Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Správce databází pro vybraný NoSQL databázový systém

Plzeň 2014 Barbora Staffová

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracovala samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 11. května 2014

Barbora Staffová

Abstract

This thesis deals with the creation of database manager for NoSQL database system Redis. In the first part of the document the basic database terminology is described. Then the current NoSQL database systems are introduced and compared with relational database systems. In the second part there is presented the analysis of solutions for the resulting program NoSqlManager. The next part describes the creation of the application itself and its functions. The applicability of the final database manager was tested in a distributed environment.

Tato diplomová práce se zabývá tvorbou správce pro NoSQL databázový systém Redis. V první části dokumentu jsou vysvětleny základní pojmy o databázích. Dále jsou představeny současné NoSQL databázové systémy a jejich srovnání s relačními databázovými systémy. V druhé části je popsána analýza řešení pro výsledný program NoSqlManager. Další část popisuje vytvoření samotného správce databáze a jeho funkce. Použitelnost výsledného správce databáze byla otestována v distribuovaném prostředí.

Obsah

Seznam obrázků

1 Úvod

V dnešní době díky nárůstu mobilních a elektronických zařízení stoupá i množství uložených dat a nároky na rychlost jejich zpracování. Stále častěji se v reálné praxi objevuje situace, kdy dnes hojně používané relační databáze přestávají dostačovat svým datovým modelem. Díky tomu se na scéně objevily NoSQL databáze, které jsou schopny zpracovat velké objemy dat (tzv. Big data) různých struktur. Nejedná se však o náhradu stávajících relačních databází. Obě tato řešení mají stále svá specifická uplatnění. Relační databáze vycházejí z klasického pojetí datových struktur, tzn., že vazby a struktura dat je přesně specifikovaná. Na rozdíl od toho NoSQL databáze nemají přesně specifikovanou strukturu. Vývoj NoSQL databází šel několika směry, které umožnily vzniku různých typů databází. Tyto směry jsou reprezentovány databázemi key/value, sloupcovými, dokumentovými a grafovými databázemi. NoSQL databáze jsou hojně využívány především u sociálních sítí, jako je Facebook nebo Twitter, kde se každým dnem enormně zvyšuje počet uchovávaných dat.

Cílem této diplomové práce je zanalyzovat současné druhy NoSQL databázových systémů, přičemž výsledné poznatky porovnat s relačními databázovými systémy a v návaznosti na to vybrat vhodnou NoSQL databázi. V tomto případě byla zvolena na základě analýzy problému databáze Redis, která je typu key/value.

Dalším cílem bylo praktické využití zmíněné databáze pro navržení správce, který bude umožňovat uživatelsky příjemnou práci s daty uloženými v uvedené databázi a následně na to implementovat řešení s ohledem na poskytovaná API (Application Programming Interface) systému a v závěru pak výsledného správce otestovat a zhodnotit.

Diplomová práce je rozdělena do kapitol. První kapitola se zabývá teoretickým úvodem do problematiky databází. V druhé kapitole jsou podrobně popsány relační databáze a jejich vlastnosti. V třetí kapitole je detailně probrána problematika NoSQL databází. Jsou zde uvedeny hlavní představitelé jednotlivých typů NoSQL databází. Další kapitola se zabývá porovnáním relačních a NoSQL databází. Na to navazují kapitoly, v kterých dochází k popisu realizace tvorby správce NoSQL databáze. Na závěr práce je uvedena kapitola, která zahrnuje výsledky testování napsaného správce.

V dodatku je přiložena uživatelská dokumentace, graf závislostí jednotlivých jmenných prostoru a popis obsahu přiloženého DVD.

2 Základní pojmy a teorie

V současnosti se hojně pracuje s velkými objemy informací, jejichž zpracování a ukládání se stává významnou otázkou především v oblasti informačních technologií. Informace vznikají zpracováním a strukturováním surových dat, která dávají určitému jedinci nebo organizaci význam. [Con09]

Organizované soubory dat, které se používají k modelování různých organizačních struktur nebo procesů, se nazývají databáze.

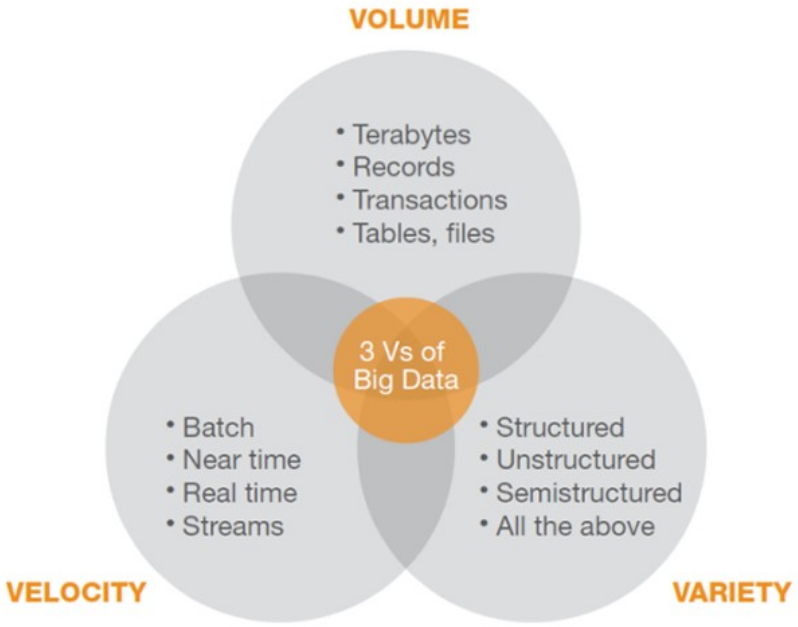
Databáze bývá definována jako sebe popisující kolekce integrovaných záznamů. Díky sebe popisujícímu charakteru databáze je zajištěna nezávislost dat. Tudíž změnou struktur v databázi nejsou ovlivněny existující databázové aplikace, pokud se jich změna přímo netýká. [Her06]

2.1 Big data

V posledních letech se díky rozšíření webu, mobilních aplikacích a dalších nových technologiích výrazně zvětšilo množství uložených dat, která jsou nestrukturovaná a distribuovaná. S tím přichází i výraz big data. Tento pojmem si lze představit jako soubory dat, které jsou tak rozsáhlé, že se nedají spravovat, zachycovat a zpracovávat normálními softwarovými prostředky v rozumném čase. Velikost dat je chápana nejenom z hlediska objemu dat, ale také z hlediska rychlosti jejich tvorby a přenosu. Pro efektivní práci s big daty se vyžadují nové technologie, které budou zpracovávat data v přijatelném čase. Často se v souvislosti s big daty mluví o trojrozměrnosti velikosti a růstu dat, známé také jako 3V, viz obrázek 2.1.

- Objem (volume) – při provozu firem narůstá množství dat různých typů exponenciálně.
- Typ (variety) – big data jsou různých typů, mimo obvyklých strukturovaných dat jsou zpracovávána také nestrukturovaná data, například textové soubory, data ze senzorů, video soubory a data z logů.
- Rychlost (velocity) – velká rychlost vznikajících dat a potřeba jejich analýzy v reálném čase.

Bývají uváděná i další „V“ jako věrohodnost (veracity) nebo variabilita (variability)[1]. Ve výsledku lze však shrnout big data tak, že přibývá mnoho různých zařízení a systémů, kde data vznikají. Také roste množství dat a zvyšují se nároky na jejich zpracování. [Dol11]



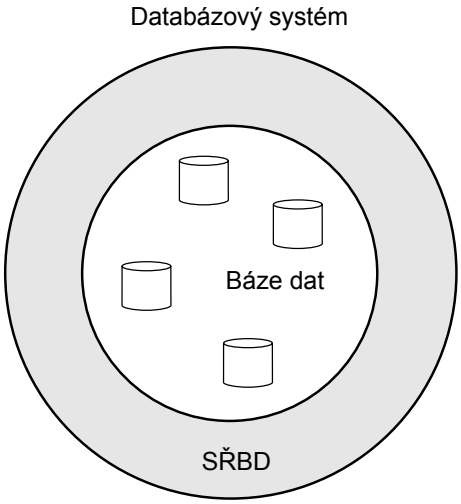
Obrázek 2.1: Big data a 3V. [Lac12]

2.2 SŘBD

Pojmem databáze bývá často zaměňován s databázovým systém. Databázový systém se skládá z báze dat, neboli databáze, a ze systému řízení báze dat (SŘBD, anglicky též DBMS), viz obrázek 2.2. SŘBD je softwarové vybavení, které umožňuje uživateli definovat, vytvářet, udržovat a řídit databázi. SŘBD dovoluje uživatelům aktualizovat, vkládat, mazat a vyvolávat data z databáze. Součástí SŘBD je také možnost dotazování na data pomocí dotazovacích jazyků jako je např. SQL (Structured Query Language). [Con09]

Před vznikem SŘBD se pro počítače dodávalo všeobecné programové vybavení, které mělo za úkol plnit některé funkce SŘBD. Jednalo se o systémy na řízení souborů (file management systems). Za kritérium, které rozlišuje SŘBD od jiných systémů

[1] `http://www.adastra.cz/co-jsou-big-data`



Obrázek 2.2: Báze dat a systém řízení báze dat (SŘBD) tvořící databázový systém. [Sch88]

s podobnou funkcionalitou, se považuje to, zda si systém vede a obsahuje katalog dat, do kterého si ukládá strukturu souborů, s kterými pracuje. Takováto charakteristika SŘBD umožňuje dosáhnout nezávislosti programů na datech.[Sch88]

Pokud je SŘBD bez podpory zobrazování dat na obrazovce počítače či v tištěné podobě, je označována jako databázový stroj. Zobrazování dat zajišťuje uživatelská aplikace. Mezi některé SŘBD patří např. Oracle, MS SQL Server, Sybase, Informix nebo freeware programy mSQL, MySQL a PostgreSQL [Kos99].

2.3 Databázové modely

Systémy pro řízení báze dat lze rozdělit do několika typů, které se označují jako modely. Databázové modely určují logickou strukturu databáze a také jakým způsobem budou data uložena, organizována a zpracována.

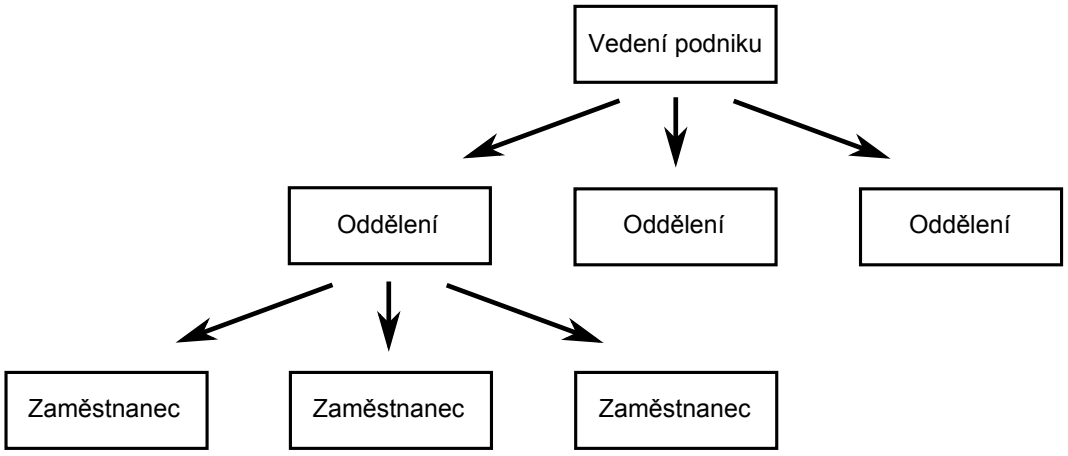
2.3.1 Otevřené soubory

Obyčejné soubory operačního systému jsou označovány jako otevřené soubory (flat files). To znamená, že záznamy těchto souborů nenesou žádné doplňující informace, které by cílové aplikaci předávaly strukturu souborů nebo vztahy mezi záznamy.

Informace o struktuře dat musí být součástí aplikace, která se souborem pracuje. Databázový systém má, na rozdíl od operačního systému, o těchto souborech k dispozici metadata, díky jimž může provádět různé převody mezi otevřenými soubory ve fyzické vrstvě. Metadata, neboli data o datech, jsou v tomto případě použita k označení informací, které popisují, jaká data jsou uložena v databázi a vztahy mezi nimi. Otevřené soubory zde byly již před vznikem databází. První databázové systémy se vyvinuly právě z nich. [Opp06]

2.3.2 Hierarchický databázový model

Hierarchický model, který je zobrazen na obrázku 2.3, je nejstarší používaný model. Poprvé byl použit v roce 1968 společností IBM u systému IMS. Základem tohoto modelu je stromová struktura vycházející z jednoho kořene. Kořen (jak je to chápáno v teorii grafu) je jediný uzel, který nemá rodičovský uzel. Jednotlivé uzly stromu jsou propojené do hierarchie rodič-potomek, kde jeden rodič může mít více potomků, ale každý potomek má jen jednoho rodiče, neboli uzel vyšší úrovně se vztahuje k více uzlům na nižší úrovni. Pokud z uzlu nevychází další větev, je označen jako list. V této struktuře jsou dva druhy vztahů mezi položkami, rodičovské a sourozenecké. Kromě listu může mít každý uzel libovolný počet potomků. Díky tomu se v hierarchickém modelu dobře definují vztahy 1:n, jeden ku mnoha. [Kal12]



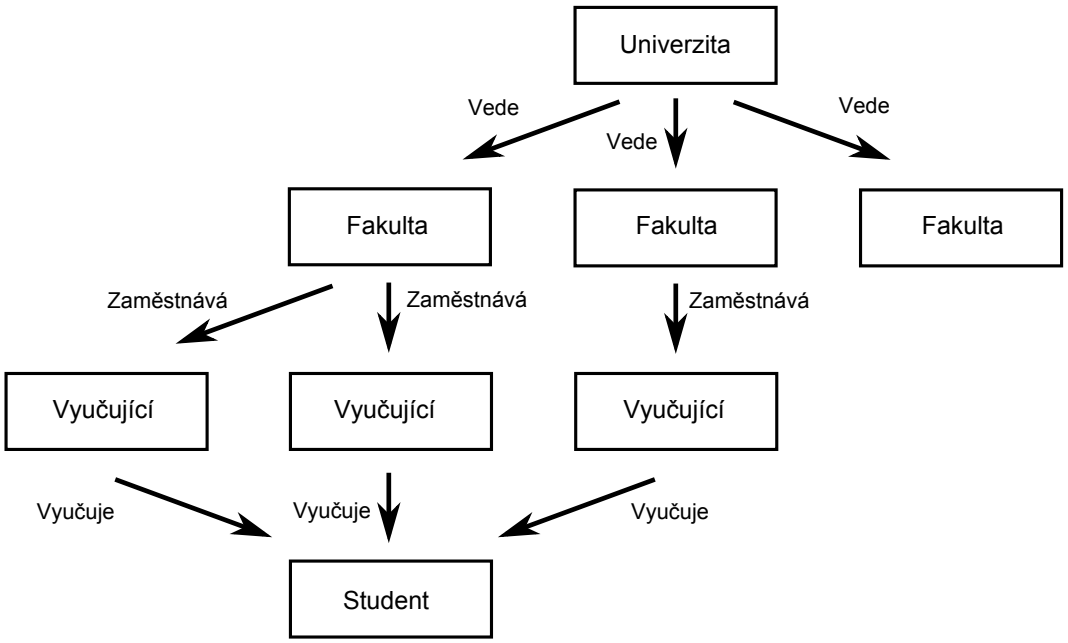
Obrázek 2.3: Hierarchický model databáze.

Hierarchické databáze byly používané zejména pro ukládání dat v 70. letech na sálových počítačích. Výhodou hierarchického modelu byl přímý a rychlý přístup k datům a také možnost přidávání nových položek. Mezi nevýhody patří složitá

správa struktura databáze. Při úpravě vztahů rodič-potomek se musí složitě předělávat celá struktura. Největší nevýhodou je však tvorba vztahů m:n, mnoha k mnohým. [Her06]

2.3.3 Síťový databázový model

Síťová databáze byla vyvinuta jako možnost pro vyřešení problémů vznikajících u hierarchické databáze. Tento model umožňuje popisovat v databázi vztahy m:n, mnoha k mnohým. Struktura síťové databáze je popsána pomocí pojmů uzel a množinová struktura, kde uzel představuje soubor záznamů a množinová struktura zastupuje a zřizuje vztah. Vztahy mezi daty jsou pojmenovány, tudíž je v databázi možný přechod z jednoho záznamu na jiný přes konkrétní relaci. Rozdíl oproti hierarchické konstrukci rodič-potomek je takový, že jeden uzel je definovaný jako vlastník a druhý jako člen. Díky množinové struktuře je podporován vztah 1:n. Vlastník může být v relaci k jednomu nebo více záznamům v uzlu člen. Jeden záznam v uzlu člen je však ve vztahu pouze k jednomu záznamu v uzlu typu vlastník. Záznam v uzlu typu člen také nemůže existovat, pokud by nebyl ve vztahu k záznamu v odpovídajícím uzlu typu vlastník. Příklad síťového modelu je na obrázku 2.4. [Her06]



Obrázek 2.4: Síťový model databáze.

Síťový model přináší především vyšší flexibilitu, snadné přidávání nových zá-

znamů, omezení některých duplicitních záznamů a rychlé vyhledávání dat. Přes některá zlepšení oproti hierarchickému modelu se síťový model setkává s nutností správného návrhu prvotní struktury databáze.

2.3.4 Relační databázový model

Roku 1969 zveřejnil Dr. E. F. Codd článek, který definoval model databází založený na matematickém pojmu relačních množin. Tento model se stal nejrozšířenějším modelem používaným při správě databází. Data se v relačních databázích ukládají ve vztazích, které uživatelé vidí jako tabulky. Vztahy jsou složeny z tzv. n-tic (řádků) a atributů (sloupců). Každý řádek je v tabulce identifikován sloupcem obsahující jedinečnou hodnotu. Díky tomu nemusí uživatel, na rozdíl od hierarchického a síťového modelu, znát přesné fyzické umístění záznamu, pokud s ním chce pracovat. Více o relačních databázích je uvedeno v kapitole 3.

2.4 ACID

Zkratka ACID (Atomicity, Consistency, Isolation, Durability) označuje čtyři základní vlastnosti, které musí splňovat transakce. Databázová transakce je soubor příkazů, který zjišťuje převod z jednoho konzistentního stavu do druhého. Pokud se vyskytne chyba, musí být možné vrátit celou operaci do původního stavu.

- Atomicita (Atomicity) – Každá transakce je atomická (nedělitelná), buď proběhnou všechny operace v transakci, nebo žádná. Pokud selže jedna část, selže celá transakce a stav databáze se vrátí do původního stavu.
- Konzistence (Consistency) – Data se před i po provedení transakce nachází v konzistentním stavu. Veškerá data zapsaná do databáze musí být platná podle stanovených pravidel, transakce nesmí porušit žádné integritní omezení.
- Izolace (Isolation) – Při paralelním zpracování jsou jednotlivé transakce od sebe izolovány a navzájem se neovlivňují.
- Trvalost (Durability) – Změny po úspěšné transakci jsou v databázi uloženy natrvalo.

Systém, který splňuje všechny vlastnosti ACID, se označuje za „silně konzistentní“. Takovéto databázové systémy jsou potřeba v některých případech, jako jsou např.

systémy v obchodech nebo v bankách. Databáze, které zcela nepodporují vlastnosti ACID, jsou nazývány „eventuálně (případně) konzistentní“. Data v takovýchto databázích nejsou konzistentní v libovolném čase, ale konzistence je dosaženo postupně. Přístup eventuálně konzistentních databází umožňuje získat výhodu lepší horizontální škálovatelnosti. Toho využívají především NoSQL databáze. [Pok97][Tiw11]

3 Relační databáze

Relační databáze je normalizovaný systém, který je založen na relačním modelu dat a je řízen pomocí SŘBD. Základem každé relační databáze je tabulka neboli relace. Databáze je následně tvořena kolekcí tabulek, které jsou navzájem propojeny. Svou oblibu mezi programátory získaly především kvůli své jednoduchosti. [Skř02]

3.1 Popis

Relace je reprezentována jako tabulka, kde řádky tabulky reprezentují datové n-tice a sloupce odpovídají atributům. I když se atributy objeví v libovolném pořadí, bude se jednat o stejnou relaci se stejným významem. Tabulka má jedinečné jméno, aby byla odlišitelná od ostatních tabulek v dané databázi. Buňky tabulky musí obsahovat pouze jednu hodnotu (příslušná tabulka je tak normalizována do první normální formy). Každý sloupec má jedinečný název a určitý datový typ podle dat, jež mají být uložena. Řádky tabulky musí mít jedinečný identifikátor, což se řeší pomocí klíče. Primární klíč je sloupec nebo množina sloupců, který jedinečně určuje řádek v tabulce. Cizí klíč je také množina sloupců, která nese stejnou hodnotu jako primární klíč ale v jiné tabulce. [Con09]

S objevem objektové orientace přišlo vylepšení relačních databází. Dochází k rozšíření o nový typ, objekt, a o podporu ukládání a práce s ním. Vznikají tak objektově relační SŘBD (ORDBMS), které kombinují vlastnosti relačních SŘBD, jako je třeba dotazovací jazyk SQL, a vlastnosti objektově orientovaných SŘBD (OODBMS). Tento model má mnoho výhod, ale stále není příliš rozšířen. [Opp06]

3.2 Integritní omezení

Relační model dat určuje pouze strukturu dat. Pro praktické použití je ale nutné zajistit, aby se v relaci vyskytovala pouze logicky správná data. Integritní omezení jsou pravidla, která zabrání vložení nesprávných data a ztrátě nebo poškození stávajících údajů při práci s databází.

