnot bylo přidáno za 1 minutu a 45 vteřin.

Vyhodnocení



Graf 4 Modifikace 100 tisíc záznamů

Vyhodnocení je velmi podobné testu, který proběhl v kapitole 8.4.2, přestože zde neprobíhalo žádné parsování. Redis zvládl změnit 100 tisíc hodnot za 7 vteřin, kdežto MySQL stejná práce trvala 61 minut. S upraveným konfiguračním souborem trvalo zpracování příkazů 1 minutu a 45 vteřin. Opět to poukazuje na velkou závislost na nastavení.

8.4.5 Vyhledávání záznamů

Posledním testem bylo vyhledávání záznamů. V něm měli Redis a MySQL za úkol v co nejkratším možném čase najít veškeré informace podle klíče, který se opět náhodně generoval. Tento cyklus se opakoval 100000x.

Redis

Pro vyhledávání záznamů v sorted set má Redis funkci `ZRANGE`. Ta přijímá celkem 3 parametry. Prvním je klíč, který nás zajímá. Druhým a třetím parametrem jsou hodnoty indexu, to znamená, které hodnoty nás u daného klíče zajímají. V našem případě jde o všechny. Zápis takovéhoto dotazu je

```
ZRANGE z:hodnota 0 -1
```

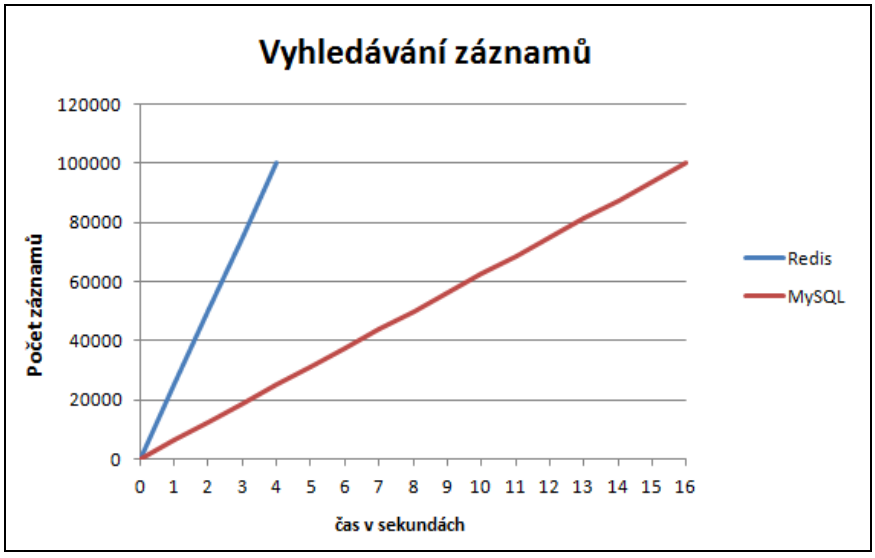
Proměnná „hodnota“ je v našem případě generována pomocí třídy Random. Následující nula říká, že nás zajímají atributy od prvního v řadě (indexy se značí od nuly) do posledního, což značí -1. Tímto číslem se vyhýbáme zadání přesného indexu posledního atributu. Samozřejmě bychom zde místo -1 mohli zadat 5 a výsledek by byl stejný (máme 6 atributů), pokud bychom ale měli záznam s více atributy, vypsalo by se pouze prvních 6. Nalezení všech hodnot u 100000 záznamů trvalo Redisu 4 vteřiny.

MySQL

U MySQL probíhalo vyhledávání snad vůbec nejznámějším příkazem, SELECTem. Jeho zápis vypadal `SELECT * FROM Log WHERE Id=hodnota.`

Hvězdičkou zde říkáme, že nás zajímají všechny hodnoty. Log nám říká, jakou tabulku budeme prohledávat a za klíčovým slovem `WHERE` specifikujeme, že hledané hodnoty musí být v záznamu, který má atribut `Id` s velikostí „hodnota“, jež se v průběhu vyhledávání měnila. Tento dotaz, který byl v cyklu prováděném 100000x, trval MySQL 16 vteřin, přičemž změna konfiguračního souboru nepřinesla rozdíl.