Entitní integrita zajišťuje jednoznačnost řádku v tabulce, neboli úplnost primárního klíče. Dalším pravidlem je referenční integrita, která se vztahuje k cizím klíčům. Existuje-li v tabulce cizí klíč, musí jeho hodnota odpovídat primárnímu klíči

v jiné tabulce. Databáze, která vyhovuje zadaným integritním omezením, se označuje za konzistentní. [Con09]

3.3 Normální formy

Důvodem k zavedení normalizace vedly problémy, jež byly označovány jako anomálie, nenormalizovaných relací, které se projevovaly při aktualizaci dat. Normalizace je dekompozice relace, která vznikla kvůli jednodušší práci s daty, jejich lepší manipulaci, zabránění redundance dat a lepší konzistenci. Počet normalizačních úrovní, označovaných jako normální formy, je šest. [Opp06]

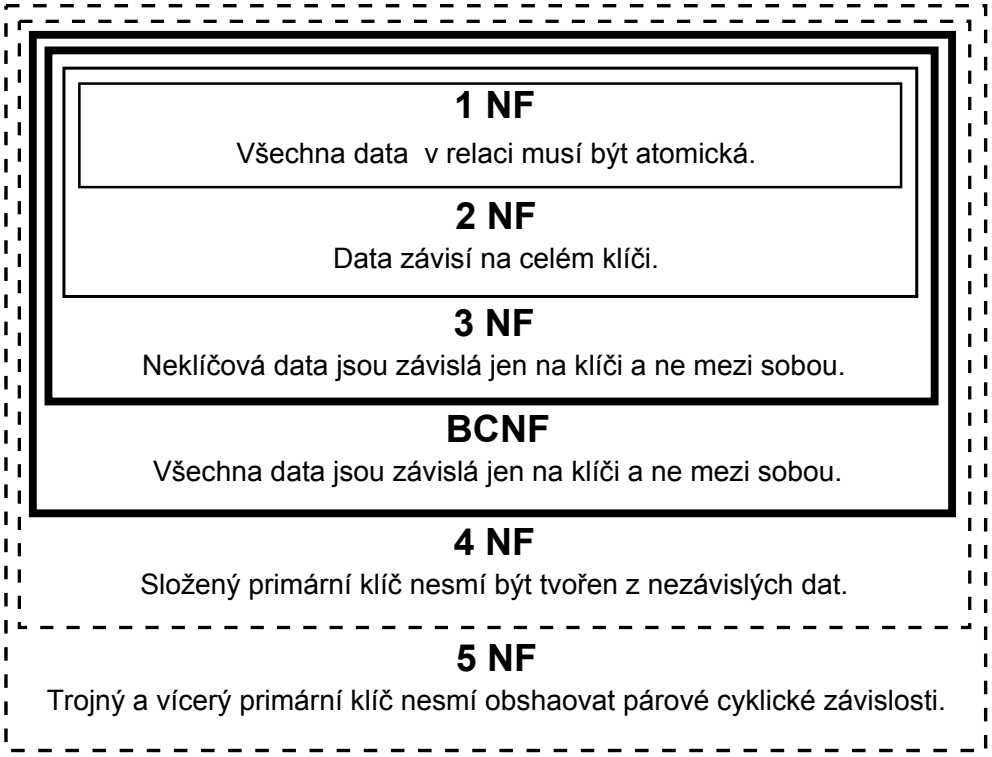
Normální formy jsou pravidla, podle kterých by se měly navrhovat tabulky a jejich vzájemné relace. Jsou označeny jako 1NF, 2NF, 3NF, BCNF, 4NF a 5NF. Formy směřují postupně od nižších k vyšším, kdy každá vyšší normální forma obsahuje svá vlastní pravidla a podmínky forem předchozích, viz obrázek 3.1. [Kul08]

Relace je v první normální formě (1NF), pokud jsou data atomická. To znamená, že by již dále neměla být dělitelná, neboli neobsahuje žádné atributy s násobnými hodnotami. V druhé normální formě (2NF) se relace nachází, když je v 1NF a každý neklíčový atribut je plně funkčně závislý na celém primárním klíči relace. Relace se nachází v třetí normální formě (3NF), pokud je v 2NF a každý neklíčový atribut není tranzitivně závislý na primárním klíči, neboli neexistuje závislost mezi dvěma neklíčovými atributy.

Vedle 3NF existuje Boyce-Coddova normální forma (BCNF), která je silnější formou 3NF. Relace je v BCNF, pokud je v 3NF a v relaci neexistuje žádný určující atribut, který by byl primárním nebo kandidátním klíčem. Tzn., že žádný neklíčový atribut nesmí určovat hodnotu žádného jiného atributu, a to ani atributů, které jsou součástí definice primárního klíče. Každá relace, která je v BCNF je vždy ve 3NF. Opačně však 3NF nemusí být v BCNF, jestliže nastane situace, kdy v relaci je více kandidátních klíčů, všechny tyto kandidátní klíče jsou složené a existuje atribut, který je pro všechny kandidátní klíče společný.

Z praktického hlediska postačí většinou transformace relaci do BCNF. Je však popsán převod relací dále do čtvrté a páté normální formy. Opomíjí se ale často proto, že představují poměrně výjimečné situace a je celkem obtížné je porušit.

Ve čtvrté normální formě (4NF) je relace, pokud je v BCNF a všechny vícehodnotové závislost jsou zároveň funkčními závislostmi z kandidátních klíčů. V poslední páté normální formě (5NF) se relace nachází, jeli ve 4NF a nemůže-li být dále bez-



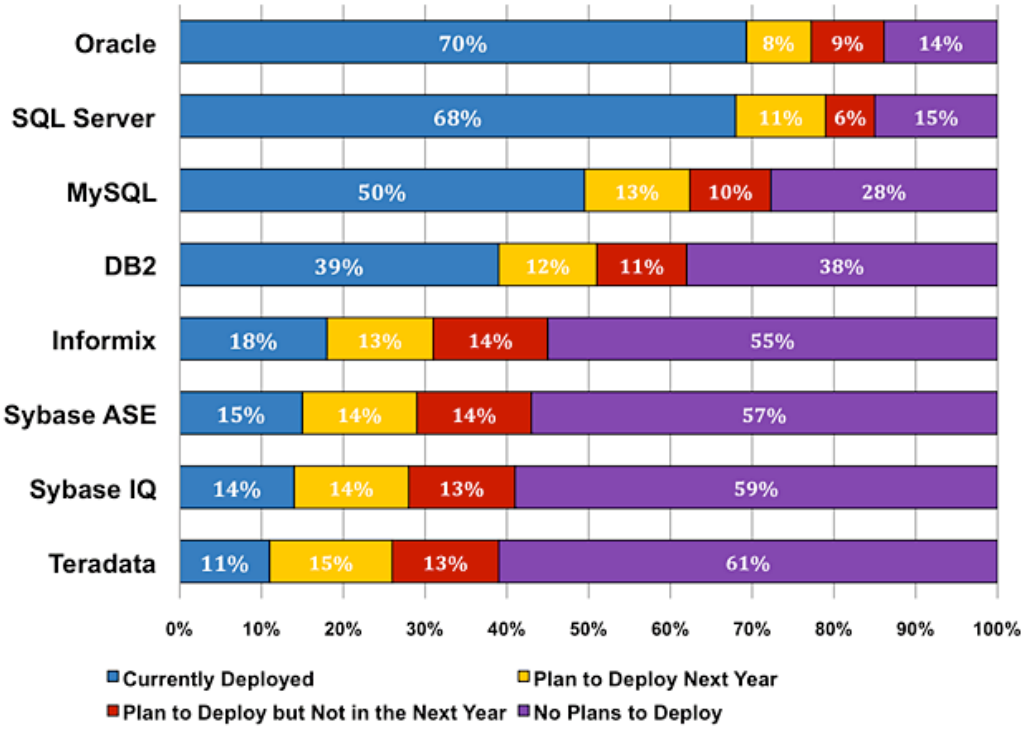
Obrázek 3.1: Normální formy. [Kul08]

ztrátově rozložena. [Kal12] [Opp06] [Norm07]

3.4 SQL

SQL (Structured Querry Language, v překladu strukturovaný dotazovací jazyk) je standardizovaný dotazovací jazyk. Používá se především pro práci s daty v relačních databázích. SQL vznikl z projektu SEQUEL v 70. letech v laboratořích firmy IBM. Cílem bylo vytvořit standardní metodu přístupu k datům, která by byla nezávislá na dalších vývojových nástrojích.

Jazyk SQL je neprocedurální. Zadává, jaké informace se vyžadují, nikoliv způsob, jak je získat. Tento způsob využití je označován jako dynamické SQL. Také se ale může SQL použít v aplikaci, která je napsána procedurálním jazykem. Potom se nazývá jako vnořené SQL, které poskytuje přístup k databázi nezávisle na jazyku, ve kterém je napsána zbývající část aplikace.



Obrázek 3.2: Počet nainstalovaných a nasazených databázových systémů podle společnosti Gartner v roce 2008.[MaS08]

Skládá se z několika skupin. První je jazyk pro definici dat DDL (Data Definition Language), který určuje strukturu databáze (např. příkazy CREATE, ALTER, DROP). Druhým jazykem je DML (Data Manipulation Language), neboli jazyk pro manipulaci dat, který slouží pro získávání dat z databáze a jejich aktualizaci. Mezi některé příkazy DML jazyka patří např. SELECT, INSERT, UPDATE a DELETE. Dalšími jazyky jsou DCL (Data Control Language), jazyk pro řízení dat, který slouží pro správu uživatelských rolí a práv, a TCL (Transaction Control Language), jazyk pro řízení transakcí. [Con09][Skř00]

3.5 Hlavní představitelé

V současné době patří k nejvíce rozšířeným a nejpropracovanějším SŘBD právě relační databázové systémy (RDBMS). V další části jsou uvedeny základní informace o nejznámějších zástupcích. Na obrázku 3.2 je zobrazen přehled nejpoužívanějších databázových systémů.

MySQL

Databázový systém MySQL[1] vlastněný společností Oracle Corporation je poskytován pod dvěma licencemi. Základní verze je pod bezplatnou licencí GPL (General Public License). Komerční verze je poskytována pod placenou licencí a obsahuje rozšířené funkce a nástroje. Díky své multiplatformnosti, vysokému výkonu a nízkým finančním nákladům je tento systém často využíván především na serverech pro potřeby webových aplikací. Společně s operačním systémem Linux, skriptovacím programovacím jazykem PHP a webovým serverem Apache tvoří tzv. LAMP server. Pro správu MySQL databází se používá open-source nástroj napsaný v jazyce PHP, PhpMyAdmin, který umožňuje kompletní správu systému pomocí webového rozhraní. [MyS11]

Microsoft SQL Server

Microsoft SQL Server, označován jako MSSQL nebo jen SQL Server, je RDBMS vyvinutý společností Microsoft. Vývoj SQL Serveru začal ve firmách Microsoft a IBM v roce 1985. Ještě nedávno byla k dispozici placená verze a Developer Edition, která byla zdarma pouze pro vývojářské týmy a studenty. Dnes je k dispozici i neplacená verze SQL Server Express. Pro práci s SQL Serverem se nejčastěji využívá SQL Server Management Studio poskytovaný od verze MSSQL 2005. Hlavními jazyky jsou pro SQL Server SQL a T-SQL (Transact-SQL), který rozšiřuje standardní SQL o procedurální programování, lokální proměnné a další funkce. Nejnovější verzí je SQL Server 2014[2], který se již specializuje na big data a také umožní propojení s cloudovými službami Azure. [MSSQL01]

Oracle Database

Společnost Oracle Corporation[3] patří mezi největší světové výrobce podnikových systémů, jako jsou relační databáze, nástroje pro správu databází a CRM (Customer Relationship Management) systémy. První Oracle databáze označena jako Oracle Version 2 vznikla již v roce 1979, jako první relační databázový systém. Nejnovější verze Oracle Database 12c byla první databáze určená pro cloud. Pro správu databáze dodává firma zdarma grafický nástroj Oracle SQL Developer, který umož-

[1] `http://www.mysql.com/about/`
[2] `https://www.microsoft.com/sqlserver/cs/cz/`
[3] `http://www.oracle.com`

ňuje prohlížení databáze, spouštět SQL příkazy, editovat a debugovat PL/SQL příkazy. Společnost Oracle vlastní i konkurenční MySQL, když v roce 2010 zakoupila společnost Sun Microsystems. Společnost poskytuje Oracle NoSQL Database, což je NoSQL databáze typu key/value.

PostgreSQL

PostgreSQL patří mezi objektově-relační databázové systém, distribuovaný zdarma pod licencí PostgreSQL License, která je podobná licenci MIT (svobodná licence z Massachusetts Institute of Technology). Byl vytvořen týmem profesora Michaela Stonebrakera na Kalifornské univerzitě v Berkeley v roce 1986[4]. Hlavními nástroji pro správu databáze jsou psql, aplikace pro příkazovou řádku umožňující zadávaní SQL dotazů, a grafické administrační rozhraní pgAdmin.

Firebird

Databázový systém Firebird vychází ze systému InterBase společnosti Borland. V roce 2000 byly uvolněny zdrojové kódy systému InterBase, které začal ve stejném roce tým vývojářů upravovat, a tak vznikl Firebird, který je dál vyvíjen pod vedením neziskové organizace Firebird Foundation. Současná verze Firebirdu je 2.5.2. Díky přepsání do jazyka C++ a velkého množství úprav původního kódu se tento multiplatformní systém stal velmi stabilní. Systém je nabízen pod volnou licencí Initial Developer's Public License vycházející z MPL (Mozilla Public Licence) pro svobodný software[5]. [Jak07]

[4] `http://www.postgresql.org/docs/current/static/history.html`
[5] `http://www.firebirdsql.org/en/licensing/`

4 NoSQL databáze

S nárůstem objemu dat, jejich vzrůstající propojeností a ztrátě předvídatelné struktury v posledních letech přestaly být relační databázové systémy vhodnou volbou pro některá data. Google byl první společností, která se rozhodla řešit problém s ukládáním big dat. Vytvořil nové databázové uložiště BigTable, které neukládalo data klasicky po řádcích ale po sloupcích. Dalším řešením byl projekt Dynamo od společnosti Amazon, který se zabývá ukládáním dat na principu key/value. Oba projekty stanovily základy pro vznik a vývoj NoSQL databází. [Šel11]

4.1 Popis

V roce 2009 byl termín NoSQL databáze zvolen pro početnou skupinu nerelačních datových uložišť. Takové databáze na rozdíl od relačních většinou nepoužívají SQL jako svůj dotazovací jazyk. Název NoSQL je občas mylně vykládán jako „No SQL“ (žádné SQL), databázová komunita ale raději vysvětluje tento pojem spíše jako „Not only SQL“ (nejen SQL). Někdy se pro tato datová uložiště používá termín postrelační. V širším významu se do této skupiny zahrnují také databáze XML, grafové, dokumentové, objektové a také softwarová řešení, která dokonce nejsou ani v tradičním pojetí databázemi vůbec. Základními charakteristikami většiny NoSQL databází jsou: [Flo12]

- Nepoužívají relační model a jazyk SQL,
- většina NoSQL databází jsou open-source,
- jsou navrženy pro práci v clusteru,
- umožňují snadnou podporu replikací a distribuce dat,
- poskytují vlastní jednoduchá API,
- nemají stanovené žádné schéma (nestrukturovaná data),
- jsou vysoce horizontálně škálovatelné,
- částečné využívají ACID,
- podporují transakci BASE,

- CAP teorém.

Data uložená v NoSQL databázi nemusejí mít pevné uspořádání, mohou být strukturovaná, nestrukturovaná nebo semistrukturovaná. V NoSQL se především využívá schopnosti ukládat a načítat velké množství dat bez závislosti na vztazích. Díky tomuto jinému přístupu k ukládání dat do databáze jsou NoSQL v některých případech vhodnější než databáze relační. Příkladem použití může být ukládání několika milionů dat typu key/value do jednoho asociativního pole.

NoSQL systémy pracují s distribuovanou architekturou a daty, které jsou redundantním způsobem uloženy na několika serverech. Výhodou nerelačních databází je schopnost tolerovat výpadky sítě a vysoká rychlost zpracování obrovského objemu dat, na úkor malé funkcionality omezující se v některých případech na pouhé ukládání dat. NoSQL databáze neposkytují plnou podporu ACID. Umožňují proto pouze částečnou konzistenci, díky čemuž se ale zvyšuje rychlost vykonávaných dotazů, zvětšuje se dostupnost a možnost škálovatelnosti databáze. [Pok12]

4.2 Base

NoSQL databáze považují ACID za příliš přísný. Místo toho preferují raději přístup označovaný jako BASE (Basically Available, Soft-state, Eventually Consistent), který upřednostňuje dostupnost, částečnou degradaci a výkon před konzistencí a izolací databáze. Často je BASE chápan jako protiklad ACID. BASE se skládá z následujících principů:

- **Basically Available** – při poruše spojení mezi uzly zůstane ve většině případů dostupnost dat zachována a systém je v podstatě funkční po celou dobu. Tato vlastnost se dosahuje pomocí vysoce distribuovaného přístupu ke správě databází. NoSQL databáze mají data uložené v systémech pro ukládání dat s vysokým stupněm replikace.
- **Soft-state** – databáze většinou nedodržují konzistentní požadavky ACID přístupu, neboli aplikace nemusí být v každém okamžiku konzistentní. Základní koncepcí je, že konzistence dat je problémem vývojáře a neměla by být řešena v databázi.
- **Eventually Consistent** – NoSQL databáze musí být schopny dosahovat konzistentních dat postupně, neboli v určitém okamžiku v budoucnosti budou data v konzistentním stavu. Není však zaručeno, kdy k tomu dojde. To je úplný odklon od požadavku přístupu ACID na okamžitou konzistenci dat.

V praxi, např. v internetových aplikacích, se používají oba zmíněné přístupy, ACID a BASE. V případech, kdy je nezbytná silná konzistence dat, jako jsou bankovní operace, se používá ACID. V jiných případech, kde je preferována dostupnost, se používá přístup BASE. Použití přístupu ACID nebo BASE, případně kombinace obou, závisí především na konkrétním přístupu. [Cha11][Tiw11]

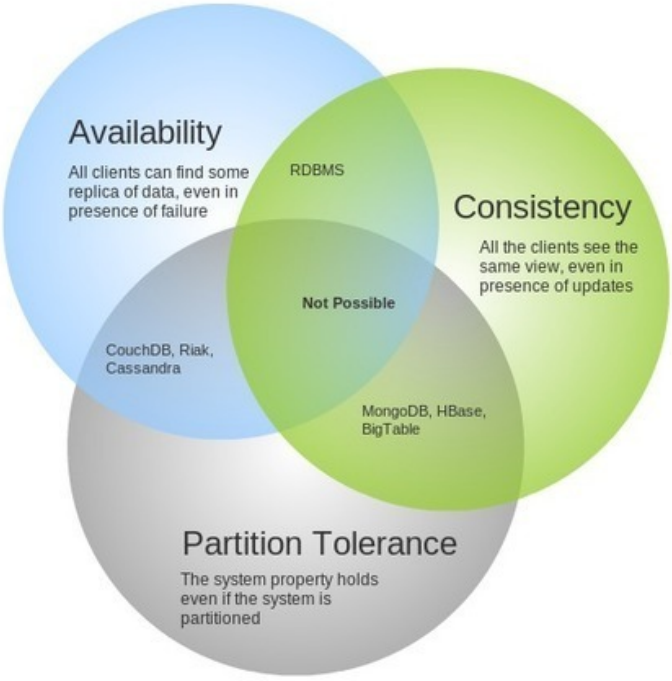
4.3 CAP teorém

V roce 2000 byl profesorem Erikem Brewerem z University of California představen CAP teorém (Consistency – C, Availability – A, Partition tolerance – P), který vysvětluje možnosti realizace distribuovaných systémů, nazývaný také Brewerův teorém. V posledních letech si tento princip získal velikou popularitu i přesto, že je některými odborníky kritizován. Trojice požadavků na distribuované prostředí:

- **Konzistence** (Consistency) znamená, že kdykoliv jsou data zapsána, je pro kohokoliv, kdo čte z databáze, zobrazena poslední verze dat, neboli všechny uzly distribuovaného systému vidí ve stejný časový okamžik shodná data.
- **Dostupnost** (Availability) znamená, že každý klient po svém dotazu dostane informaci o tom, zda byla operace úspěšná nebo neúspěšná. Také se někdy používá i vysvětlení, že každý klient může číst i zapisovat data. Vysoká dostupnost je dosahována obvykle pomocí většího počtu propojených fyzických serverů působící jako jedna databáze s rozdělením dat mezi jednotlivé databázové uzly a za pomoci využití replik dat.
- **Odolnost vůči rozdělení sítě** (Partition tolerance) vyjadřuje schopnost systému číst a zapisovat data i v případě, pokud došlo k přerušení spojení mezi význačným počtem databázových uzlů. Odolnost může být dosažena pomocí mechanismů, kterými jsou zápisy místo na požadovaných nedosažitelných uzlech poslány stále ještě přístupným uzlům. Když se pak navrátí nedostupné uzly zpět do sítě, obdrží požadované zápisy, které chyběly.

CAP teorém tvrdí, že pro jakýkoliv distribuovaný systém je nemožné splňovat současně všechny tři uvedené požadavky. Z obrázku 4.1 plyne, že systém lze rozdělit do následujících tří kategorií:

- **CA** (Consistency + Availability) – je zaručena dostupnost a konzistence na úkor odolnosti k rozdělení sítě. Takové systémy obvykle splňují zásady ACID a používají se např. v bankách. Představitelé: RDBMS.

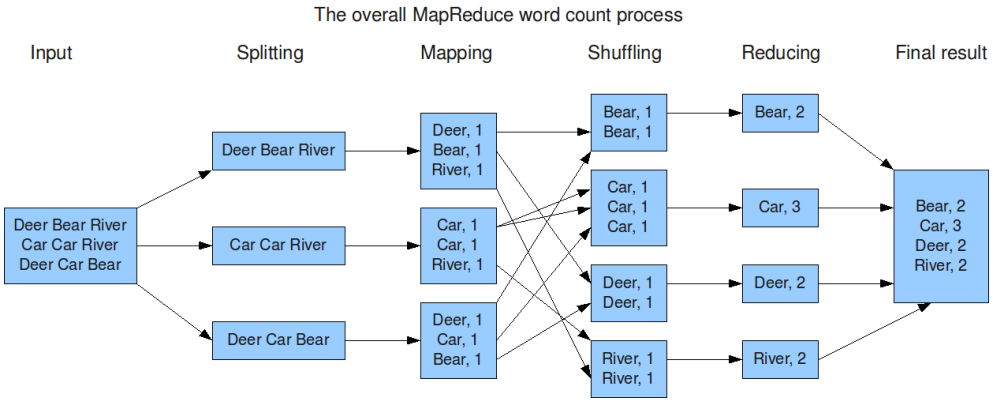


Obrázek 4.1: CAP teorém. [Ver13]

- **CP** (Consistency + Partition tolerance) – nesplněnou vlastností je dostupnost. Tato volba je vhodná pro systémy, kde dostupnost není kritická a je důležité zachvoat především konzistenci dat. Představitelé: Redis, MongoDB, BigTable.
- **AP** (Availability + Partition tolerance) – systém má zaručenou dostupnost a odolnost k rozdělení sítě, tentokrát na úkor konzistence. Tyto systémy jsou označovány jako slabě konzistentní a konzistence dat dosahují postupně. Jsou především vhodné v případech, kdy je nepostradatelná odolnost vůči rozdělení dat. Představitelé: Cassandra, Riak, CouchDB. [Pok12]

4.4 Map Reduce

Map Reduce je model určující, jak zpracovávat velké objemy dat v clusterech s využitím paralelního zpracování na větším počtu uzlů. První významnou realizaci byl The Map Reduce Framework napsán v jazyce C++ od společnosti Google. Základní myšlenky tohoto projektu byly převzaty softwarem Hadoop a vznikla tak open-source implementace v jazyce Java, jež je v dnešní době hojně využívána. Tento



Obrázek 4.2: MapReduce. [Gro09]

model používá např. MongoDB.

Princi Map Reudce, viz obrázek 4.2, vychází z funkcionálního programování, kde existuje mapovací a redukční funkce. Map funkce uplatní na každý prvek datové struktury operaci. Výstupem je pak nová kolekce hodnot, přitom ale není původní kolekce změněna. Funkce Reduce použije agregační funkci na novou kolekci hodnot a na výstupu vytvoří jednu jedinou hodnotu. Podobný princi se používá u zpracování velkých objemů dat v distribuované databázi. [Dea04][Tiw11]

- **Krok Map** – hlavní uzel, který získává vstup, rozdělí vstupní data zadání na části a předá je pracovním uzlům. Pracovní uzel může opět rozdělit svojí část a předat jí ke zpracování jiným pracovním uzlům, pro které se tak stane řídícím uzlem. Tento způsob vede k víceúrovňové stromové struktuře. Pracovní uzel pak použije funkci na každou kolekci dvojic klíč/hodnota a výslednou kolekci nově vytvořených dvojic klíč/hodnota předá zpět svému nadřazenému uzlu.
- **Krok Reduce** – hlavní uzel shromažďuje výsledky od pracovních uzlů, které zkombinuje (spojí data se stejným klíčem) a vytvoří výstup neboli odpověď na původní řešený problém.

4.5 Distribuce dat

Data mohou být umístěna na jednom uzlu. To je výhodné, protože to nezpůsobuje komplikace s rozmístěním dat na více uzlech, např. u grafových databází, se skoro hotovým grafem, je těžké data distribuovat. Jelikož jsou ale možnosti jednoho uzlu omezené, je výhodnější rozmisťovat data na více uzlů. Existují dvě varianty distribuce dat, sharding a replikace.

Sharding je metoda, která určuje, na jakém uzlu se nacházejí data. Shard je pak označení části dat. Každý shard může být umístěn na odlišném databázovém serveru. Jelikož je databáze rozdělena a distribuovaná na více uzlech, je množství dat v každé databázi menší. Každý uzel pracuje se svojí částí dat, čímž umožňuje snížit zatížení serveru. Díky tomu se snižuje rychlost zpracování jednotlivých dotazů. Pokud dojde k výpadu jednoho z uzlů, je zbytek dat stále přístupný[1]. Nevýhodou ale je, že data se mohou při výpadku ztratit. Tuto metodu používá například Redis nebo MongoDB.

Na rozdíl od shardingu replikace duplikuje data z jednoho uzlu na ostatní, což znamená, že na každém uzlu budou uložena stejná data. Díky tomu nedojde při výpadku uzlu ke ztrátě dat. Replikace může být buď master-slave nebo peer-to-peer. Master-slave replikace, jak již název napovídá, má jeden primární uzel (master), který může číst i zapisovat data. Sekundární uzly (slaves) mohou pouze číst. Tato replikace je výhodná při větším počtu dotazů na čtení dat, naopak při větším počtu požadavků na zápis či aktualizaci dat má uzel master omezené schopnosti, jelikož jako jediný smí tyto požadavky provádět. Další výhodou master-slave replikace je, že i při výpadku primárního uzlu, umožňují sekundární uzly čtení dat. Sekundární uzel může být pak automaticky nebo manuálně povýšen na primární uzel, čímž převezme práva mastera.

Peer-to-peer (P2P) replikace se od master-slave liší především tím, že jsou všechny uzly rovnocenné a každý uzel má právo pro zápis i čtení. Tím se řeší některé nedostatky master-slave replikace především omezený výkon primárního uzlu při zápisu. Díky tomu je zaručena větší spolehlivost, jelikož výpadek jednoho uzlu nezpůsobí výpadek celého systému. Problémem P2P replikace je konzistence. V případě zápisu na dvě rozdílná místa může vzniknout konflikt. Sharding a replikace se můžou navzájem kombinovat. V případě master-slave replikace a shardingu bude existovat více primárních uzlů, ale každý záznam bude mít pouze jednoho mastera. Díky tomu může nastat situace, kdy pro jedna data bude uzel primární, ale pro jiná sekundární. [Tiw11][Sad13]

[1] `http://highscalability.com/blog/2010/10/15/troubles-with-sharding-what-can-we-learn-from-the-foursquare.html`

4.6 Škálovatelnost

Škálovatelnost je vlastnost systému umožňující reagovat na náhle změny potřeby obsluhy systému, např. pojmout rostoucí počet dat. Škálovatelnost může být vertikální (scale up) a horizontální (scale out). Při vertikálním škálovatelnosti se dosáhne zvýšení fyzického výkonu serveru přidáním paměti a změnou procesoru za výkonnější. Takové servery bývají ale drahé. Při horizontální škálovatelnosti nedochází k vylepšování jednoho uzlu, ale místo toho se data rozdělí na více vzájemně spolupracujících serverů. Díky tomu může být výpočet rozdělen na jednotlivé paralelní úlohy. Přidávání další jednotky do systému se jeví, oproti stálému vylepšování technických parametrů jednoho stroje, finančně výhodnější, jelikož postačí i levnější stroje. Horizontální škálovatelnost se vyskytuje zejména u NoSQL databází. V NoSQL mohou být ale data škálována i vertikálně. [Tiw11][Pok12]

4.7 Dotazování

Jak již samotný název NoSQL databáze napovídá, tyto systémy postrádají dotazovací jazyk SQL. Jelikož se NoSQL systémy nezakládají na relačním modelu a neexistuje v nich žádný standard pro dotazování, tak se k datům v různých systémech přistupuje rozdílnými způsoby. Některé NoSQL databáze podporují jazyk SQL, pouze však v omezené formě (SimpleDB). Operace agregace, spojení a zanořování dotazů nejsou v této omezené formě podporovány. Širší podmnožinou SQL jsou jazyky GQL (Google Query Language), používaný v nástroji Google AppEngine, a HQL (Hypertable Query Language), který používá databáze Hypertable.

Dotazování v NoSQL probíhá především prostřednictvím klíče pomocí jednoduchého API. Obvyklé API pro NoSQL databázi obsahuje následující jednoduché operace:

- `get(klíč)` – získá hodnotu daného klíče,
- `put(klíč, hodnota)` – vytvoří nebo aktualizuje hodnotu klíče,
- `delete(klíč)` – odstraní klíč včetně jeho hodnoty,
- `execute(klíč, operace, parametry)` – vyvolá požadovanou operaci na hodnotě určené klíčem.

Kvůli horizontální škálovatelnosti nepodporují NoSQL databáze operace spojení (JOIN) a uspořádání (ORDER BY). V případě, že jsou tyto operace požadovány, mohou být implementovány na straně klienta, stejně tak, jako ostatní potřebné dotazovací možnosti. Takovéto řešení znamená ruční programování dotazů, což je vhodné pro jednoduché úlohy, ale u komplikovanějších úloh je to naopak velmi časově náročné. [Pok12]

V budoucnu je pravděpodobné, že vznikne pro všechny NoSQL systémy nějaký standardizovaný dotazovací jazyk. Pro dokumentově orientované databáze se začalo již hovořit o UnQL (Unstructured Data Query Language)[2] od společnosti Couchbase. Jeho další vývoj je však nejistý.

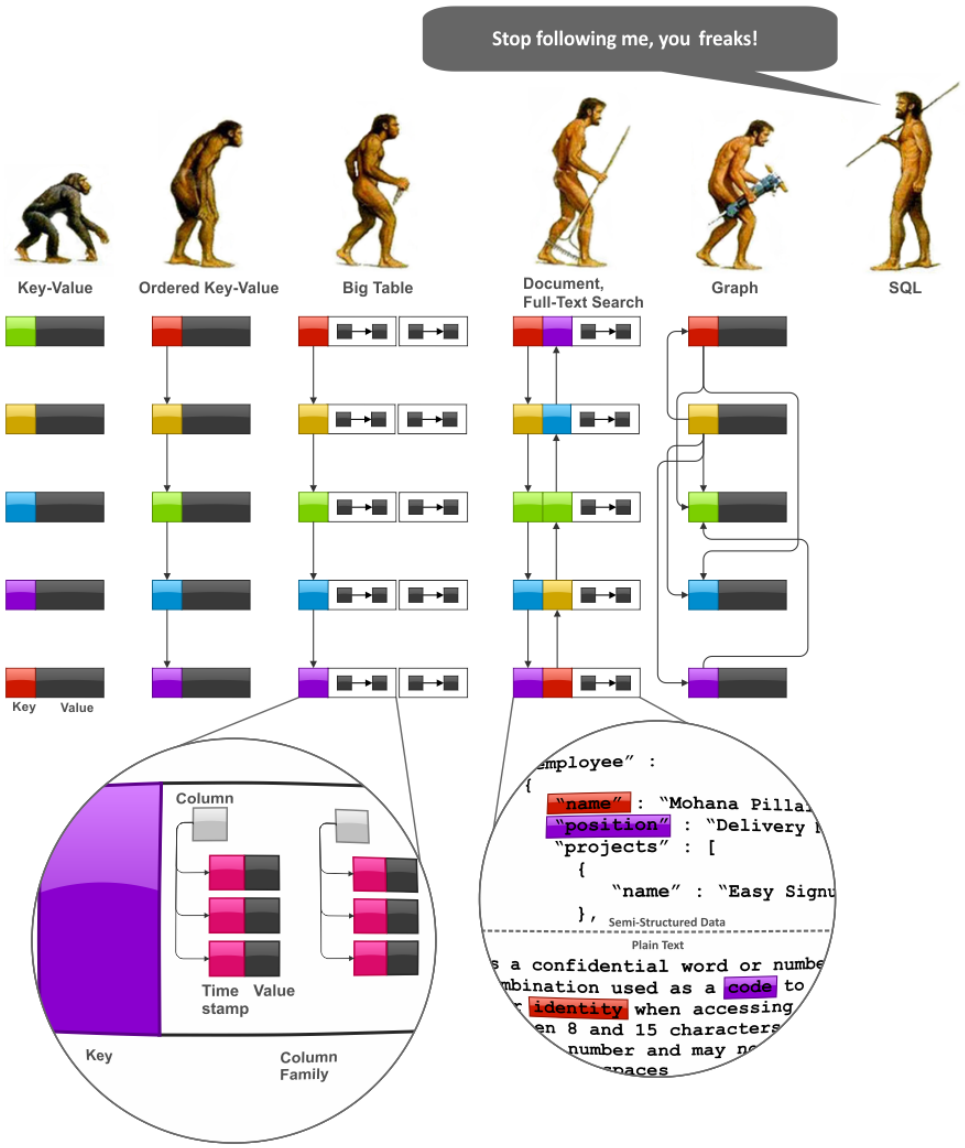
4.8 Datový model

Při klasickém popisu databází se používá (logický) datový model. NoSQL však využívají jiný přístup, intuitivní a bez jakýchkoliv předepsaných formálních základů. Díky tomu je terminologie velice různorodá a stírají se zde rozdíly mezi konceptuálním a databázovým pohledem na data. Existuje několik druhu dělení NoSQL. Nejčastěji se používá dělení do čtyř kategorií, viz obrázek 4.3, podle kvalitativních parametrů a především podle jejich datového modelu.

- Databáze typu klíč/hodnota (Key/value database),
- dokumentově orientované databáze (Document-oriented database),
- sloupcové databáze (Column oriented database),
- grafové databáze (Graph databases).

Uvedené dělení zobrazuje pouze užší koncepci NoSQL databází. Do širšího pojetí lze dále přidat i objektové databáze, XML databáze, multidimenzionální databáze, cloud řešení a další. [Pok12]

[2] `http://www.couchbase.com/press-releases/unql-query-language`



Obrázek 4.3: Datové modely NoSQL. [Kat12]

4.8.1 Databáze typu klíč/hodnota

Databáze typu klíč/hodnota, označovány také jako key/value, mají velmi jednoduchý datový model. Model lze chápat jako hašovací tabulku, která se skládá z textových dvojic klíč a hodnota. Klíč jednoznačně identifikuje hodnotu. Hodnota může obsahovat libovolný blok dat (typicky BLOB) bez potřeby analýzy obsahu na straně

databáze. Základní operace, které tyto databáze podporují, jsou vložení hodnoty pro klíč (put), získání hodnoty podle klíče (get) a odstranění klíče a jeho hodnoty z datového uložiště (delete). Tento model se hodí například pro ukládání informací o uživatelských profilech. Každý uživatel má přiděleno vlastní unikátní `user_id`, `user_name` a další nastavení, jako je jazyk a časová zóna.

Jelikož existuje vždy jedinečný klíč pro konkrétní objekt, stačí v databázi vyhledat pouze tento jedinečný klíč a získat výsledky. Pokud by ale bylo požadováno dotazovat se podle nějakého atributu uloženého v poli hodnota, musí být zvolen jiný datový model. Typ databáze key/value je bez schématu a používá se především pro velký objem nestrukturovaných dat. Výhodou takovýchto databází, které jsou podobné souborovým systémům, je především velký výkon, rychlost, dobrá škálovatelnost a také to, že dvojice klíč/hodnota mohou být různých typů. Naopak za nevýhodu je považován až příliš jednoduchý datový model. Mezi nejznámější key-value databáze patří Redis, Riak, Amazon DynamoDB, Oracle NOSQL Database, Memcached a Project Voldemort. Podrobnější informace viz kapiotla 4.9 a 4.10. [Pok12]

4.8.2 Dokumentové databáze

Dokumentově orientované databáze jsou rozšířeným datovým modelem key/value. Rozdíl je především v tom, že základním prvkem každé dokumentové databáze jsou semistrutkurované dokumenty. Každý dokument je reprezentován pomocí unikátního klíče, kterým může být jednoduchý řetězec. Data mohou být ukládána ve formátu XML (Extensible Markup Language), JSON (JavaScript Object Notation), BSON (Binary JSON) a v dalších fotmátech. Pro ukládání dokumentu je nutné dodržet zvolený formát, který databáze podporují. Interním jazykem je JavaScript.

Hlavní výhodou dokumentového modelu je, že jednotlivé dokumenty mohou mít navzájem různou strukturu, která nemusí být předem definována. Díky tomu může struktura dokumentu obsahovat velké množství různých informací a také, oproti klasickým relačním databázím, může měnit strukturu dat za běhu. JavaScript je v poslední době hojně využívaný na webu a proto se tento typ NoSQL databází stal velmi populární především u webových vývojářů. Mezi nejpoužívanější dokumentově orientované databáze se řadí MongoDB a CouchDB. [Tiw11][Nia13]

MongoDB

MongoDB[3] patří mezi nejpopulárnější dokumentově orientované databáze. Databáze byla vytvořena firmou MongoDB Inc. (dříve 10gen). Jedná se o open-source databázi napsanou v jazyce C++, která nemá strukturu (schema-free). MongoDB ukládá data jako dokumenty ve formátu BSON. Maximální velikost dokumentu je 16 MB. Dokumenty jsou shromažďovány do kolekcí, které se podobají tabulkám z relačních databází, nemají však pevné schéma. Datový model je velmi flexibilní, jakékoliv políčko může obsahovat více hodnot a může v něm být uložena další datová struktura nebo může zůstat prázdné. Libovolné pole v dokumentu může být indexované, k dispozici je také sekundární indexování. [Mon14]

MongoDB disponuje vlastním dotazovacím jazykem, který je velmi podobný SQL. Podporuje množinové operace, regulární výrazy nebo dokonce dovoluje spustit metodu napsanou v JavaScriptu na straně serveru. Také využívá návrhový vzor Map Reduce pro hromadné operace nad daty a sharding pro horizontální škálování. Umožňuje master-slave replikaci, kde uživatel do masteru zapisuje a čte z něj. Slave slouží klasicky jen pro čtení nebo zálohování zkopírovaných dat z primárního uzlu. Pokud dojde k výpadku uzlu master, systém automaticky zabezpečí zvolení nového primíárního uzlu, kterým se se stane jeden ze sekundárních uzlů. [Klo12]

Všechnu dostupnou paměť RAM využívá MongoDB jako cache pro data, aby nemuselo ustavičně přistupovat na disk. Je ale potřeba sledovat zaplnění kapacity disku a paměti RAM. Pokud by byl disk zaplněn, tak MongoDB zablokuje všechny operace včetně mazání. V případě, že se zaplní paměť RAM, zpomalí se celý systém na úroveň rychlosti disku. [Hof10]

CouchDB

Dokumentově orientovaná databáze CouchDB[4] je napsaná v jazyce Erlang a vyvíjena organizací Apache Software Foundation jako open-source. Patří k nejstarším dokumentovým databázím, první verze vyšla již v roce 2005. Tato databáze je určena především pro web, také je vhodná pro použití v mobilních zařízeních především kvůli svým replikačním

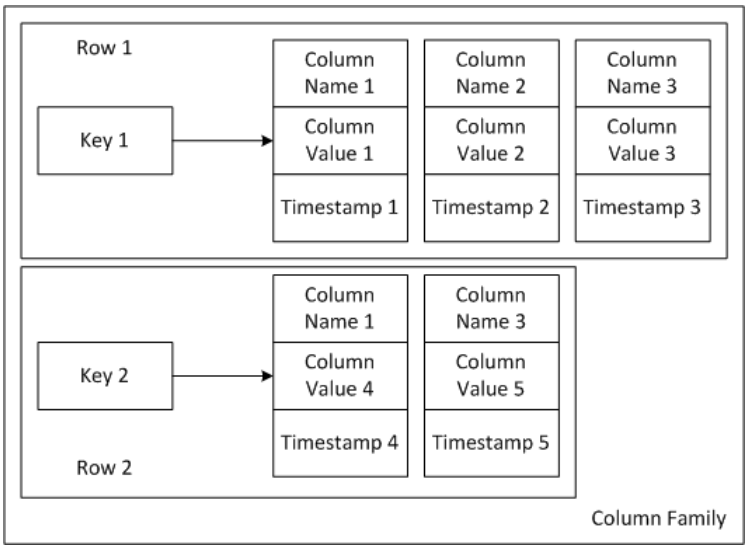
[3] `https://www.mongodb.org/`
[4] `http://couchdb.apache.org/`

a synchronizačním schopnostem. Tato databáze bývá nainstalována v mobilních telefonech a dalších mobilních zařízení tak, že se synchronizuje s centrální databází kdykoliv je online. Pro dotazování nabízí použití REST HTTP API a pro ukládání dat využívá formát JSON.

CouchDB spravuje obdobně jako MongoDB kolekce JSON dokumentů[5]. Jednotlivé dokumenty v rámci kolekce nemají společné schéma, přičemž nabízejí dotazovací kapacity prostřednictvím pohledů. Obdobě jako v Map Reduce jsou pohledy vymezeny agregačními funkcemi a současně s tím tříděny. Databáze CouchDB podporuje také peer-to-peer replikaci s automatickou detekcí konfliktů. Změny dat jsou pravidelně kopírovány mezi servery, čímž se eliminuje zranitelné místo. V CouchDB je pro ukládání všech dat, dvojic klič/hodnota, použit B-strom tak, že je podporováno seřazení podle klíče. [Pok12]

4.8.3 Sloupcové databáze

Sloupcově orientované databáze si ukládají data po sloupcích namísto klasického přístupu ukládání dat po řádcích. Díky tomu se uchovají data bez zbytečného zaplnění místa na disku, jelikož se při ukládání nevytvoří sloupec, pokud pro něj neexistuje hodnota. Tento přístup zlepšuje výkon především v datových centrech a v analytických systémech, které potřebují velké množství agregovaných dat.



Obrázek 4.4: Datový model v databázi Cassandra. [Mam11]

[5] `http://wiki.apache.org/couchdb/Introduction`

Datový model sloupcových NoSQL databází tvoří tři základní pojmy. Sloupec (Column) je nejnižší úroveň dat a je tvořen trojicí, která se skládá ze jména, hodnoty a časového razítka. Časová razítka modelují čas a slouží k rozlišování verzí jednotlivých hodnot. Super sloupce (Super column family) nemají časové razítko, obsahují však několik sloupců (ne však jiný super sloupec) a tvoří agregovanou pojmenovanou jednotku. Struktura, která obsahuje neomezené množství řádků, se označuje jako skupina sloupců (Column family). Každý řádek má svoje jméno (klíč). Jednotlivé řádky jsou tvořeny sloupci případně super sloupci. Skupina sloupců znázorňuje, jak jsou data uložená na disku. Celá struktura datového modelu pro sloupcovou databázi Cassandra je znázorněna na obrázku 4.4. Výhodou těchto databází je oproti jednoduchému přístupu key/value bohatší datový model. Takováto data se řadí spíše již do kategorie semistrukturovaných dat. Hlavními představiteli sloupcově orientovaných databází jsou Cassandra, Google BigTable, HBase a další. [Pok12][Rah10]

Cassandra

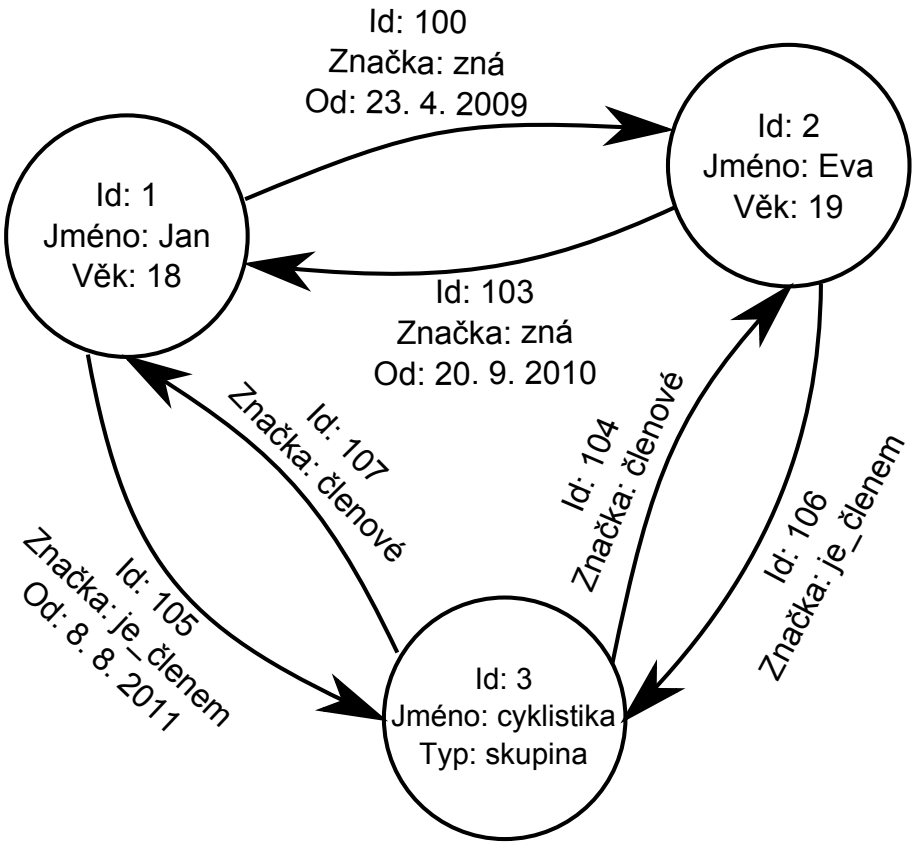
Významným představitelem sloupcově orientovaných NoSQL databází je vysoce škálovatelná distribuovaná databáze Apache Cassandra[6]. Navržená byla v roce 2008 pro sociální síť Facebook. V roce 2009 by tento open-source projekt převzat společností Apache Software Foundation, která ho stále rozvíjí. Cassandra je využívána společnostmi Instagram, Cisco, Ubisoft, Twitter a dalšími[7].