Vyhodnocení



Graf 5 Rychlost vyhledávání 100 tisíc záznamů

Z grafu je vidět, že v případě vyhledávání je na tom MySQL lépe, než v případě vkládání a modifikace záznamů. Přesto zaostává za Redisem o 400%. Zatímco se Redisu podařilo vyhledat 100000 záznamů za 4 vteřiny, tak MySQL to trvalo 16 vteřin.

8.5 Celkové vyhodnocení

Z této kapitoly si čtenář může udělat názor na to, jak rychlý je Redis a MySQL. U Redisu nebylo třeba v žádném z testů upravovat konfigurační soubor a přesto vyhrál ve všech měřeních. MySQL se naopak chovalo velmi podivně při pokusech o `INSERT` a `UPDATE` příkazy. Při výchozím nastavení konfiguračního souboru u těchto příkazů dosahoval pouze rychlosti 27 příkazů za vteřinu. Až po menší úpravě souboru se rychlost zvýšila. Přesto ne dostatečně na to, aby se v testech alespoň vyrovnala Redisu. V tomto testu jsem pracoval s databází, která měla pouhých 5 milionů záznamů. Dá se očekávat, že s narůstajícím počtem záznamů bude i lineárně růst čas potřebný pro vykonání příkazů. Pro tyto situace bych tedy doporučil použít jeden z NoSQL produktů. Případně nahrazení MySQL Community Server za MySQL Cluster. Jde o horizontálně škálovatelnou databázi, umožňující sharding, která se právě hodí na podobné situace. Je k dispozici jako open-source i pro komerční využití.

9 Závěr

Tato práce se zaobírala technologií NoSQL. V teoretické části jsem osvětlil samotný pojem NoSQL, co se pod ním skrývá. Vysvětlil jsem, jaké jsou rozdíly mezi SQL a NoSQL, včetně jejich výhod a nevýhod. Vybral jsem nejznámější zástupce za SQL a NoSQL a blíže s nimi čtenáře seznámil. Dále jsem také nastínil, jak se má situace okolo NoSQL ve světě, kde je nasazené na velkých serverech.

V praktické části jsem porovnal vybrané NoSQL produkty podle toho, jak je dostupná jejich dokumentace, jaké operační systémy a programovací jazyky podporují. Za zástupce NoSQL a SQL jsem vybral Redis a MySQL a ukázal, jak produkty nainstalovat, nakonfigurovat a pracovat s nimi. Naplnil jsem je 5000000 záznamy, přičemž oba systémy měly stejný zdroj dat. Následně jsem si v jazyce java vytvořil metody na vkládání, upravování a vyhledávání záznamů. Oba systémy jsem otestoval a výsledky zanesl do grafů.

Na závěr bych rád řekl svůj názor na využitelnost NoSQL. Dokážu si představit, že bych některý z NoSQL produktů provozoval v případě, že budu mít desítky až stovky milionů záznamů nebo by se struktura dat často měnila. Tehdy by výhody NoSQL převažovaly. V ostatních případech bych se spíše držel SQL systémů, jejichž jazyk je nádherně mocný a jednoduchý. Co se týče výběru NoSQL databáze, tak bych si v případě přechodu z relačních databází vybral MongoDB, které je výborně zpracované. Zároveň se asi nejlépe hodí pro seznamování s NoSQL, protože má dobře zpracovanou dokumentaci a materiály k němu jsou nejsnáze dostupné.

Stanovené cíle bakalářské práce byly splněny.

10 Terminologický slovník

Termín	Význam
SŘBD	Jde o softwarové vybavení, které zajišťuje práci s databází
SQL	Dotazovací jazyk pro práci s daty v relačních databázích
NoSQL	Databáze, které nejsou primárně postaveny na tabulkách a často nevyužívají SQL jazyk.