Apache Cassandra je napsána v programovacím jazyce Java a spojuje vlastnosti distribuovaného systému Amazon Dynamo s datovým modelem Google BigTable. Jedná se o multiplatformní databázi, může být tak používaná jak na platformě Windows, tak i na Unix. Cassandra má podobně jako BigTable sloupcově orientovaný datový model (Column family). Využívá výše zmíněný model Map Reduce a díky tomu je dobře horizontálně škálovatelná. Od projektu Dynamo přebírá symetrickou peer-to-peer architekturu a eventuální konzistenci. Databáze poskytuje vlastní dotazovací jazyk CQL (Cassandra Query Language), který je svou syntaxí podobný jazyku SQL. [Bla10][Dat14][Str11]

[6] `http://cassandra.apache.org/`
[7] `http://planetcassandra.org/companies/`

4.8.4 Grafové databáze

Předchozí uvedené modely NoSQL databází se snažily být bez jakéhokoliv schématu. V těchto databází můžou záznamy naopak mít velké množství vztahů. Grafové databáze jsou vlastně druhem síťových databází, které si ukládají data reprezentovaná prostřednictvím grafu. Většinou se jedná o orientovaný a ohodnocený graf, který je tvořen vrcholy (nodes) a hranami (edges). Jednotlivé vrcholy představují objekty a hrany jejich vazby. Do každého vrcholu běžně vstupuje a vystupuje několik hran. Mohou však existovat i takové vrcholy, které mezi sebou žádné vazby nemají. Hrana musí mít vždy jeden vrchol, ze kterého vystupuje, a další vrchol, do kterého vstupuje. Většinou jsou data ukládána na vrcholech, ale např. u Neo4j mohou být data uložena stejným způsobem ale na hranách. Na obrázku 4.5 je znázorněn příklad jednoduchého modelu grafové databáze. Všechny vrcholy obsahují vlastnosti a mezi každýma dvěma vrcholy existuje hrana, která též obsahuje nějakou vlastnost. [Pok12]



Obrázek 4.5: Ukázkový model grafové databáze. [Pok12]

Grafové databáze se v současné době velmi rychle rozvíjejí. Používají se především pro řešení úloh, jež je velmi obtížné spravovat v klasických relačních databázích.

Jejich výhodou je možnost použít grafové algoritmy pro hledání nejkratší cesty mezi dvěma uzly, nalezení minimální kostry grafu nebo např. Dijkstrův algoritmus. Tyto databáze nacházejí využití především u sociálních sítí, které vyžadují vést záznamy o velkém počtu a typu vztahů. Významnými představiteli grafových databází jsou Neo4j, Titan a OrientDB[8].

Neo4j

Neo4j[9] je velmi populární robustní databázový systém, který je velmi škálovatelný a vysoce výkonný. Byl vytvořený v programovacím jazyce Java společností Neo Technology v roce 2007. Na rozdíl od ostatních NoSQL databází umožňuje transakční zpracování s úplnou podporou ACID vlastností. Zvládne ukládat data i s několika miliardami uzlů a vztahů, které dokáže vysokorychlostně zpracovávat.

Databáze Neo4j používá vlastní dotazovací jazyk Cypher, který inspirován jazyky SQL a SPARQL. Cypher je deklarativní jazyk, který dovoluje popsat, co a jak se má v databázi nalézt. Díky podpoře vlastního REST API, které je používáno pro komunikaci mezi klientem a databázovým serverem pomocí protokolu HTTP, se Neo4j využívá v moderních webových aplikacích. Hodí se pro použití u sociálních sítí, různých wikipediích a dalších síťově orientovaných webových dat. Obdobně, jako v případě MongoDB, využívá Neo4j replikaci master-slave. Neo4j využívají například společnosti Adobe, EBay nebo online časopis CHIP[10]. [Hof09]

4.9 Riak

Riak[11] je open-source databázový systém, který využívá principy Amazon Dynamo. Tento databázový systém, který je napsaný v jazyce Erlang, byl vytvořen firmou Basho Technologies v roce 2009. Slouží převážně jako distribuované uložiště typu key/value. Riak je určen především pro web a webové aplikace a podporuje operační systémy Linux, BSD, Mac OS X a Solaris. [Cat11]

[8] `http://db-engines.com/en/ranking/graph+dbms`
[9] `http://www.neo4j.org/`
[10] `http://www.neotechnology.com/customers/`
[11] `http://basho.com/`

Objekty Riaku, které mohou být ve formátu JSON, obsahují unikátní hodnotu s klíčem. Dvojice je uložena v tzv. bucketu (do češtiny volně přeloženo jako kbelík). Hodnotami pro jednotlivé klíče může být cokoliv, od prostého textu, JSON nebo XML po obrázky a video soubory. Bucket je vlastně jmenný prostor pro klíče. Má podobnou funkci jako tabulky v relačních databází nebo jako adresáře ve file systému. Bucket společně s klíčem vytváří unikátní identifikátor pro konkrétní hodnotu.

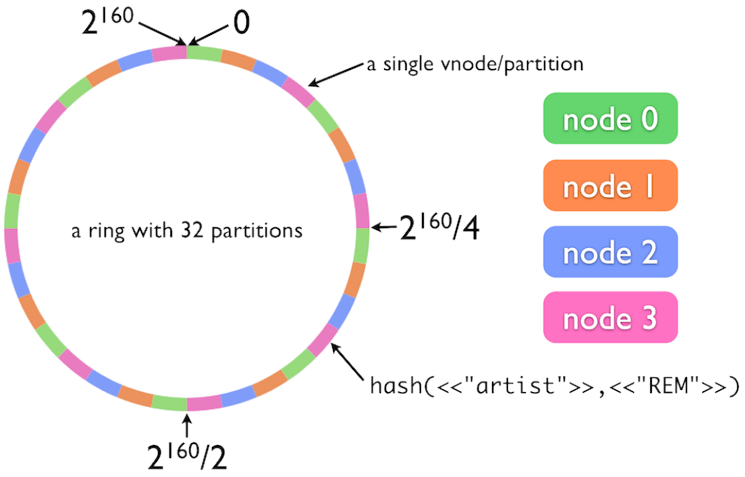
Riak obsahuje kromě rozhraní přímo pro jazyk Erlang také i REST rozhraní, které umožňuje podporu knihoven ostatních jazyků jako je Java, Python, PHP, .NET a mnoho dalších. Následně je ukázán kód pro vytvoření a čtení hodnot z bucketu v Riaku pomocí jazyku Erlang. Ukládá se integer 1 do klíče pojmenovaného „one“.

```
%% Creating Objects In Riak
%% For these examples we will be using the "test" bucket.
MyBucket = <<"test">>.

Val1 = 1.
Obj1 = riakc_obj:new(MyBucket, <<"one">>, Val1). riakc_pb_socket:put(Pid, Obj1).

%% Reading Objects From Riak
{ok, Fetched1} = riakc_pb_socket:get(Pid, MyBucket, <<"one">>).

Val1 =:= binary_to_term(riakc_obj:get_value(Fetched1)).
```

Všechny uzly v distribuovaném systému Riak obsahují kompletní nezávislou kopii dat. Riak automaticky replikuje data pomocí peer-to-peer replikace, defaultně je nastaven počet replikací na tři. Žádný uzel nemá větší odpovědnost než jiný a žádný uzel nemá úlohy, které by vykonávaly jiné uzly. Kvůli tomu i při výpadku většího počtu uzlů je stále možno číst a zapisovat data, čímž je zajištěna škálovatelnost a vysoká tolerance proti výpadku uzlu. Pokud je uzel nedostupný, přebírá jeho operace sousední uzel. Při návratu nedostupného uzlu mu jeho soused pošle zpět aktualizovaná data. Cluster je rozdělen pomocí takzvaného Riak ringu, díky čemuž běží na jednom fyzickém uzlu několik virtuálních (vnode). Data jsou distribuována pomocí konzistentní hašovací funkce. Funguje to tak, že pokud se provádí nějaká operace s key/value v Riaku, je následně hašováná kombinace bucketu a klíče. Výsledek hašovací funkce je namapován na 160-bitový integrový prostor, který je rovnoměrně rozdělen na části. Na obrázku 4.6 je znázorněn ring pro 4 fyzické uzly, které se dělí na 32 částí spravovaných 32 virtuálníma uzly. Každý ze čtyř fyzických strojů má nárok na osm virtuálních uzlů, tím se stroj stává zodpovědný za všechny klíče, které jsou reprezentovány osmi virtuálními uzly.

Databázový systém Riak má implementovanou schopnost využívat již výše zmí-

něný model Map Reduce. Definice jednotlivých funkcí je systému předávána ve formátu JSON. Pro Map Reduce je možné využít indexy, které Riak podporuje. Indexace je zajištěna pomocí přiřazení několika hodnot danému indexovacímu klíči.



Obrázek 4.6: Riak ring. [Ria11]

Riak se v praxi využívá například v projektu GitHub nebo v projektech firem Bestbuy a Dell[12]. [Ria11][Red12]

4.10 Redis

Redis[13] patří k nejvýznamnějším NoSQL databázím, řadí se do kategorie typu key/value. Byl vytvořen v roce 2009 jako soukromý projekt Salvatora Snafilippa. Později byl projekt uvolněn jako open-source a pro jeho vysokou popularitu se do jeho vývoje zapojila firma VMWare, od roku 2013 je výhradně sponzorován společností Pivotal[14]. Redis je napsán v jazyce C a vyznačuje se vysokým výkonem. Oficiálně je podporován pouze na UNIX platformě, ale existují projekty pro verzi Windows.

Nejedná se o klasickou databázi typu key/value, kromě jednoduchého ukládání

[12] `http://basho.com/riak-users/`
[13] `http://redis.io/`
[14] `http://redis.io/topics/sponsors`

řetězce k danému klíč umožňuje ukládat i složitější struktury. Také se odlišuje jiným přístupem uchovávání dat. Redis si totiž veškerá svá data drží v RAM paměti. Díky tomu, že přístup k datům v paměti bývá řádově rychlejší než čtení uložených dat z disku, se stává jednou z nejrychlejších databází ve svém oboru. Využívají ho společnosti jako je Twitter, GitHub, StackOverflow a další[15].

Na svých domovských stránkách poskytuje Redis výbornou dokumentaci. Ta je velmi jednoduchá, přehledná a také kompletní, často bývá doplněna o komentáře uživatelů a interaktivní možnost vyzkoušení si příkazů přímo na dané stránce. K dispozici je i jednoduchý interaktivní tutoriál příkazové řádky Redisu[16].

Struktura dat

Redis podporuje na rozdíl od jiných NoSQL databází typu key/value pět datových typů. Všechna data jsou uložena do společného prostoru a rozlišována jsou díky klíčům.

- **String** (řetězec) – jedná se o nejjednodušší strukturu, která může obsahovat jakýkoliv druh dat, např. serializovaný objekt nebo JPEG. Jeho velikost je omezena na 512 MB. S řetězci je možné provádět základní operace. Redis také umožňuje provádět základní matematické operace, jako je zvýšení či snížení uložené hodnoty pomocí příkazů INCR, DECR a INCRBY. Pokud jsou tyto příkazy použity, pak se k řetězci přistupuje jako k číslu, provede se nad ním požadovaná operace a opět se uloží jako řetězec.

- **List** (seznam) – List v Redisu je jednoduchý seznam řetězců seřazený podle pořadí vložení. Maximální velikost seznamu je $2^{32}-1$ prvků. Je možné přidávat nový řetězec na začátek (LPUSH) nebo konec seznamu (RPUSH). Seznamy dokáží přistupovat velmi rychle k datům ze začátku či konce seznamu. Naopak, pokud je potřeba přistupovat k prvkům z prostředku velmi velkého seznamu, je poté časová složitost operace $O(N)$.

- **Set** (množina) – je to neseřazená kolekce řetězců, do které je možno přidávat, odebírat a testovat na existenci členy s časovou složitostí $O(1)$. Maximální počet prvků je $2^{32}-1$. Množina nepodporuje duplicitní členy, pokud se tedy jeden prvek přidá vícekrát, bude množina obsahovat stále jen jednu kopii tohoto prvku. Nad množinami je možno provádět mnoho operací, jako je sjednocení (SUNION), průnik (SINTER) nebo rozdíl (SDIFF).

[15] `http://redis.io/topics/whos-using-redis`
[16] `http://try.redis.io/`

- **Hash** (mapa) – hash je v Redisu mapa a jedná se o nejkomplexnější datový typ. Používá se především k ukládání objektů. Mapa je vlastně kolekce, kde je pod daným klíčem uložena dvojice pole a hodnota. Tento způsob ukládání objektů zabírá jen velmi malou část paměti, proto je dobré používat mapy, kdykoliv je to možné. Každá mapa je schopna uložit $2^{32}-1$ dvojic polí a hodnot.
- **SortedSet** (setříděná množina) – setříděná množina se liší od jednoduchých množin v tom, že k prvkům se přidává ještě tzv. skóre. To je číselná hodnota, podle které jsou pak záznamy seřazené. Díky tomu je možné vybírat například množiny spadající do určitého intervalu skóre.

Redis podporuje tzv. expiraci klíčů u všech podporovaných datových struktur. Každému klíči je možné zadat tzv. délku jeho života pomocí příkazu EXPIRE a počtem sekund. Po uplynutí této doby je klíč z databáze automaticky odstraněn. Životnost klíče je vrácena pomocí příkazu TTL (time to live). [Red12]

Perzistence dat

Databázový systém Redis si udržuje celou databázi v operační paměti RAM. V případě výpadku serveru by mohlo ale dojít ke ztrátě dat, aby byla ztráta dat co nejmenší, ukládá si Redis obraz paměti na pevný disk. Redis podporuje dva druhy ukládání dat, RDB a AOF[17], které je možné zkombinovat.

RDB (Random Database File) funguje tak, že se v určitých intervalech uloží aktuální kopie databáze (snapshot) na disk. Tento způsob je velmi vhodný pro obnovu dat po havárii a pro klasické zálohování dat. Nehodí se však v případě, kde je i minimální ztráta dat od posledního zálohování nepřípustná. Standardně je nastaveno ukládání po 60 vteřinách a při 10 000 změnách klíčů, nebo po každých pěti minutách a deseti změněných klíčích, případně za 15 minut a jedné změně klíče.

AOF (Append Only File) je bezpečnější alternativou pro zálohování dat. Umožňuje zapnout logování všech operací ovlivňující data v databázi do logovacího souboru na pevném disku. V případě, že dojde k obnově dat, jsou všechny příkazy z logu načteny a provedeny. Tím je databáze obnovena do posledního známého stavu. Nevýhodou AOF je, že logovací soubor bývá často větší než ekvivalentní RDB soubor. Kvůli tomu bývá obnova dat pomocí AOF výrazně pomalejší než při obnově dat v RDB. [Red12][Red14]

[17] `http://redis.io/topics/persistence`

Replikace

Redis používá master-slave replikaci, která byla popsána v kapitole 4.5. Replikace je asynchronní a neblokující a je vhodná pro zajištění redundance dat a také pro škálovatelnost. Díky asynchronní replikaci je možné normálně pracovat s databází, zatímco jsou data synchronizována s jiným uzlem. Sekundární uzel může být hlavním uzlem pro další uzly a vytváří se tak stromová struktura. V předchozích verzích Redisu bylo umožněno zapisovat data i na sekundární uzly, které se ale při synchronizaci s hlavním uzlem přepsaly. Od verze 2.6 jsou sekundární uzly defaultně nastaveny pro čtení. Replikace se spravuje příkazy SYNC a SLAVEOF pro určení primárního a sekundárního uzlu. U sekundárních uzlů je možné nastavit prioritu, podle které při výpadku hlavního uzlu vybere Redis Sentinel nový primární uzel. Redis Sentinel je zatím vyvíjený systém pro správu několika běžících instancí Redisu. Používá se pro monitorování, notifikaci, automatické povýšení sekundárního uzlu na primární, pokud je hlavní uzel nedostupný. [Red12][Seg12]

V nové beta verzi 3.0.0 poskytuje Redis možnost použití Redis Cluster, který by poskytoval řešení pro automatické shardování dat přes několik Redis uzlů. Pro možnost ukládání distribuovaných dat je také možnost řešení na straně klienta, který díky svému algoritmu může rozhodovat o instanci, na kterou budou data uložena. Třetí možností rozdělení dat je Twemproxy vyvinutý pro Twitter. Twemproxy je mezivrstva mezi klientem a instancí Redisu . Klient tak nekomunikuje přímo s Redisem ale s proxy, která přeposílá požadavky pro daný uzel. [Red14]

Systémové rozhraní

Redis používá klasickou architekturu klient-server a komunikuje pomocí protokolu TCP/IP. Standardní instance serveru běží na portu 6379. Server tvoří vždy jednovláknový proces a klient je v základu reprezentován příkazovou řádkou Redis-cli. Alternativní klienty lze implementovat pomocí četných knihoven, v různých programovacích jazycích, jejichž seznam se nachází na domovských stránkách Redisu. K dispozici je i velké množství nástrojů a doplňků.

Stejně jako některé jiné databáze umožňuje Redis volání vlastních skriptů nad instancí databáze. Skripty musí být napsány v jazyce Lua. V konfiguračním souboru lze také nastavit počet databází. Defaultní počet je nastaven na šestnáct. Je vhodné počet databází změnit před prvním spuštěním serveru a dále ho již neměnit. Pokud se pokusí Redis po opětovném spuštění načíst data z dump.rdb, který má data uložena pro jiný počet databází, než má aktuální instance Redisu, nejsou data korektně

načtena. [Red14]

Rozhraní pro programování aplikací

API neboli rozhraní pro programování aplikaci poskytuje Redis pro širokou škálu programovacích jazyků[18]. Oficiálním klientem pro jazyk C je hiredis, který podporuje všechny poskytované příkazy. Doporučené knihovny pro jazyk C# jsou knihovna ServiceStack.Redis a nově vyvinutý StackExchange.Redis, který nahradil původní projekt BookSleeve. Pro jazyk Java je k dispozici knihovna Jedis a Reddisson. Uvedeny jsou jen hlavní zástupci pro nejpoužívanější jazyky, na stránkách Redisu je ale mnohem větší seznam poskytující odkazy na další projekty a knihovny.

Redis podporuje mnoho příkazů. Všechny jsou uvedeny na domovských stránkách s podrobným popisem[19]. Následuje přehled některých základních příkazů, jež jsou součástí většiny poskytovaných API. Příklady jsou psány ve formátu pro příkazovou řádku Redis-cli.

```
>SET key1 Hello
>GET key1
```

Výše uvedený příklad zobrazuje přidání a následné vyhledání řetězce označeného klíčem key1 s hodnotou Hello. Pokud se provedla operace správně a nevrací žádnou hodnotu, je vypsána hodnota OK.

```
>LRANGE mylist 0 -1
```

Tento příkaz umožňuje vypsat celý seznam. Podle zadaného intervalu je možné vypsat jen některé hodnoty seznamu.

Do množiny se nové hodnoty přidávají příkazem SADD. K výpisu obsahu množiny pak slouží příkaz SMEMBERS. Podobným příkazem je ZADD u seřazené množiny, nutné je však k němu zadat číselné skóre. Možné je zadávat současně i více dvojic skóre hodnota. Následně se seřazená množina vypíše obdobným příkazem, jako je u seznamů. Příznak WITHSCORES umožňuje vypsat také skóre.

```
>ZADD myzset 1 one
```

[18] `http://redis.io/clients`
[19] `http://redis.io/commands`

```
>ZADD myzset 2 two 1 uno
>ZRANGE myzset 0 -1 WITHSCORES
1) "one"
2) "1"
3) "uno"
4) "1"
5) "two"
6) "2"
```

Pro vytvoření nového hash klíče je používán příkaz HSET, u kterého se musí zadat jméno klíče, jméno hash klíče a hodnota. Pro výběr klíče slouží příkaz HGET. HGETALL je pro vypsání všech hash klíčů a hash hodnot uložených pod daným klíčem.

```
>HSET user name Karel
>HSET user surname Novak
>HGETALL user
1) "name"
2) "Karel"
3) "surname"
4) "Novak"
```

Dále je uvedeno několik dalších zajímavých a užitečných příkazů, které Redis podporuje.

BGREWRITEAOF Asynchronně přepíše append-only soubor.

CONFIG GET Vrátí hodnotu konfigurace podle zadaného parametru.

EXPIRE Nastaví čas života klíče ve vteřinách.

MIGRATE Atomicky přesune klíč z jedné instance Redisu do druhé.

PING Ping serveru.

RENAME Přejmenuje klíč.

RPOPLPUSH Vezme poslední prvek v seznamu, přidá ho do jiného a vypíše jej.

SAVE Synchronně uloží data na disk.

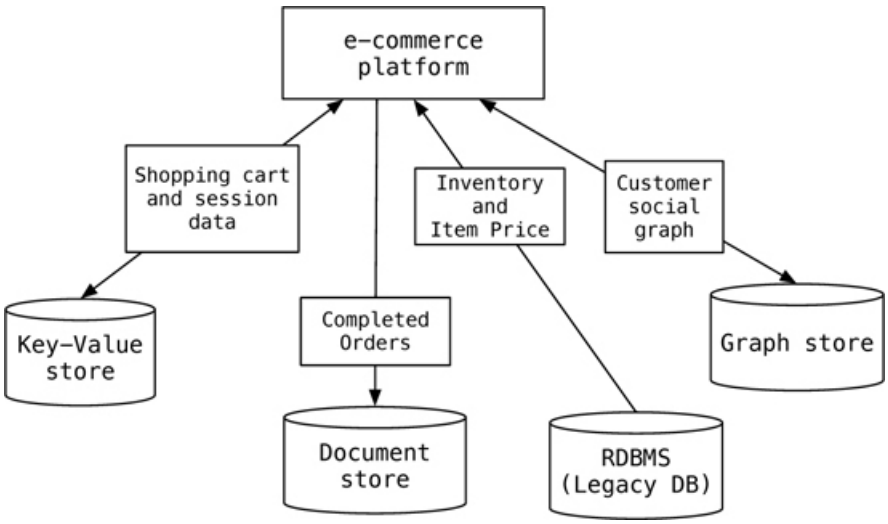
SHUTDOWN Synchronně uloží data na disk a pak se vypne server.

TTL Získá zbývajíc čas života klíče.

5 Porovnání NoSQL databází s relačními databázemi

Relační databáze využívající dotazovací jazyk SQL jsou nejpoužívanějším dnešním typem databází. Prošly si dlouhým vývojem a dnes jsou tak na vysoké úrovni poskytovaných služeb a spolehlivosti. Za jejich největší přednost je považovaná schopnost snadné transformace objektů z reálného světa do tabulek a relací. U relačních databází je díky transakčnímu zpracování zajištěna silná konzistence dat. To se například hodí pro klasické bankovní systémy, kde je vyžadováno dodržení ACID vlastností. Problém s relačními databázemi může nastat, pokud je potřeba zpracovávat velké množství různých dat, která mohou být ještě distribuovaná. Vzrůstají tak požadavky na výkon, škálovatelnost a flexibilitu. Řešením pak může být složité řízení shardování v aplikaci nebo pořízení drahého SQL řešení, ani jedna z možností není ideální. Dalším problémem jsou relace mezi více tabulkami. Pro propojení více tabulek se používá SQL příkaz JOIN. Pokud se tento příkaz příliš využívá, může docházet k výraznému snížení rychlosti vykonávání dotazu nad více tabulkami. V těchto případech se jeví výhodnější přejít k NoSQL databázím, které tyto problémy řeší jednoduše a efektivně. [Sto10]

NoSQL databáze byly vytvořeny na základě potřeby zpracovávat a uchovávat velké množství nestrukturovaných dat, která stále narůstají. Jsou vhodné především v oblastech, kde je velká pracovní zátěž, ale dotazy na data nejsou náročné. Příkladem použití mohu být například údaje o uživatelích ze sociální sítě, různé sociální grafy atd. Data mohou mít v dnešní době různou strukturu, kterou relační databáze kvůli svému předdefinovanému schématu neumí uchovat. Na rozdíl od toho NoSQL databáze mají dynamické schéma právě pro nestrukturovaná data. Díky jinému přístupu jsou NoSQL databáze horizontálně škálovatelné, čímž umožňují distribuovat data na více serverů a rozloží tak zatížení jednotlivých uzlů. Relační databáze jsou především vertikálně škálovatelné, a tudíž u nich narůstají požadavky na výkon serveru. Nevýhodou NoSQL databází bývá jejich jednoduchý model, nestandardní dotazovací jazyky oproti SQL a také jejich neznalost mezi programátory. Dalším negativem některých NoSQL databází je to, že nepodporují integritní omezení a ACID. [Cat11]



Obrázek 5.1: Ukázka použití několika databázových systémů v jednom projektu. [Sad12]

Jak relační, tak NoSQL databáze mají v dnešní době svá opodstatnění v určitých případech použití. NoSQL nemá za úkol nahradit relační databáze. Je to vlastně jen alternativní řešení, které je v některých případech výhodnější. U větších aplikací může ale dojít k tomu, že např. na některá data je vhodná relační databáze, na další data dokumentová NoSQL a na úplně jiná data třeba key/value databáze. Potom se takovýto přístup, kdy se kombinuje v jednom projektu použití více databází, znázorněný na obrázku 5.1, označuje jako polyglot persistence. [Aug11]

6 Analýza problému

V této části diplomové práce se rozebírají možnosti pro tvorbu správce NoSQL databáze. Úkolem je pro vybranou databázi vytvořit správce, který bude poskytovat uživatelsky příjemnou práci s uloženými daty. Zde jsou popsány možnosti výběru typu NOSQL databáze a příslušného systému. Dále je věnována část textu porovnání webového nebo desktopového rozhraní. Následně je uvedeno několik existujících řešení, která jsou vůči výslednému programu referenční, a z nich vyplývající funkce výsledné aplikace.

6.1 Typ NoSQL databáze

Ve výše zmíněné podkapitole 4.10, byly uvedeny čtyři kategorie NoSQL databází. Každá kategorie má svá vlastní specifika a možnosti použití. Již na začátku práce byla pozornost věnována především databázím typu key/value. Grafové a dokumentově orientované databáze jsou v dnešní době totiž velmi oblíbené a tudíž existuje i velká řada správců těchto databázových systémů. Pro grafovou databázi Neo4j existuje např. program Gephi[1] nebo Linkurious[2], u MongoDB je to třeba program Fluentd[3] a mnoho dalších. Zbývající dva druhy NoSQL, key/value a sloupcové databáze, jsou si ve své podstatě velmi podobné, pouze sloupcové databáze mají o trochu složitější datový model. Pro tvorbu správce byla zvolena NoSQL databáze typu key/value, především kvůli menšímu počtu kvalitních správců, a také kvůli svému jednoduchému modelu, jenž plně charakterizuje princip NoSQL databází.

6.2 Databázový systém pro key/value databázi

Pro databáze typu key/value existuje mnoho databázových systémů. Mezi dva nejpoužívanější open-source systémy patří již zmíněný Riak a Redis. Hlavní výhoda Redisu je především jeho vysoká rychlost, nicméně Riak poskytuje zase lepší odolnost vůči pádu serveru. Oba dva systémy poskytují na svých domovských stránkách podrobnou dokumentaci. Riak poskytuje oficiální knihovny jen pro jazyky Erlang, Java, PHP, Python a Ruby. Předností Redisu je především, rychlost, podpora růz-

[1] `http://gephi.org/`
[2] `http://linkurio.us/`
[3] `http://fluentd.org/`

ných struktur dat a velká škála kvalitních knihoven pro programovací jazyky. Největší výhoda Redisu je však podpora velkého množství příkazů, které činí práci s různými druhy dat velmi jednoduchou. V tabulce 6.1 jsou znázorněny základní vlastnosti Redisu a Riaku. Pro tvorbu správce byl nakonec zvolen systém Redis, který je používanější a výkonnější než Riak.

Jméno	**Riak**	**Redis**
Popis	Distribuované, vůči chybám odolné key/value uložiště	In-memory databáze s možností nastavení výkonu oproti perzistenci
Licence	Open-source – Apache verze 2	Open-source – BSD
Typ databáze	Key/value	Key/value
API a ostatní přístupové metody	HTTP/REST a nativní Erlang rozhraní	Telnet / Proprietární
Replikace	P2P	Master-slave replikace
Podporované programovací jazyky	C, C#, C++, Erlang, Java, JavaScript, Lisp, Perl, PHP, Python, Ruby, Smalltalk a další	C, C#, C++, Erlang, Java, JavaScript, Lisp, Lisp, Lua, Objective-C, Perl, PHP, Python, Ruby, Smalltalk, Tcl a další
Serverová skriptování	Erlang, JavaScript	Lua
Map Reduce	Ano	Ne
Hlavní přednosti	Odolný proti chybám	Velmi rychlý

Tabulka 6.1: Srovnání základních vlastností sytémů Riak a Redis. [Dbe14]

Jak již bylo zmíněno Redis i Riak podporují mnoho jazyků. Jazyk Java je multiplatformní, díky tomu lze programy napsané v tomto jazyce spouštět na libovolném operačním systému, kde běží Java Virtual Machine. Její nevýhodou je pomalejší start programů a větší paměťová náročnost. Jazyk C# má mnohem lepší a modernější možnosti zobrazení GUI (grafické uživatelské rozhraní). Nevýhodou jazyka C# je, že

aplikace v něm vytvořené jsou spustitelné defaultně pouze na systémech Microsoft Windows. PHP je jazyk především pro webové řešení. Posledním krátce zmíněným jazykem, který Redis podporuje, je jazyk C, který je velmi rozšířený, ale v některých ohledech je pro tvorbu GUI aplikací zbytečně nízkoúrovňový. Ostatní podporované jazyky nejsou moc rozšířené.

6.3 Webové vs. desktopové řešení

Základní otázkou bylo, jestli bude výsledný program webový nebo desktopový. V dnešní době je populárním řešením vytvořit webovou aplikaci, která je poskytovaná z webového serveru přes internet. Její největší výhodou je především možnost připojení se odkudkoliv prostřednictvím webového prohlížeče. S tím ale souvisí problém, že musí být k dispozici stálé připojení k internetu. Webové aplikace jsou v případě tenkého klienta nainstalovány pouze jednou na vzdáleném serveru. Výpočetní výkon díky tomu nezatěžuje počítač uživatele, protože probíhá vzdáleně. Pokud dojde k aktualizaci aplikace, nemusí se updatovat na každém počítači zvlášť, jako je tomu v případě desktopových aplikací. Webové aplikaci jsou však častěji vystaveny bezpečnostním rizikům. Mohou také kvůli zvýšenému provozu reagovat pomaleji než desktopové.

V případě, kdy bude potřeba např. zobrazit ve webové aplikaci záznamy z NoSQL databáze, která je schopna obsahovat miliony klíčů, by mohlo být načítání stránky pro uživatele až příliš zdlouhavé. Také, pokud jde o funkcionalitu, není webová aplikace schopna dosáhnout možností desktopových. Příkladem mohou být rozdíly u webových a desktopových manažérů pro správu relačních databází, kdy desktopová varianta poskytuje větší množství nástrojů pro správu, modelování a řízení dat. Na rozdíl od toho webová aplikace umožňuje klasické funkce, ale je limitována počtem a schopnostmi těchto funkcí. Desktopové aplikace jsou spolehlivé a důvěryhodné. I přes mnoho výhod, které poskytují webové aplikace, jsou stále používané především kvůli jejich uživatelsky příjemnějšímu a intuitivnějšímu rozhraní.

Díky získaným poznatkům byla zvolena desktopová verze manažéru, která má splňovat požadované vlastnosti. Výsledný správce by měl být pro NoSQL databáze obdobný jako manažér Oracle SQL Developer, který poskytuje velkou funkcionalitu a stabilitu a umožňuje rychle zpracovávat zadané úkoly, pro relační databáze. Snahou je vytvořit jednoduchou intuitivní aplikaci, která bude pro uživatele příjemně ovladatelná. Jelikož je stále mnoho běžných uživatelů navyklých používat klasické desktopové programy, byla vybrána právě tato varianta.

6.4 Existující řešení pro Redis

Pro zvolený databázový systém existuje již několik jednoduchých desktopových a webových manažerů. Mezi ně patří např. projekt Redsmin[4], který je orientovaný na vývojáře a umožňuje správu služeb pro Redis. Zpřístupněn byl na podzim v roce 2013. Je vytvořen jako komerční aplikace, kde je možnost si zdarma vyzkoušet verzi s připojením na jeden server. Poskytuje mnoho funkcí, např. pomocí grafu zobrazuje počet provedených operací za dané období, také poskytuje možnost používání příkazové řádky Redisu. Jeho hlavní nevýhodou je zobrazování klíčů a hodnot pomocí GUI, které je občas nepřehledné a nutí uživatele znát základní příkazy Redisu.

Dalším zkoumaným programem pro Redis byl jednoduchý desktopový manažér RedisConsole[5]. V roce 2011 ho vytvořil Petr Trofimov, jelikož nemohl najít žádný desktopový nástroj pro Windows, který by uměl pracovat s Redis databází. Program je napsán v jazyce C++ a založen na QT frameworku. Umožňuje zadat nastavení pro připojení k databázi a veškeré operace jsou u něj zadávány skrze příkazovou řádku. Zobrazuje jednoduchý seznam klíčů a ke každému klíči umožňuje zobrazit jeho hodnotu, která je pouze text ve formátu JSON. Tento program je ve výsledku pouze vzdálenou příkazovou řádkou Redisu.

Redis Desktop Manager[6] je multiplatformní open-source GUI manažér pro Redis vytvořený Igorem Malinovskiym na podzim roku 2013. Tento manažer splňuje mnoho funkcí, které z něho tvoří velmi kvalitní nástroj pro správu Redis databází. Projekt je stále ve vývoji a jeho poslední verze byla vydána v dubnu 2014. Podporuje správu připojení pomocí SSH protokolu k více serverům. Připojení jsou seřazena ve stromě, který dovoluje rozbalení a zobrazení jednotlivých databází a klíčů. V hlavním okně, jsou pak zobrazeny záložky, na kterých jsou hodnoty daného klíče a pomocí tlačítek je možno tyto hodnoty pohodlně upravovat. Dalším prvkem je Redis konzole, která podporuje většinu příkazů Redisu. Některé jednoduché operace jako např. přidání nového klíče, je možné provádět pouze přes konzoli. Současně s občasnou nestabilitou patří způsob přidávání nového klíče k jednomu z mála nevýhod tohoto nově vznikajícího nástroje.

[4] `https://redsmin.com/`

[5] `http://ptrofimov.wordpress.com/2011/06/19/redisconsole-windows-gui-tool-to-view-redis-databases/`

[6] `http://redisdesktop.com/`

6.5 Základní funkce správce

Z analýzy existujících řešení pro systém Redis byly vyvozeny základní funkce, které by měl výsledný správce umožňovat. Jak již bylo výše zmíněno, byla zvolena desktopová varianta s uživatelsky příjemným GUI, které by mělo umožňovat přidávat a spravovat nová připojení k databázovému systému. Také by měly být všechny dvojice klíčů a hodnot přehledně zobrazeny. Další vlastností, která by měla být určitě podporována, je přidávání, mazání a editace požadovaného klíče a hodnoty. S tím samozřejmě souvisí podpora všech pěti datových struktur, které Redis nabízí. Všechny zmíněné operace by měly být přístupné i uživateli, který nezná základní příkazy Redisu. Požadovanou funkcí na program, byla možnost exportovat a importovat data z databáze do formátu JSON. Uvedené funkce jsou pouze základními požadavky získanými z analýzy databázového systému a jeho manažérů.

7 Realizace správce databáze

V rámci této diplomové práce bylo navrženo a realizováno řešení správce NoSQL databází typu key/value pro databázový systém Redis, které se nazývá NoSqlManager. V následujícím textu je popsán jeho návrh a řešení.

7.1 Návrh správce databáze

Pro správce byl zvolen pouze jediný databázový systém, a to Redis, především kvůli neuniverzálnosti příkazů, které používají NoSQL databáze. Na rozdíl od relačních databází, které využívají jednotný jazyk SQL, u NoSQL zatím neexistuje standardizovaný přístup.

Jak již bylo zmíněno v analýze, podkapitola 6.5, vytvořený program bude poskytovat základní funkce. Těmi jsou přidávání nového klíče a hodnoty pro všechny struktury dat, editace hodnot klíčů a také odstranění požadovaného klíče. Další vlastností, kterou aplikace umožňuje, je vytvoření nového připojení, podle zadané IP adresy, jména serveru, portu, na kterém databáze běží, a hesla, které slouží k připojení zabezpečené databáze. Všechny údaje o připojení jsou uloženy, aby mohly být při dalším použití programu znovu načteny. Program také umožňuje zobrazit podrobné informace o serveru, jako je např. verze Redisu, počet připojených klientů nebo využití paměti. V průběhu realizace byla dodána ještě konzole, která umožňuje využívat veškeré příkazy Redis-cli. Konzole výrazně rozšiřuje možnosti použití správce a poskytuje znalému uživateli možnost plně využívat funkcionalitu Redisu.

7.1.1 Programovací jazyk

K napsání programu byl zvolen vysokoúrovňový objektově orientovaný jazyk C#, který byl vytvořen firmou Microsoft společně s platformou .NET. Na rozdíl do Javy je C# rychlejší a nezabírá tolik operační paměti. Hlavní motivací pro zvolení jazyka C# bylo zejména prohloubení znalostí v tomto jazyce a v používání vývojového prostředí Visual Studio. Pro tvorbu byly použity nové technologie, jako je např. WPF (Windows Presentation Foundation) se značkovacím jazykem XAML (Extensible Application Markup Language) umožňující data binding.

7.1.2 Rozvržení programu

Aplikace je vytvořena podle návrhového vzoru MVVM (Model View ViewModel) pro WPF aplikace. MVVM odděluje data, stav aplikace a uživatelské rozhraní. Řešení MVVM umožňuje, aby proces běžící na pozadí nebyl závislý na okně, které konzumuje informace tohoto procesu. Vrstva Model popisuje jednotlivá data, se kterými aplikace pracuje. View reprezentuje uživatelské rozhraní napsané v jazyce XAML. ViewModel spojuje vrstvy Model a View pomocí bindingu. Provádí se v něm filtrování dat podle stavu aplikace, a proto musí být implementováno rozhraní *INotifyPropertyChanged*.

Struktura projektu je rozdělena do čtyř složek, Helpers, Images, Model a View. View obsahuje ještě jednu vloženou složku pojmenovanou Partial. Podle vzoru MVVM jsou zde uloženy XAML a jejich code-behind (kód na pozadí) v příslušných CS (zdrojové soubory pro C#) souborech. Níže je stručně popsána charakteristika všech souborů v projektu NoSqlManager. Ve složce Images jsou obrázky používané u kontrolek v GUI. Soubory umístěné ve View jsou následující:

- `AddWindow` – slouží pro přidání nových dat do databáze.
- `ConnectWindow` – poskytuje možnost vytvoření nového připojení k serveru.
- `ConsoleWindow` – třída reprezentující okno a funkce konzole.
- `EditConnectWindow` – upravuje nastavení připojení k serveru.
- `EditValuesWindow` – umožňuje upravovat hodnoty klíčů rúzných struktur dat Redisu.
- `EditWindow` – poskytuje náhled na data, která mohou být dále editována.
- `InfoWindow` – zobrazuje informace získané od serveru, na kterém běží Redis.
- `MainWindow` – hlavní okno aplikace, které poskytuje přehledné zobrazení požadovaných prvků a základní logiku aplikace.

Ve složce Partial jsou umístěné XAML soubory, které se vkládají do jiných oken. `DatabaseGrid` slouží pro zobrazení dat z databáze v tabulce, která je umístěna v záložce na hlavním okně. `TabCloseButton` reprezentuje tlačítko na záložce, které umožní její zavření.

Model obsahuje objekty, které zastupují data, se kterými aplikace operuje. Dále je uveden jejich přehled.

- `CloseableTabItem` – je třída, reprezentující záložku, kterou je možné zavírat pomocí tlačítka popsaného v TabCloseButton.xaml.
- `DatabaseNode` – reprezentuje databázi příslušného serveru a její obsah.
- `DatabaseRow` – zastupuje celý řádek z databáze. Obsahuje, o jaký typ dat se jedná, klíč a hodnoty.
- `ITreeViewItem` – interaface pro stromovou strukturu, zobrazující servery a databáze.
- `IValue` – interface pro typ, klíč a hodnotu.
- `ServerNode` – představuje server, jeho nastavení a příslušné potomky, které jsou zastoupeny jako `DatabaseNode`. Také se zde udržuje informace o roli (slave nebo master) serveru.

Helpers je složka, která v sobě obsahuje různé třídy potřebné pro řešení některých úkolů.

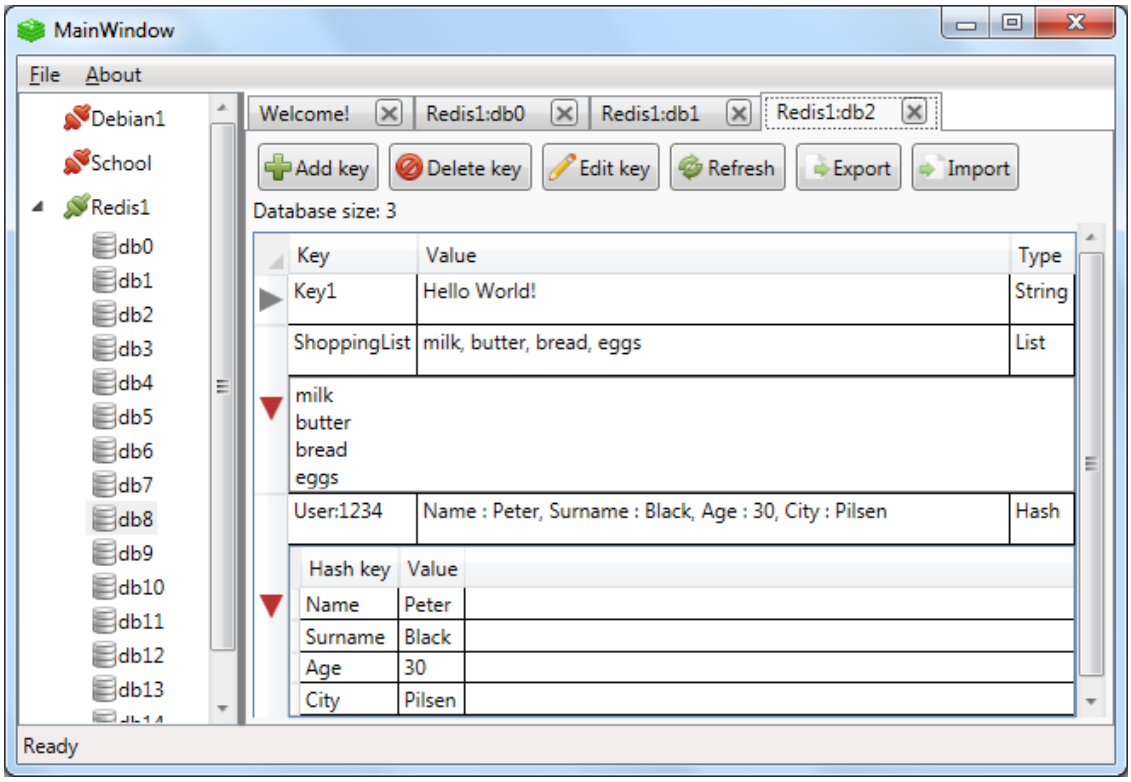
- `InverseBooleanConverter` – slouží pro inverzi Boolean hodnoty.
- `RedisBackgroundWorker` – patří k nejdůležitějším třídám, obsahuje logiku práce na pozadí při dotazování na server.
- `SerializeHelper` – slouží k uložení seznamu připojení do config.xml souboru.
- `TypeTemplateSelector` – pomáhá určovat strukturu výpisu v tabulce pro různé typy dat.
- `VisibilityToNullableBooleanConverter` – converter pro inverzi viditelnosti kontrolek.