Primární klíč	Jedinečný identifikátor pro záznam v relační databázi
Cizí klíč	Identifikátor záznamu, díky němuž je možné propojit tabulky v relační databázi
Normální formy	Sada pravidel, které by měly tabulky splňovat
ACID	Vlastnosti, které by měly transakce splňovat (atomicita, konzistence, izolovanost, trvalost)
Shard	Části v clusteru, které drží stejná data
Cluster	Skupina spojených počítačů za účelem spolupráce a lepších výsledků
CAP	Teorie, která říká, že je možné zvolit pouze 2 ze 3 vlastností u NoSQL databází (konzistence, dostupnost, tolerance k rozdělení)

11 Použitá literatura

[1] *Proffesional NoSQL*. Indianapolis, Indiana: John Wiley & Sons, Inc., 2011. ISBN 978-0-470-94224-6.

[2] Normalizace relačních databází. *Programujte.com - web o programování, webdesignu, počítačové grafice, databázích, elektrotechnice a designu* [online]. 23. 7. 2008 [cit. 2013-04-05]. Dostupné z: http://programujte.com/clanek/2008071900-normalizace-relacnich-databazi/

[3] Microsoft SQL Server. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-04-05]. Dostupné z: http://en.wikipedia.org/wiki/Microsoft_SQL_Server#Architecture

[4] Srovnání databázových serverů. *Linux E X P R E S* [online]. 7. 3. 2007 [cit. 2013-04-05]. Dostupné z: http://www.linuxexpres.cz/software/srovnani-databazovych-serveru

[5] PostgreSQL: Documentation: 9.2: A Brief History of PostgreSQL. *PostgreSQL: The world's most advanced open source database* [online]. © 1996-2013 [cit. 2013-04-05]. Dostupné z: http://www.postgresql.org/docs/current/static/history.html

[6] SQLite - ultra lehké sql. *Root.cz - informace nejen ze světa Linuxu* [online]. 4. 8. 2003 [cit. 2013-04-05]. Dostupné z: http://www.root.cz/clanky/sqlite-ultra-lehke-sql/

[7]MySQL Editions. *MySQL :: The world's most popular open source database* [online]. © 2013 [cit. 2013-04-05]. Dostupné z: http://www.mysql.com/products/

[8] Supported Platforms: MySQL Database. *MySQL :: The world's most popular open source database* [online]. © 2013 [cit. 2013-04-05]. Dostupné z: http://www.mysql.com/support/supportedplatforms/database.html

[9] Uložené procedury. *Linux Software* [online]. 21. 10. 2005 [cit. 2013-04-05]. Dostupné z: http://www.linuxsoft.cz/article.php?id_article=1003

[10] *Dostupné škálovateľné riešenia pre spracovanie veľkého objemu dát a dátové sklady* [online]. 2011[cit. 2013-04-05]. Dostupné z: http://laclavik.sk/publications/datakon_final_2011.pdf

[11] *Srovnání distribuovaných "NoSQL" databází s důrazem na výkon a škálovatelnost* [online]. Praha, 2012 [cit. 2013-04-05]. Dostupné z:

http://isis.vse.cz/zp/111109. DIPLOMOVÁ PRÁCE. Vysoká škola ekonomická v Praze.

[12] *NoSQL* [online]. 2011[cit. 2013-04-05]. Dostupné z: http://static.usenix.org/publications/login/2011-10/openpdfs/Burd.pdf

[13] How Twitter Uses NoSQL. *Web Apps, Web Technology Trends, Social Networking and Social Media ReadWrite* [online]. 2.1. 2011 [cit. 2013-04-05]. Dostupné z: http://readwrite.com/2011/01/02/how-twitter-uses-nosql

[14] Redis Quick Start. *Redis* [online]. 2012 [cit. 2013-04-05]. Dostupné z: http://redis.io/topics/quickstart

[15] Command reference – Redis. *Redis* [online]. 2012 [cit. 2013-04-05]. Dostupné z: http://redis.io/commands

[16] Databáze MariaDB válcuje MySQL. *Root.cz - informace nejen ze světa Linuxu* [online]. 21. 12. 2012 [cit. 2013-04-05]. Dostupné z: http://www.root.cz/clanky/databaze-mariadb-valcuje-mysql/

[17] How fast is Redis?. *Redis* [online]. 2012 [cit. 2013-04-05]. Dostupné z: http://redis.io/topics/benchmarks