V příloze B je zobrazen graf závislostí jednotlivých jmenných prostoru znázorňující výše popsané třídy a vztahy mezi nimi. Diagram byl vygenerován pomocí Visual Studia.

7.1.3 GUI

Rozložení jednotlivých části GUI bylo navrženo tak, aby byla uživateli poskytnuta pohodlná a intuitivní práce s manažerem. Inspirací pro umístění některých prvků

rozhraní byly existující nástroje, popsané v podkapitole 6.4. Na obrázku 7.1 je znázorněné výsledné grafické rozhraní vytvořené aplikace.

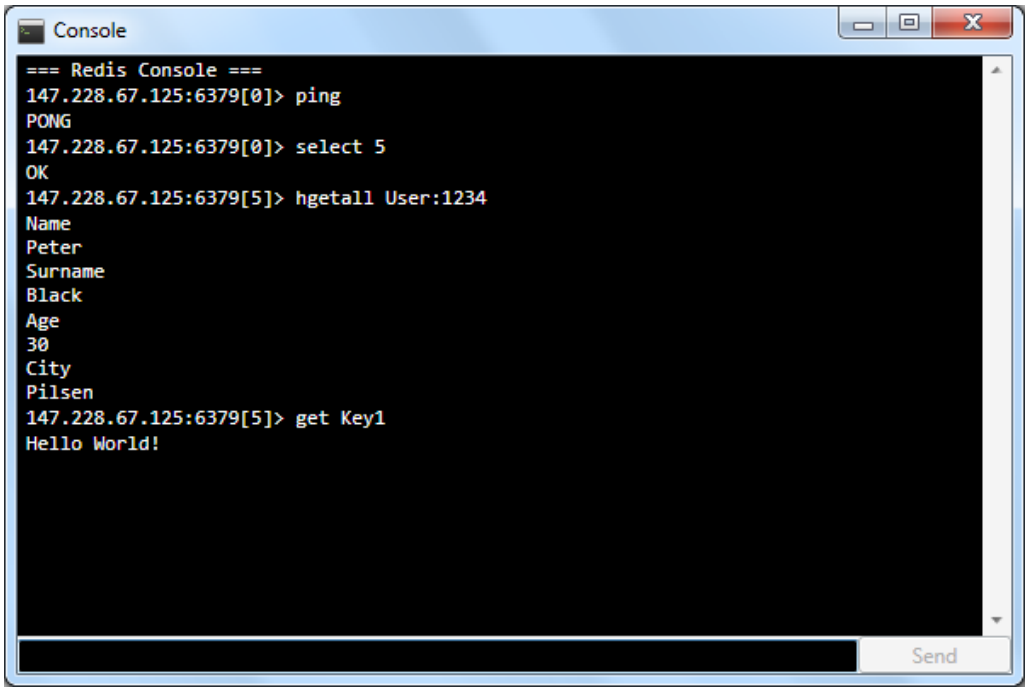


Obrázek 7.1: Hlavní okno programu NoSqlManager.

Program se skládá z hlavního okna, které je rozděleno na dvě základní části. V levé části je stromová struktura, ve které jsou zobrazeny jednotlivé servery se svými databázemi. Druhá část hlavního okna zobrazuje tabulku s dvojicemi klíč a hodnota, které jsou abecedně seřazené, pro zvolenou databázi. Pro možnost jednoduchého přepínání mezi databázemi a porovnávání obsahu, jsou použity záložky. Každá záložka obsahuje pouze data z jedné databáze. Na záložce jsou pak umístěna tlačítka pro přidání, smazání a editaci klíče, dále také pro aktualizaci dat a pro import/export dat do/z JSON souboru. Tlačítka jsou umístěna v horní části záložky, simulují tak nástrojovou lištu. Pod tlačítky je pak zobrazena tabulka, která je rozdělena na tři sloupce, klíč, hodnotu a typ klíče. V každém řádku je znázorněn náhled na hodnoty dat, a jelikož Redis podporuje pět různých datových struktur, je možno po rozkliknuti příslušného klíče zobrazit podrobnější strukturu jeho hodnot, např. u mapy je po kliknutí zobrazena další tabulka s hashkey a hodnotami.

V horní části hlavního okna je umístěna lišta, která pod nabídkou *File* posky-

tuje možnost *Add New Connection*. Pomocí nabídky se tak otevře okno pro přidání nového připojení. Nad názvy jednotlivých serverů ve stromové struktuře je možno pravým tlačítkem vyvolat kontextové menu, které nabízí otevření konzole nad daným serverem, editaci nastavení serveru, zobrazení základních informací o serveru, aktualizaci stavu serveru a také smazání nastavení serveru.



Obrázek 7.2: Konzole programu NoSqlManager.

Zvláštním oknem je konzole, zobrazena na obrázku 7.2, která zpřístupňuje uživateli příkazovou řádku Redis-cli. Skládá se ze dvou částí, první zobrazuje napsané příkazy a odpověď serveru. Spodní část reprezentuje pole, do kterého se zadávají příkazy. Pro přehlednost bylo přidáno tlačítko *Send* k odeslání příkazu. Pro poslání příkazu také stačí stisknout klávesu Enter. Uzavření okna lze provést příkazem *exit*. Ostatní okna aplikace, která se otevřou po zvolení jednotlivých nabídek nebo po kliknutí na tlačítko, a funkce programu jsou blíže popsány v uživatelské příručce, která je součástí přílohy.

7.2 Vývojové nástroje

Pro tvorbu programu bylo zvoleno vývojové prostředí Microsoft Visual Studio Ultimate 2013 s .NET Frameworkem 4.5.1., které je distribuováno společně s jazykem C#. Verze Redisu, pro kterou se program vyvíjel, je 2.8.2.

Ke správě projektu byl použit Team Foundation Server (TFS) od společnosti Microsoft, který umožňuje spravovat zdrojové kódy, podporuje verzování a mnoho dalších funkcí. TFS byl nasazen na externím serveru především pro potřeby zálohování projektu. Další výhodou takovéhoto přístupu je možnost při vývoji porovnat jednotlivé verze kódu mezi sebou. Pro každý commit provedený na server je přidán krátký popis provedených změn.

V projektu jsou používané různé knihovny. Pro umožnění exportování a importování dat do JSON souborů byla použita knihovna Json.net. Knihovna Extended WPF Toolkit byla použita pouze pro jedinou kontrolku IntegerUpDown. Ta se používá v okně pro přidání nového připojení u nastavení čísla portu.

Pro API k Redisu je použitá knihovna ServiceStack.Redis[1]. Druhá knihovna, která byla uvažována, byla BookSleeve[2]. Její hlavní nevýhodou byla téměř neexistující dokumentace. Na jaře 2014 byla nahrazena knihovnou StackExchange.Redis[3], která již poskytuje základní dokumentaci. Zvolená knihovna ServiceStack.Redis poskytuje velmi dobrou dokumentaci a její další předností je oblíbenost u komunity, která poskytuje mnoho rad v této oblasti. ServiceStack.Redis je nabízena pod licencí GNU Affero General Public License, verze 3[4]. Původně byla v projektu zahrnuta pomocí NuGetu (open-source balíčkovací systém pro platformu .NET). Při vývoji konzole bylo však potřeba upravit zdrojový kód knihovny a zpřístupnit tak některé metody. Proto bylo nutné do projektu knihovnu přidat jako další projekt. S ní byly přidány i další potřebné knihovny od ServiceStack[5]. Těmi jsou ServiceStack.Text, ServiceStack.Common a ServiceStack.Interfaces.

[1] `https://github.com/ServiceStack/ServiceStack.Redis`
[2] `https://code.google.com/p/booksleeve/`
[3] `https://github.com/StackExchange/StackExchange.Redis`
[4] `http://www.gnu.org/licenses/agpl-3.0.html`
[5] `https://github.com/ServiceStack`

7.3 Připojení k databázi

Vytvořený správce NoSqlManager umožňuje připojit se k serveru s běžícím systémem Redis pomocí knihovny ServiceStack.Redis. Nejprve se vytvoří objekt `RedisClient`, jehož vstupní parametry konstruktoru jsou `Host` – adresa serveru, `Port` – číslo portu, na kterém běží Redis, a `Auth` – heslo, které se používá, pokud je instance Redisu zabezpečena. Pro jednodušší práci s objektem je použité klíčové slovo `using`, které se používá pro korektní uvolňování zdrojů.

```
1 using (var redisClient = new RedisClient(Host, Port, Auth))
2 {
3     var info = redisClient.Info;
4 }
```

Pokud se provede dotazování na server, které může trvat určitý čas, mohlo se stát, že by aplikace tzv. zamrzla a uživatel by se poté mylně domníval, že program nefunguje. Pro tento případ je ošetřena veškerá komunikace s databází pomocí vláken. Při jakémkoliv dotazování na databázi se vytvoří nové vlákno, které obsluhuje příslušný dotaz, neboli pracuje na pozadí. Mezitím hlavní vlákno aplikace stále umožňuje využívat aplikaci, i když jen v omezené míře, např. některá tlačítka jsou v průběhu práce vedlejšího vlákna nedostupná.

```
1  RedisBackgroundWorker.BackgroundAction = delegate
2  {
3      using (var redisClient = new RedisClient(Host, Port, Auth))
4      {
5          var info = redisClient.Info;
6      }
7  };
8
9  RedisBackgroundWorker.OnActionFail = ...;
10 RedisBackgroundWorker.OnActionComplete = ...;
11
12 RedisBackgroundWorker.Start("Information about server " + server.Name + " ...");
```

K obsluze práce na pozadí slouží třída `RedisBackgroundWorker`, která ve svém konstruktoru nastavuje informativní hlášku zobrazenou v dolní liště hlavního okna. Pro spuštění nového vlákna je zde metoda `Start`, která se provede, jestliže již neběží jiná práce na pozadí. Obě dvě vlákna pak běží asynchronně.

Jelikož není ve vícevláknových aplikací povoleno zavolat metodu pro ovládací prvek jiným vláknem než tím, které prvek vytvořilo, je nutné použít metodu `Dispat-`

cher.Invoke. Pomocí ní se může přesměrovat volání metody z pracovního vlákna na hlavní. BackgroundWorker je pomocná třída, která obaluje pracovní vlákno a automaticky volá podle potřeby Dispatcher.Invoke. Aby se mohl použít BackgroundWorker, bylo nutné vytvořit handler pro událost DoWork a zavolat metodu RunWorkerAsync, která spustí instanci BackgroundWorkeru. Událost RunWorkerCompleted, která je také součástí BackgroundWorkeru, se provede, když DoWork dokončí svojí práci.

Pokud dojde během vykonávání DoWork k chybě, např. server, na který je dotazováno, je nedostupný, je ošetřena výjimka a uživatelovi je zobrazeno chybové okno s hláškou, kterou vrací přímo systém. Následně je výjimka předána zpět volajícímu vláknu. Díky tomu je zajištěno její snadnější ošetření. Chybová hláška je prezentována v jazyce systému, ve kterém je program spouštěn.

```
1  public void Start(string statusMessage)
2  {
3      if (!_backgroundWorker.IsBusy)
4      {
5          _backgroundWorker = new BackgroundWorker();
6          MainWindow.Status = statusMessage;
7
8          _backgroundWorker.DoWork += delegate
9          {
10             try
11             {
12                 BackgroundAction.Invoke();
13             }
14             catch (Exception e)
15             {
16                 MainWindow.Dispatcher.Invoke(delegate
17                 {
18                     ...
19
20                     MessageBox.Show(e.GetBaseException().Message, "Error",
                           MessageBoxButton.OK, MessageBoxImage.Error);
21                  });
22             }
23         };
24
25         _backgroundWorker.RunWorkerCompleted += delegate
26         {
27             MainWindow.IsWorkOnBackground = false;
28             MainWindow.Status = DefaultStatus;
29             OnActionFail = null;
30
31             if (OnActionComplete != null)
```

```
32              {
33                  OnActionComplete.Invoke();
34              }
35              OnActionComplete = null;
36          };
37
38          MainWindow.IsWorkOnBackground = true;
39          _backgroundWorker.RunWorkerAsync();
40      }
41      else{...}
42  }
```

Také je možno zadat další chování aplikace při vyvolání výjimky pomocí akce `OnActionFail`, která např. při neúspěšném připojení k databázi nastaví roli serveru jako nepřipojený a zavře záložky, které náleží nedostupnému serveru. Obdobně je po úspěšném ukončení práce vlákna možnost zavolat akci `OnActionComplete`, která provede další činnost, jako je třeba aktualizace dat v databázi.

```
1   RedisBackgroundWorker.OnActionFail = delegate
2   {
3       Role = RoleNotConnected;
4       var tabs = Children.Select(child =>
5                       RedisBackgroundWorker.MainWindow.TabList
6                           .FirstOrDefault(m => m.HeaderText == Name + ":" + child.
                                Name))
7                           .Where(tab => tab != null);
8
9       foreach (var tab in tabs)
10      {
11          RedisBackgroundWorker.MainWindow.TabList.Remove(tab);
12      }
13      Children.Clear();
14  };
15
16
17  RedisBackgroundWorker.OnActionComplete = dbNode.Refresh;
```

Výše popsaný způsob umožňuje hlavnímu vláknu stále komunikovat s uživatelem. Nedochází tak k zbytečnému zamrznutí aplikace, jež může být pro uživatele nepříjemné.

7.4 Konzole

Pro poskytnutí kompletní funkčnosti správce byla do projektu během vývoje přidána konzole. Konzole podporuje veškeré příkazy Redisu, např. vypnutí serveru s databází příkazem SHUTDOWN, proto je lepší, když jí používá jen zkušený uživatel. Jedná se o velmi užitečný nástroj, díky kterému se může NoSqlManager řadit mezi nejúčinnější správce pro systém Redis.

Konzole je tvořena `ConsoleWindow.xaml` a jejím code-behindem. Vstupním parametrem konstruktoru je objekt ServerNode určující server, nad kterým byla konzole zavolána. Popsání logiky získávání odpovědi od serveru je uvedeno ve třídě `ConsoleContent`, která implementuje interface `INotifyPropertyChanged`. Metoda `RunCommand` v této třídě se zavolá po kliknutí na tlačítko `Send`, nebo stisknutím klávesy Enter. Nejprve se přidá na výstup prompt obsahující adresu serveru, port, číslo databáze (na počátku nastaveno defaultně na nulu) a zadaný příkaz uživatele.

Následně je zavolán `RedisBackgroudWorker`, viz předchozí kapitola, a dochází k připojení k serveru. Nad `redisClient` se zavolá `CreatePipelineCommand`, tím se připraví prostředí pro odeslání příkazu. Poté je pomocí regulárního výrazu rozdělen vstupní řetězec, včetně správného určení jednotlivých částí příkazu podle uvozovek. Zpracovaný příkaz je následně převeden na pole bytů pomocí `ToUtf8Bytes`. Takto upravený příkaz je zapsán do vyrovnávací paměti metodou `WriteAllToSendBuffer` a zaslán na server pomocí `FlushSendBuffer`.

```
1  redisClient.CreatePipelineCommand();
2  var regex = new Regex(@"((""((?<token>.*?)(?<!\\)"")|(?<token>[\S]+))(\s)*)");
3  var cmd = new List<string>();
4  foreach (Match m in regex.Matches(ConsoleInput))
5  {
6      if (m.Groups["token"].Success)
7                  cmd.Add(m.Groups["token"].Value);
8  }
9
10 var commands = cmd.Select(c => c.ToUtf8Bytes());
11
12 redisClient.WriteAllToSendBuffer(commands.ToArray());
13 redisClient.FlushSendBuffer();
```

Pro možnost přečíst odpověď, kterou server pošle zpět správci, bylo nutné provést menší změny ve zdrojovém kódu knihovny ServiceStack.Redis. Bylo nutné zpřístupnit private a protected metody z třídy `RedisNativeClient` umístěné v `RedisNativeClient_Utils.cs`. Všechny níže uvedené metody z knihovny byly pře-

psány na public.

```
private int SafeReadByte()
protected string ReadLine()
private byte[] ParseSingleLine(string r)
protected public string SendExpectString(params byte[][] cmdWithBinaryArgs)
private byte[] ReadData()
private byte[] ParseSingleLine(string r)
```

Pomocí metody `SafeReadByte` je přečtena odpověď serveru, která je pomocí switche zpracována dál. Kód, který je uvnitř bloku switch, je inspirován obsahem jednotlivých metod ze třídy `RedisNativeClient`. Pokud je v odpovědi na začátku znak „$“ a metoda `ReadLine` vrátí hodnotu „-1“, je na výstup konzole přidán výpis (nill). Pokud nevrátí „-1“, zavolá se metoda `ParseSingleLine` s odpovědí serveru jako parametrem. Výsledek je pak přidán na výstup do konzole.

V případě, že odpověď začíná symbolem „:“, „-“ nebo „+“, dojde k zavolání metody `ReadLine` a její výstup je předán na výstup konzole. Pokud odpověď začíná symbolem „*“, opět se zavolá metoda `ReadLine` a její výsledek se dále zpracovává. Pomocí `int.TryParse` se získá počet částí, které byly vráceny metodou `ReadLine`. Je-li počet roven nule, na konzoli je přidán výpis ”(empty list or set)” a znamená to, že dotaz do databáze nenašel pod požadovaným klíčem žádný seznam ani množinu. Při kladném počtu je výsledek převeden na řetězec a přidán na výstup konzole. Pomocí příkazu `dispatcher.Invoke(() => ConsoleOutput.Add(output));` je výstup pro konzoli předán hlavnímu vláknu, a uložen do `ConsoleOutput`. Díky změně této vlastnosti je pak zobrazena odpověď serveru na konzoli.

```
1  string readLine;
2  var redisAnswer = redisClient.SafeReadByte();
3  switch (redisAnswer)
4  {
5      case '$':
6          readLine = redisClient.ReadLine();
7          if (readLine == "-1")
8          {
9              dispatcher.Invoke(() => ConsoleOutput.Add("(nill)"));
10         }
11         else
12         {
13             var parsingLine = new byte[2][];
14             parsingLine[1] = redisClient.ParseSingleLine(string.Concat(char.
                   ToString((char)redisAnswer), readLine));
15
16             var output = string.Join("\n", parsingLine.Where(m => m != null).
```

```
                    Select(b => b.FromUtf8Bytes()));

                dispatcher.Invoke(() => ConsoleOutput.Add(output));
            }
            break;
        case ':':
        case '-':
        case '+': readLine = redisClient.ReadLine();
            dispatcher.Invoke(() => ConsoleOutput.Add(readLine));
            if (cmd.First().Equals("select", StringComparison.
                InvariantCultureIgnoreCase) && (cmd.Count == 2))
            {
                _dbNumber = cmd.Last().ToInt64();
            }
            break;
        case '*':
            int count;
            readLine = redisClient.ReadLine();
            if (int.TryParse(readLine, out count))
            {
                if (count == -1)
                {
                    throw new Exception();
                }
                if (count == 0)
                {
                    dispatcher.Invoke(() => ConsoleOutput.Add("(empty list or set)
                        "));
                }
                else
                {
                    var result = new byte[count][];

                    for (int i = 0; i < count; i++)
                              result[i] = redisClient.ReadData();

                    var output = string.Join("\n", result.Where(m => m != null).
                        Select(b => b.FromUtf8Bytes()));

                    dispatcher.Invoke(() => ConsoleOutput.Add(output));
                }

            }
            break;
}
```

7.5 Import a export JSON souborů

Za základní funkci programu byla již na začátku vývoje programu stanovena možnost exportovat data ze správce do JSON souboru a také funkce, která umožní zpětné importování dat ze souboru. V programu je tato funkcionalita řešena za pomoci knihovny *Json.net*. Po kliknutí na tlačítko `Export` je nejdříve vytvořena instance dialogu pro uložení. Automaticky je k názvu souboru přidána přípona *.json*.

```
1 var dlg = new Microsoft.Win32.SaveFileDialog
2 {
3     FileName = "",
4     DefaultExt = ".json",
5     Filter = "JSON text documents (.json)|*.json"
6 };
```

Pro zápis do souboru je použit blok `using` s novou instancí `StreamWriteru`, který slouží k zapisování do textových souborů. Dále je podobným způsobem použit `JsonTextWriter`, kterým je za pomoci objektu `JsonSerializer` uložen obsah databáze do `StreamWriteru`. Do souboru se vlastně zapisují objekty `DatabaseRow`.

```
1 using (var sw = new StreamWriter(filename))
2 {
3     using (JsonWriter jw = new JsonTextWriter(sw))
4     {
5         jw.Formatting = Formatting.Indented;
6         var serializer = new JsonSerializer();
7         serializer.Serialize(jw, dbNode.ContentOfDb);
8     }
9 }
```

Jelikož je požadováno uložení jen klíče, typu a hodnoty, je u vlastnosti `SetList`, `HashList` a `Value` použit atribut serializace `[JsonIgnore]`, díky kterému nebudou zvolené vlastnosti serializovány. U vlastnosti `ValueData` je uveden atribut `[JsonProperty("Value")]`, který slouží k přejmenování názvu proměnné ve výsledném JSON souboru.

```
[JsonProperty("Value")]
public T ValueData { get; set; }
```

Při importu dat z JSON souboru je postup trochu složitější. Nejprve se vytvoří `OpenFileDialog`, který umožňuje vybrat pouze soubory s příponou *.json*. Následně

je pomocí `StreamReader` a `JsonTextReader` deserializován zvolený soubor. Metoda `Deserialize` vrací objekty, které je nutno ještě přetypovat. Podle `Type` se rozdělí načtené prvky. Následně se podle různých datových struktur získají hodnoty. Společně se poté hodnoty s klíčem a typem importují do správce. Pro `SortedSet` a `Hash` jsou data ukládána podobně, proto získávají data stejným způsobem. Je tomu také v případě `Listu` a `Setu`.

```
1  var json = serializer.Deserialize<JArray>(jr);
2  var result = new BindingList<IDbRow>();
3
4  foreach (var element in json)
5  {
6   switch (element["Type"].Value<int>())
7   {
8       case (int)RedisKeyType.SortedSet:
9       case (int)RedisKeyType.Hash:
10          var hashValues = element["Value"].Select(
11             item =>new KeyValuePair<string, string>(item["Key"].Value<string>(),
12             item["Value"].Value<string>())).ToList();
13
14          result.Add(new DatabaseRow<IEnumerable<KeyValuePair<string, string>>>
15          {
16              Type = (RedisKeyType)element["Type"].Value<int>(),
17              Key = element["Key"].Value<string>(),
18              ValueData = hashValues
19          });
20          break;
21      case (int)RedisKeyType.String:
22          result.Add(new DatabaseRow<string>
23          {
24              Type = RedisKeyType.String,
25              Key = element["Key"].Value<string>(),
26              ValueData = element["Value"].Value<string>()
27          });
28          break;
29      case (int)RedisKeyType.List:
30      case (int)RedisKeyType.Set:
31          ...
32          break;
33   }
34  }
```

Ukázka exportovaného souboru JSON s daty je uvedena níže. Typy dat jsou určeny číselně podle enumerátoru `RedisKeyType`. Například typ String je enumerátorem označen jako číslo jedna, List jako číslo dva atd.

```
[
    {"Type": 1, "Key": "Key1", "Value": "Hello World!"},
    {"Type": 2, "Key": "ShoppingList", "Value": [
        "milk",
        "butter",
        "bread",
        "eggs" ]
    },
    {"Type": 5, "Key": "User:1234", "Value": [
        {"Key": "Name", "Value": "Peter"},
        {"Key": "Surname", "Value": "Black"} ]
    }
]
```

7.6 Ukládání konfigurace připojení

V aplikaci je možné přidávat nastavení pro připojení k serverům s Redisem. Konfigurace je ukládána do souboru *config.xml*, který je vytvořen v `c:\Users\<USER>\AppData\Local\Temp\`. K ukládání dochází pokaždé, když dojde ke změně v seznamu s připojeními. Do souboru se ukládají základní informace, kterými jsou jméno serveru, jeho adresa, port a heslo pro případné připojení k zabezpečenému serveru. V aplikaci se ještě uchovává informace o roli, která určuje, zda je server v roli primárního nebo sekundárního uzlu. V případě, že se k serveru nepodařilo připojit, je označen jako „`notconnected`“. Informace o roli se získává vždy aktuálně při připojení k serveru, proto se tato informace neukládá do konfiguračního souboru a vlastnost `Role` je tudíž doplněna o atribut `[XmlIgnore]`. Aktuální seznam připojení je udržován v aplikaci a současně je i při jakékoliv změně uložen také na disk.

```
private void ConnectionListOnListChanged(object sender, ListChangedEventArgs
    listChangedEventArgs)
{
    SerializeToXml<BindingList<ServerNode>>(Path.GetTempPath() + ConfigName,
        ConnectionList);
}
```

Pro ukládání se opět využívá serializace, která je podrobněji popsána v předchozí kapitole. Tentokrát je ale pro formát XML. XML serializaci poskytuje přímo .NET Framework. Používá se k tomu třída `XmlSerializer` patřící do jmenného prostoru `Systém.Xml.Serialization`.

```
1  public static void SerializeToXml<T>(string path, object obj)
2  {
3      var xs = new XmlSerializer(typeof(T));
4      using (var fs = File.Create(path))
5      {
6          xs.Serialize(fs, obj);
7      }
8  }
```

Načítání aktuálního konfiguračního souboru je vždy voláno na začátku spuštění aplikace při inicializaci hlavního okna aplikace. Pokud konfigurační soubor neexistuje, vytvoří se nový seznam pro ukládání nastavení serverů. V opačném případě je použita deserializace souboru *config.xml*. Výsledek, který je vrácen metodou `Deserialize`, je převeden na aktuální seznam připojení.

```
1  private void LoadConfig()
2  {
3      if (!File.Exists(Path.GetTempPath() + ConfigName))
4      {
5          ConnectionList = new BindingList<ServerNode>();
6      }
7      else
8      {
9       ConnectionList = DeserializeFromXml<BindingList<ServerNode>>(Path.
            GetTempPath() + ConfigName);
10     }
11 }
12
13 public static T DeserializeFromXml<T>(string path)
14 {
15     var xs = new XmlSerializer(typeof(T));
16     using (var fs = File.OpenRead(path))
17     {
18         return (T)xs.Deserialize(fs);
19     }
20 }
```

Metody `SerializeToXml` a `DeserializeFromXml` jsou uvnitř třídy `SerializeHelper`, která se nachází ve složce `Helpers`. Níže je uveden příklad obsahu souboru *config.xml*.

```
1  <?xml version="1.0"?>
2  <ArrayOfServerNode xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:
       xsd="http://www.w3.org/2001/XMLSchema">
```

```
3  <ServerNode>
4   <Name>School</Name>
5   <Host>147.228.67.125</Host>
6   <Port>6379</Port>
7  </ServerNode>
8  <ServerNode>
9   ...
10 </ServerNode>
11 </ArrayOfServerNode>
```

8 Testování

Tato kapitola je věnována samotnému testování vytvořeného správce pro databáze. Cílem je ověřit správnou funkčnost programu v různých testovacích případech. Proces testování aplikace probíhal na lokálním počítači s operačním systémem Windows. Databázový systém Redis byl spouštěn na lokálním počítači pomocí verze pro Windows[1]. Dále byl Redis spuštěn na virtualizovaném operačním systému Debian pomocí nástroje VirtualBox a následně byla také databáze zprovozněna na třech strojích výpočetního clustru, které poskytla univerzita opět s operačním systémem Debian. Díky tomu mohly být vyzkoušeny vlastnosti master-slave replikace.

Konfigurace počítače

Testování aplikace proběhlo na počítači s následující konfigurací:

- **Procesor:** Intel Core2 Duo CPU
- **Pamět' RAM:** 3GB
- **Operační systém:** Windows 7 Professional
- **Typ systému:** 32bitový operační systém
- **Hardisk:** 500 GB

8.1 Testovací scénáře

Pro možnost otestování správné funkčnosti aplikace se mohou použít tzv. testovací scénáře. Jedná se o popis vstupních podmínek a hodnot pro vykonávání scénáře a očekávaný výsledek po provedení scénáře. Následující scénáře zachycují chování aplikace při zadání nové hodnoty do databáze, při pokusu smazání záznamu z databáze, která je v roli slave, a při smazání hodnoty v seznamu.

[1] `https://github.com/MSOpenTech/redis`

Testovací scénář	Přidání klíče do databáze
Popis	Uživatel chce zadat novou dvojici klíč hodnota do zvolené databáze. V aplikaci má již nastavené požadované připojení k aktivnímu serveru s databází.
Průběh	V hlavním okně aplikace je dvojklikem zobrazen seznam databází běžících na požadovaném serveru. Dalším dvojklikem se otevře záložka se seznamem všech klíčů v dané databázi. V horní části je zvoleno tlačítko *Add key* pro přidání nového klíče. V nabídce pro přidání je zvolen typ klíče String, název klíče a pomocí tlačítka, označeného symbolem tužky, je nastavena příslušná hodnota klíče. Pro přidání klíče do databáze je stisknuto tlačítko *Add* v hlavní nabídce. Klíč je úspěšně přidán do databáze a tabulka výpisem obsahu databáze je automaticky aktualizována.
Výsledek	Výše zmíněný popis přidání nového klíče do databáze funguje korektně. Klíč i s požadovanou hodnotou se uloží do databáze a následně se zobrazí ve výpisu. V případě, že při nastavování klíče pro přidání se stane server nedostupný, tudíž do něj není možno zapisovat, je uživatel upozorněn, že připojení nemohlo být vytvořeno. Tento testovací scénář potvrdil správnou funkčnost přidávání klíče do databáze.

Tabulka 8.1: Testování přidání klíče do databáze.

Testovací scénář	Smazání klíčů z databáze v roli slave
Popis	V případě, že je databáze v roli slave, která umožňuje pouze čtení a replikuje si data z uzlu master, není možno smazat žádný klíč (platí to i pro přidání klíče).
Průběh	Jsou aktivní dva servery, jeden v roli master a druhý v roli slave. Po dvojkliku na požadované připojení je uzel typu slave označen šedivou ikonkou. Následně je uživatelem otevřena záložka s obsahem příslušné databáze. Uživatel si pomocí tlačítka *Ctrl* může zvolit více klíčů, které chce najednou smazat. Poté stiskne tlačítko *Delete key* pro smazání zvolených klíčů. Uživatel je dotázán, zda opravdu chce vybrané klíče smazat. Při kladné odpovědi je ale uživatel následně upozorněn, že databáze je readonly slave a nemůže se do ní zapisovat, tudíž ani odstraňovat hodnoty.
Výsledek	Požadovaná operace smazání hodnot z databáze, která je v roli slave, proběhla přesně podle očekávání. Uživatel byl korektně upozorněn, že požadovaná akce nemohla být provedena a data zůstala nezměněna.

Tabulka 8.2: Testování smazání klíče.

Testovací scénář	Smazání jedné hodnoty v klíči typu List
Popis	Uživatel požaduje smazání jedné hodnoty v klíči, který je typu List. V aplikaci je otevřena záložka s daty požadované databáze.
Průběh	Na záložce jsou zobrazena data. Po dvojkliku na klíč, který je označen jako typ List, se otevře náhledové okno, stejné, které by se otevřelo po zvolení řádku s daným klíčem a stisknutí tlačítka *Edit key*. Po zvolení tlačítka se symbolem tužky se otevře editační okno, ve kterém jsou uvedeny jednotlivé hodnoty patřící zvolenému klíči. Hodnota, která má být odstraněna, se označí a pomocí klávesy *Delete* se zavolá dotaz do databáze a požadovaná hodnota se odstraní. Tabulka s daty je automaticky po provedeném dotazu do databáze aktualizována.
Výsledek	Smazání hodnoty v klíči, který je typu List, proběhlo správně. Obdobně zmíněný postup funguje i pro další typy dat. V případě, že během odstraňování došlo k výpadku serveru, je uživatel opět upozorněn příslušným chybovým hlášením.

Tabulka 8.3: Testování smazání hodnoty v seznamu.

9 Závěr

Úkolem diplomové práce bylo navrhnout a realizovat správce pro NoSQL databázi. Tomu předcházelo seznámení se s možnostmi a současnými nerelačními databázovými systémy. Při řešení a návrhu se vycházelo z porovnání různých typů NoSQL databází a klasických relačních databázových systémů.

Z nabízených možností byl zvolen databázový systém Redis, který je zástupcem key/value NoSQL databází. Druhým zvažovaným řešením bylo použití systému Riak. Z důvodu lepší podpory a větší rozšířenosti byl vybrán výše uvedený distribuovaný systém Redis, který umožňuje master-slave replikaci. API poskytované Redisem bylo důkladně prostudováno a získané poznatky byly využity při tvorbě výsledného správce – NoSqlManager.

Pro realizaci byl použit programovací jazyk C# s knihovnou ServiceStack.Redis, která poskytuje vhodné prostředky pro komunikaci se systémem Redis. Pomocí těchto nástrojů byl vytvořen správce NoSqlManager, který umožňuje uživatelsky jednoduchou správu databáze. Podporuje základní funkčnost, kterou je přidání nového klíče, odstranění vybraných klíčů a jejich hodnot, editace stávajících dat a také import a export do souboru ve formátu JSON.

Navíc byla v průběhu vývoje přidána do programu konzole, která zpřístupňuje veškeré instrukce příkazové řádky Redis-cli. Tím byla dosažena kompletní funkcionalita, kterou systém Redis nabízí. Díky tomu může NoSqlManager konkurovat existujícím řešením a zařadit se tak mezi správce, kteří poskytují uživatelsky příjemné a přitom účinné prostředí.

Ověření výsledků práce bylo provedeno za pomoci několika testovacích scénářů, jež byly vybrány tak, aby pokrývaly co nejširší množinu situací, ke kterým by mohlo v reálném případě používání dojít. NoSqlManager je vytvořen tak, aby poskytoval uživateli základní i pokročilé funkce Redisu. Vzhledem k tomu, že se v současné době NoSQL databáze stále rychle vyvíjejí, bylo by případně možné v budoucnu NoSqlManager dále rozšiřovat a použít i pro další verze NoSQL databázových systémů.

Seznam zkratek

ACID	Atomicity, Consistency, Isolation, Durability
AOF	Append Only File
API	Aplication Programming Interface
BASE	Basically Available, Soft-state, Eventually Consistent
BLOB	Binary Large Object
BSON	Binary JSON
CAP	Consistency, Availability, Partition tolerance
CQL	Cassandra Query Language
DCL	Data Control Language
DDL	Data Definition Language
DML	Data Manipulation Language
GQL	Google Query Language
GUI	Graphical User Interface
HQL	Hypertable Query Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
MVVM	Model View ViewModel
NoSQL	Not Only Structured Query Language
RAM	Random Access Memory

RDB Random Database File

RDBMS Relational Database Management System

REST Representation State Transfer

SQL Structured Query Language

SŘBD Systém řízení báze dat

UnQL Unstructured Data Query Language

TCP/IP Transmission Control Protocol/Internet Protocol

TCL Transaction Control Language

WPF Windows Presentation Foundation

XAML Extensible Application Markup Language

XML Extensible Markup Language

Literatura

[Aug11] AUGUSTÝN, Michal. Zamyšlení nad NoSQL. *Augiho web* [online]. 29. 10. 2011 [cit. 2014-04-28]. Dostupné z: http://www.augi.cz/programovani/zamysleni-nad-nosql/

[Bla10] BLACK, Benjamin. Introduction to Cassandra: Replication and Consistency. *SlideShare* [online]. 29. 4. 2010 [cit. 2014-04-25]. Dostupné z: http://www.slideshare.net/benjaminblack/introduction-to-cassandra-replication-and-consistency

[Cat11] CATTELL, Rick. Scalable SQL and NoSQL data stores. *ACM SIGMOD Record*. 2011-05-06, vol. 39, issue 4, s. 12- [cit. 2014-04-26]. DOI: 10.1145/1978915.1978919. Dostupné z: http://portal.acm.org/citation.cfm?doid=1978915.1978919

[Con09] CONOLLY, Thomas, Carolyn E BEGG a Richard HOLOWCZAK. *Mistrovství – databáze: profesionální průvodce tvorbou efektivních databází*. Vyd. 1. Brno: Computer Press, 2009, 584 s. ISBN 978-80-251-2328-7.

[Dat14] Cassandra: Getting Started Documentation. *DataStax* [online]. 2014 [cit. 2014-04-25]. Dostupné z: http://www.datastax.com/documentation/getting_started/doc/getting_started/gettingStartedCassandraIntro.html

[Dbe14] Redis vs. Riak Comparison. *DB-Engines: Knowledge Base of Relational and NoSQL Database Management Systems* [online]. 2014 [cit. 2014-05-01]. Dostupné z: http://db-engines.com/en/system/Redis%3BRiak

[Dea04] DEAN, Jeffrey a Sanjay GHEMAWAT. GOOGLE, Inc. *MapReduce: Simplied Data Processing on Large Clusters*. 3. 10. 2004 [cit. 2014-04-22]. Dostupné z: http://static.googleusercontent.com/media/research.google.com/cs//archive/mapreduce-osdi04.pdf

[Dol11] DOLÁK, Ondřej. Big data: Nové způsoby zpracování a analýzy velkých objemů dat. In: *SystemOnLine* [online]. 2011 [cit. 2014-04-16]. Dostupné z: http://www.systemonline.cz/clanky/big-data.htm

[Flo12] FOWLER, Martin. NosqlDefinition. *Martin Fowler* [online]. 9. 1. 2012 [cit. 2014-04-22]. Dostupné z: http://martinfowler.com/bliki/NosqlDefinition.html

[Gro09] GRONINGEN, Martijn. Introduction to Hadoop. *Trifork Blog* [online]. 4. 8.2009 [cit. 2014-04-22]. Dostupné z: http://blog.trifork.com/2009/08/04/introduction-to-hadoop/

[Her06] HERNANDEZ, Michael J. *Návrh databází*. 1. vyd. Praha: Grada, 2006, 408 s. ISBN 80-247-0900-7.

[Hof09] HOFF, Todd. Neo4j: a Graph Database that Kicks Buttox. *High Scalability* [online]. 13. 6. 2009 [cit. 2014-04-25]. Dostupné z: http://highscalability.com/neo4j-graph-database-kicks-buttox

[Hof10] HOFF, Todd. Troubles with Sharding: What can we learn from the Foursquare Incident?. *High Scalability* [online]. 15. 10. 2010 [cit. 2014-04-24]. Dostupné z: http://highscalability.com/blog/2010/10/15/troubles-with-sharding-what-can-we-learn-from-the-foursquare.html

[Cha11] CHAPPLE, Mike. Abandoning ACID in Favor of BASE. *About.com* [online]. 2011 [cit. 2014-04-22]. Dostupné z: http://databases.about.com/od/otherdatabases/a/Abandoning-Acid-In-Favor-Of-Base.htm

[Jak07] JAKUBČÍK, Ondřej. Srovnání databázových serverů. *Linux EXPRES* [online]. 2007 [cit. 2014-04-22]. Dostupné z: http://www.linuxexpres.cz/software/srovnani-databazovych-serveru

[Kal12] KALUŽA, Jindřich a Ludmila KALUŽOVÁ. *Modelování dat v informačních systémech*. 1. vyd. Praha: Ekopress, 2012, 125 s. ISBN 978-80-86929-81-1.

[Kat12] KATSOV, Ilya. NoSQL Data Modeling Techniques. *Highly Scalable Blog* [online]. 1. 3. 2012 [cit. 2014-04-22]. Dostupné z: http://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/

[Klo12] KLOBÁSA, Pavel. Letmý technologický pohled na MongoDB. *VsadNaJavu.cz: Odborný weblog o vývoji webových aplikací nad platformou Java a J2EE* [online]. 23. 1. 2012 [cit. 2014-04-24]. Dostupné z: `http://vsadnajavu.cz/2012-01/databaze/letmy-technologicky-pohled-na-mongodb/`

[Kos99] KOSEK, Jiří. Aplikace na Webu: Jak pracují databáze na Webu: Co je to databáze. *Domovská stránka Jirky Koska: VŠE O WWW* [online]. 1999 [cit. 2014-04-22]. Dostupné z: `http://www.kosek.cz/clanky/iweb/12.html`

[Kul08] KULHAN, Jakub a Zdeněk LEHOCKÝ. Normalizace relačních databází. *Programujte.com* [online]. 2008 [cit. 2014-04-22]. Dostupné z: `http://programujte.com/clanek/2008071900-normalizace-relacnich-databazi/`

[Lac12] LACHLAN, James. Why Big Data and Business Intelligence Are Like One Direction. SmartData Collective [online]. 2. 10. 2012 [cit. 2014-04-25]. Dostupné z: `http://smartdatacollective.com/yellowfin/75616/why-big-data-and-business-intelligence-one-direction`

[Mam11] MAMTANI, Vinod. Cassandra Data Model. *NimbleDais* [online]. 1. 6. 2011 [cit. 2014-04-25]. Dostupné z: `http://nimbledais.com/?p=150`

[MaS08] Market Share. *MySQL* [online]. 2008 [cit. 2014-04-22]. Dostupné z: `http://www.mysql.com/why-mysql/marketshare/`

[Mon14] The MongoDB 2.6 Manual. *MongoDB* [online]. 2014 [cit. 2014-04-24]. Dostupné z: `http://docs.mongodb.org/manual/`

[MSSQL01] *Microsoft SQL Server 2000: administrace systému MCSE Training Kit.* Vyd. 1. Praha: Computer Press, 2001, xxii, 651 s. ISBN 80-722-6526-1.

[MyS11] Co je to databáze MySQL?. *Artic Studio Webdesign* [online]. 2011 [cit. 2014-04-22]. Dostupné z: `http://www.artic-studio.net/slovnicek-pojmu/databaze-mysql/`

[Nia13] NIAH, Omer. Four category of NoSQL databases with examples. *Database Diary* [online]. 12. 12. 2013 [cit. 2014-04-23]. Dostupné z: `http://dbdiary.com/four-category-of-nosql-databases-with-examples/`

[Norm07] Teorie relačních databází: Normalizace. *Manualy.net* [online]. 2007 [cit. 2014-04-22]. Dostupné z: `http://www.manualy.net/article.php?articleID=13`

[Opp06] OPPEL, Andrew. *Databáze bez předchozích znalostí.* Vyd. 1. Brno: Computer Press, 2006, 319 s. ISBN 80-251-1199-7.

[Pok12] POKORNÝ, Jaroslav. NoSQL databáze: současný stav vývoje. In: *Moderní databáze 2012: Zpracování velkých objemů dat a transakcí.* Praha: KOMIX s.r.o., 2012, s. 11. ISBN 978-80-905231-0-4. Dostupné z: `http://www.analyzyareporting.cz/sitecore/content/Komix_cs-CZ/Home/Produkty/MD/2012/~/media/Komix/Downloads/Konference_Moderni_databaze/Moderni_databaze_2012/Sbornik_konference_Moderni_databaze_2012.ashx`

[Pok97] POKORNÝ, Jaroslav. Základy implementace souborů a databází. 1. vyd. Praha: Karolinum, 1997, 196 s. ISBN 80-718-4472-1.

[Rah10] RAHIEN, Ayende. That No SQL Thing: Column (Family) Databases. *Ayende @ Rahien* [online]. 14. 5. 2010 [cit. 2014-04-25]. Dostupné z: `http://ayende.com/blog/4500/that-no-sql-thing-column-family-databases`

[Red12] REDMOND, Eric, Jim R WILSON a Jacquelyn CARTER. *Seven databases in seven weeks: a guide to modern databases and the NoSQL movement* [online]. Dallas, Tex.: Pragmatic Bookshelf, 2012, xiii, 333 p. [cit. 2014-04-26]. Pragmatic programmers. ISBN 19-343-5692-1.

[Red14] Documentation: Redis. *Redis* [online]. 2014 [cit. 2014-04-27]. Dostupné z: `http://redis.io/documentation`

[Ria11] Why Riak. *Riak Docs* [online]. 2011 [cit. 2014-04-26]. Dostupné z: `http://docs.basho.com/riak/latest/theory/why-riak/`

[Sad12] SADALAGE, Pramod J. a Martin FOWLER. Introduction to Polyglot Persistence: Using Different Data Storage Technologies for Varying Data Storage Needs: Polyglot Data Store Usage. *InformIT* [online]. 5. 9. 2012 [cit. 2014-04-28]. Dostupné z: `http://www.informit.com/articles/article.aspx?p=1930511&seqNum=2`

[Sad13] SADALAGE, Pramod J a Martin FOWLER. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence.* Upper Saddle River, NJ: Addison-Wesley, c2013, xix, 164 p. ISBN 03-218-2662-0.

[Seg12] SEGUIN, Karl. *The Little Redis Book.* 23. 1. 2012. Dostupné z: `http://openmymind.net/2012/1/23/The-Little-Redis-Book/`

[Sch88] SCHEBER, Anton. *Databázové systémy.* 1. vyd. Praha: SNTL, 1988, 321 s.

[Skř00] SKŘIVAN, Jaromír. Databáze a jazyk SQL. *Interval.cz* [online]. 2000 [cit. 2014-04-22]. Dostupné z: `http://interval.cz/clanky/databaze-a-jazyk-sql/`

[Skř02] SKŘIVAN, Jaromír. Databáze nejsou jen MySQL. *Interval.cz* [online]. 2002 [cit. 2014-04-22]. Dostupné z: `http://interval.cz/clanky/databaze-nejsou-jen-mysql/`

[Sto10] STONEBRAKER, Michael. SQL databases v. NoSQL databases. *Communications of the ACM* [online]. 2010-04-01, vol. 53, issue 4, s. 10- [cit. 2014-04-28]. DOI: 10.1145/1721654.1721659. Dostupné z: `http://portal.acm.org/citation.cfm?doid=1721654.1721659`

[Str11] STRAUCH, Christof. *NoSQL Databases*. Stuttgart, 2011. Dostupné z: `http://www.christof-strauch.de/nosqldbs.pdf`

[Šel11] ŠELENG, Martin a Michal LACLAVÍK. *Dostupné škálovatelné riešenia pre spracovanie velkého objemu dát a dátové sklady*. Mikulov: DATAKON 2011, 2011, 22 s. Dostupné z: `http://laclavik.sk/publications/datakon_final_2011.pdf`

[Tiw11] TIWARI, Shashank C. *Professional nosql*. 1. vydání. Indianapolis: Wiley Pubishing, Inc., 2011. ISBN 04-709-4224-X.

[Ver13] VERMA, Ananda. CAP Theorem. *TechSpritz: Stay Tuned to Technology* [online]. 17. 5. 2013 [cit. 2014-04-22]. Dostupné z: `http://www.techspritz.com/cap-theorem/`

A Uživatelská příručka

Tato uživatelské příručka popisuje použití aplikace NoSqlManager pro databázový systém Redis, která slouží pro jednoduchou správu. Je uveden i postup pro zprovoznění serveru s Redisem.

Zprovoznění databázového systému Redis

Pro spuštění databázového systému jsou uvedeny tři možnosti, kde první je určena pro spuštění na operačním systému Windows, druhá pomocí VirtualBoxu a třetí pro operační systém Linux.

Na oficiálních stránkách Redisu je odkaz na projekt skupiny Microsoft Open Tech group[1], která vyvíjí a udržuje verzi Redisu pro operační systém Windows. Poskytuje možnost si stáhnout zip soubor s funkční verzí, který je umístěn na stránkách projektu v `/bin/release/`. Tento soubor stačí rozbalit a následně spustit redis-server.exe. Systém pak následně běží jako localhost na portu 6379. Je také možno si z poskytnutých zdrojových souborů vytvořit vlastní build.

Druhou možností je otevřít si pomocí programu VirtualBox virtualizovaný obraz operačního systému Debian, který je součástí přiloženého DVD. Ve virtualizovaném systému jsou nastaveni dva uživatelé, user s heslem user a root, který má administrátorská práva a heslo je opět stejné, root. Po připojení stačí spustit server příkazem `redis-server`. Dále jsou uvedeny podrobnosti o virtualizovaném obrazu.

- **Uživatel:** user
- **Heslo:** user
- **Administrátor:** root
- **Heslo administrátora:** root
- **IP adresa:** 192.168.56.101
- **Hostname:** debian1
- **Operační systém:** Debian 3.2.51-1 i686 GNU/Linux

[1] `https://github.com/MSOpenTech/redis`

Poslední možností je nainstalovat si systém Redis na operační systém Linux. Postup instalace je převzat z domovských stránek Redisu. Redis se nejprve stáhne, pak rozbalí a zkompiluje pomocí následujících příkazů.

```
$ wget http://download.redis.io/releases/redis-2.8.2.tar.gz
$ tar xzf redis-2.8.2.tar.gz
$ cd redis-2.8.2
$ make
```

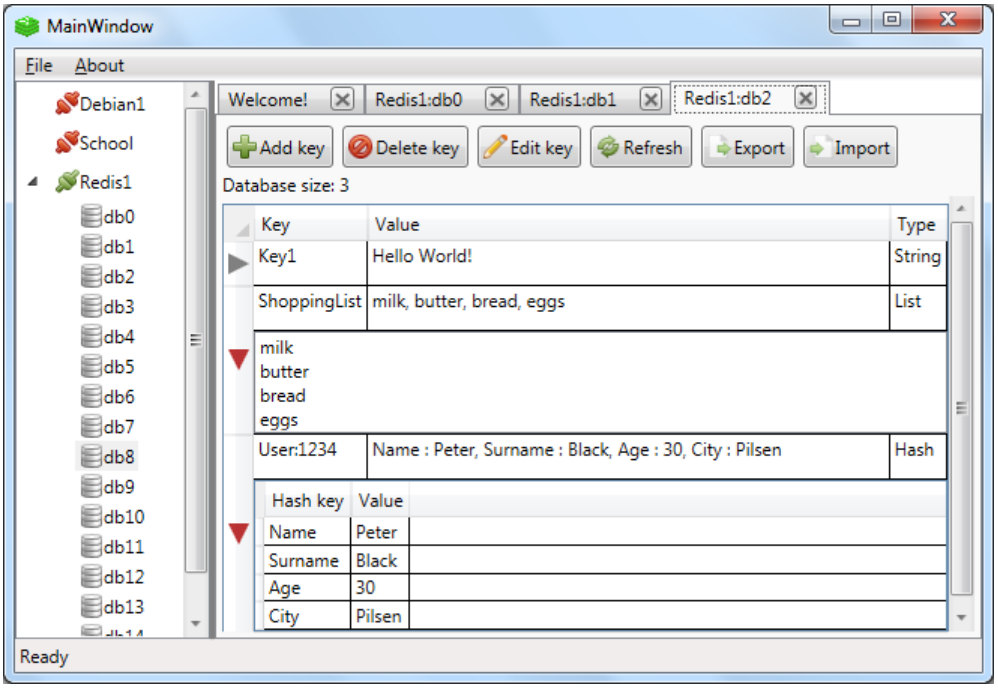
Je nutné doinstalovat také tcl verze 8.5 a provést test. Poté lze Redis spustit ve složce src opět pomocí příkazu `redis-server`.

```
$ apt-get install tcl
$ make test
$ make install
```

Vytvoření distribuované databáze s master-slave replikací probíhá přes příkazovou řádku Redis-cli. Na všech strojích musí běžet vlastní instance Redisu, poté lze na uzlu, který bude v roli slave, zadat příkaz `slaveof host port`. Pomocí příkazu `slaveof no one` se zruší replikace na daném uzlu.

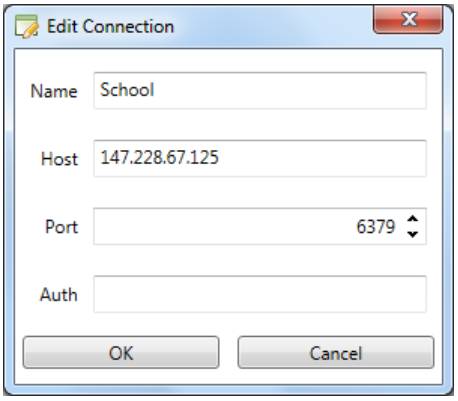
```
>slaveof 192.168.56.101 6379
```

NoSqlManager



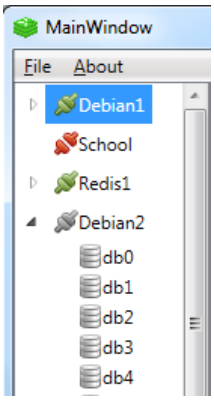
Obrázek A.1: Ukázka hlavního okna programu NoSqlManager.

Po spuštění programu NoSqlManager.exe se zobrazí uživateli hlavní okno aplikace, viz obrázek A.1. To je rozděleno do několika částí. Levou část zabírá okno pro výpis stromové struktury pro připojení, pravou část pak zabírají záložky s daty. V horní části je lišta obsahující dvě nabídky *File* a *About*. Dolní část zabírá stavový řádek, ve kterém jsou zobrazovány různé stavové informace. Při prvním spuštění je nutno přidat nové připojení k serveru s běžícím Redisem. Pomocí nabídky *File* → *Add New Connection* se otevře okno pro zadání jména serveru, kterým se bude prezentovat v aplikaci, IP adresa označena v okně jako *Host*, port, na kterém běží instance Redisu a v případě, že je Redis zabezpečen, se může zadat i heslo. Okno pro přidání nového připojení je stejné jako pro editaci stávajícího připojení, viz obrázek A.2. Nastavená připojení se ukládají do konfiguračního souboru a při spuštění jsou opětovně načtena do stromové struktury.



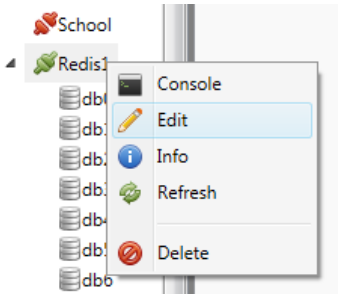
Obrázek A.2: Okno pro přidání a editaci připojení.

V levé části aplikace je názorně zobrazen seznam zadaných připojení. Na začátku jsou označeny červeným symbolem, který znamená, že se k serveru ještě nepřipojovalo nebo že je případně server nedostupný. Po dvojkliku na název serveru se stav aktualizuje. V případě úspěšného připojení se změní ikonka na zelenou a otevře se seznam databází. Pokud je server v roli slave, je opět při dvojkliku otevřen seznam databází, ale ikonka je zobrazena šedivou barvou. Tím je uživatel upozorněn, že se jedná pouze o sekundární uzel, do kterého se nezapisuje. Na obrázku A.3 je znázorněn seznam připojení. Pokud na daném serveru Redis neběží nebo je požadovaná adresa neplatná, je uživatel upozorněn chybovým hlášením, že se nepodařilo připojit k databázi.



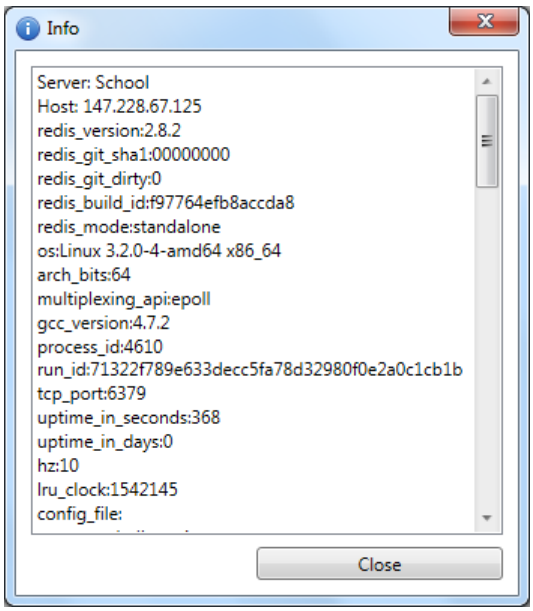
Obrázek A.3: Ukázka stromové struktury seznamu pro připojení.

Nad názvem serveru je možné zavolat další nabídku zobrazenou na obrázku A.4 pomocí stisknutí pravého tlačítka myši. Je zde možné zvolit editování nastavení připojení, možnost aktualizovat stav připojení pomocí *Refresh* a také pomocí nabídky



Obrázek A.4: Ukázka nabídky nad zvoleným serverem.

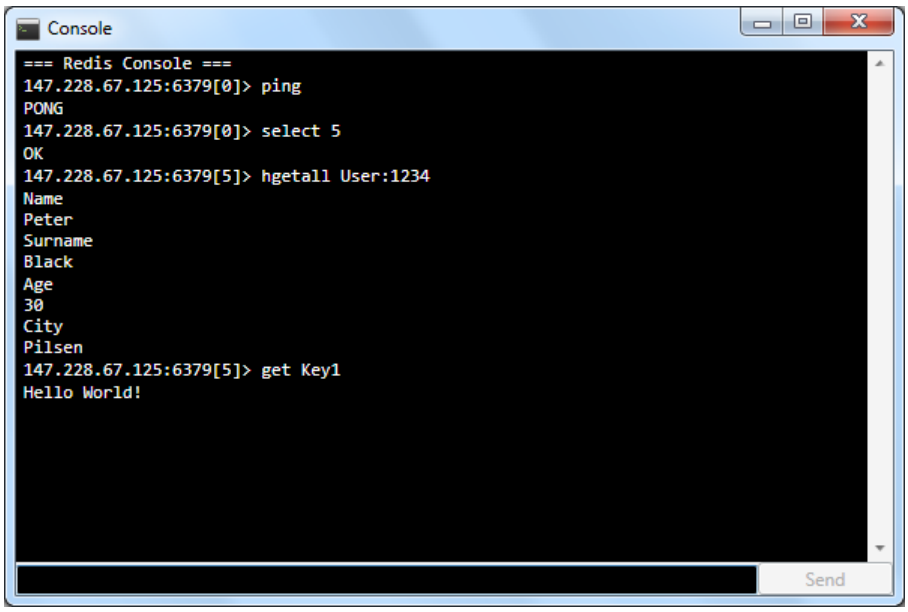
Delete zvolené nastavení smazat ze stromové struktury. Další poskytovanou nabídkou je možnost zobrazení informacích o Redisu na zvoleném serveru. Informace jsou zobrazeny formou výpisu, který vrací po dotazu Redis, v samostatném okně, jehož ukázka je na obrázku A.5. Je zde například uvedena verze systému Redis, počet připojených klientů a množství klíčů v jednotlivých databázích.



Obrázek A.5: Ukázka okna s informacemi o serveru.

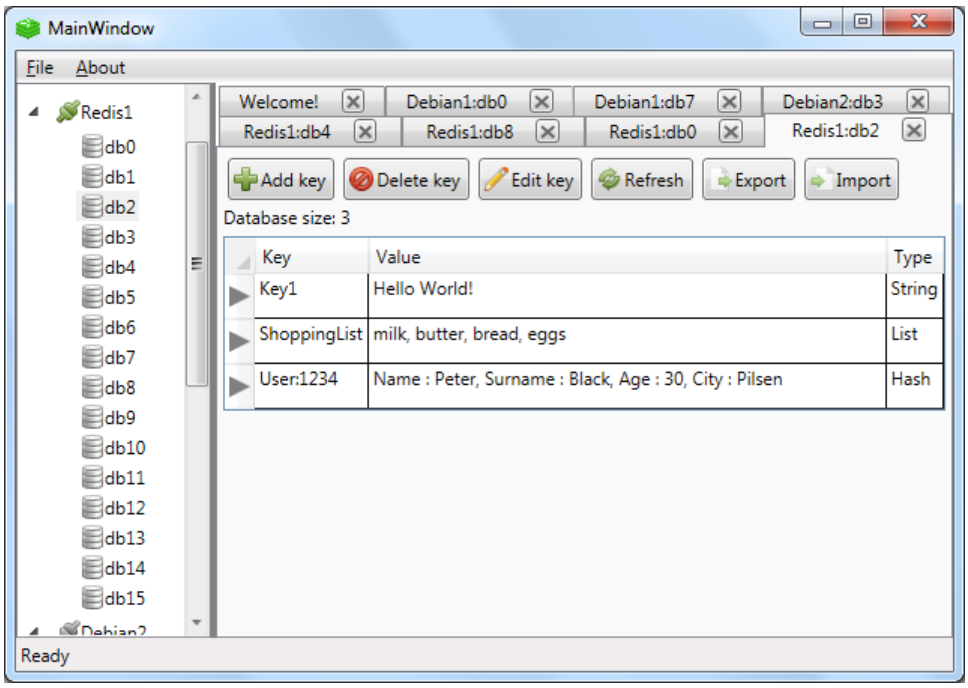
V nabídce je také uvedena možnost otevření konzolového okna pro zvolený server. Konzole je zobrazena jako jednoduché okno s černým pozadím a bílým textem, viz obrázek A.6. Okno konzole je rozděleno na dvě části. První část zobrazuje požadované dotazy a jejich výsledky. Pro přehlednost je vždy před dotazem zobrazena adresa serveru a číslo aktuální databáze. Výpis odpovídá příkazové řádce Redis-cli.

Příkazy se zadávají do pole, které je umístěno v dolní části. Pro jednodušší odesílání dotazů je zde umístěno tlačítko *Send*, které je aktivní pouze, pokud je v poli pro dotazy zapsán nějaký text. Po zapsání dotazu do vstupního pole je možné také odeslat dotaz pomocí klávesy *Enter*. Okno se zavře buď příkazem exit, nebo klasicky křížkem v pravém horním rohu. Konzolových oken může být otevřeno více a je možné přepínat mezi hlavním oknem aplikace a konzolí, díky čemuž může uživatel používat jak grafické rozhraní, tak příkazovou řádku.



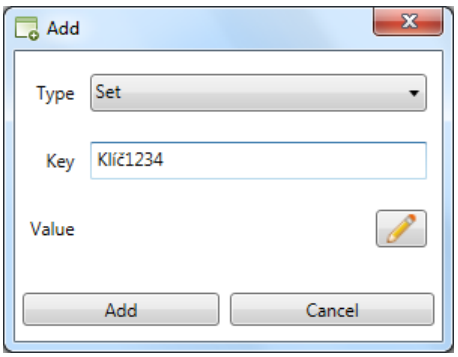
Obrázek A.6: Konzolové okno.

Po dvojkliku na zvolenou databázi serveru ve stromové struktuře se otevře v hlavní části okna nová záložka s daty. Pro jednodušší orientaci je záložka vždy označena jménem serveru a označením databáze. Každá zvolená databáze má otevřenu pouze jednu záložku. V případě, že záložka je otevřena, ale není aktivní, a uživatel chce znovu zobrazit daná data, stačí se přepnout na záložku nebo znova otevřít dvojklikem danou databázi, záložka se zvolenou databází se poté stane aktivní. Záložky je také možné pomocí tlačítka v záhlaví zavřít. Na obrázku A.7 je znázorněna ukázka hlavního okna, ve kterém je otevřeno několik záložek.



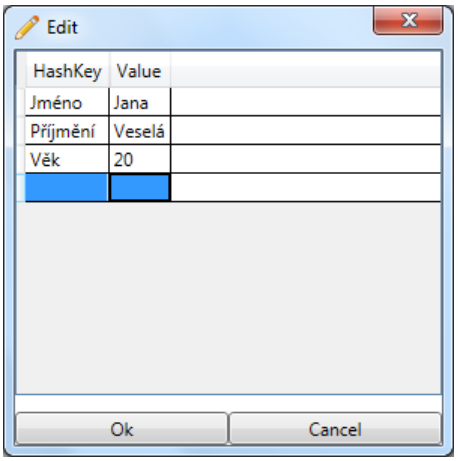
Obrázek A.7: Ukázka s několika záložkami.

Obsah každé záložky tvoří šest tlačítek a tabulka s daty. Tabulka je tvořena třemi sloupci, *Key*, *Value* a *Type*. *Key* zobrazuje jméno klíče. *Value* zobrazuje hodnotu u dat typu *String*, u zbývajících typů je zde uveden zjednodušený výpis hodnot. *Type* pak určuje typ dat. Pomocí postranních šipek tabulky je možné si rozbalit podrobnější zobrazení hodnot. U typů *List* s *Set* je pak zobrazen seznam hodnot. U *Hash* a *SortedSet* je po rozbalení uvedena menší tabulka. V případě *Hash* je tvořena dvěma sloupci *Hash* key a *Value*. *SortedSet* má také dva sloupce, *Value* a *Score*.



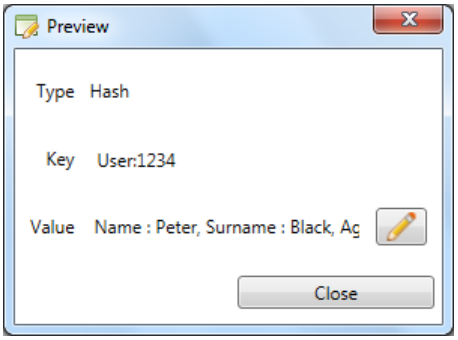
Obrázek A.8: Ukázka okna pro přidání nového klíče do databáze.

Nová hodnota do databáze se přidá pomocí tlačítka *Add key*. Zobrazí se okno s možností výběru typu klíče, viz obrázek A.8. Dále je nutno zadat jméno klíče. Hodnota se přidává pomocí tlačítka označeného symbolem tužka. Po kliknutí na toto tlačítko se otevře další okno, kde je umožněno podle zvoleného typu zadávat hodnoty. V případě zvolení typu *String*, je možno zadat pouze jednu hodnotu. V ostatních případech je umožněno po vyplnění jedné řádky v tabulce stisknout *Enter* a zadat další hodnoty. Okno pro přidávání hodnot je zobrazeno na obrázku A.9. Tlačítkem *Ok* se potvrdí zvolené hodnoty a uživatel je vrácen do okna pro nastavování jména a typu klíče. Může opětovně zavolat okno pro úpravu hodnot. V případě, že uživatel již vyplnil nějaké hodnoty a změní typ dat, je upozorněn, že může o nastavené hodnoty přijít. Po stisknutí tlačítka *Add* dojde k připojení k databázi a přidání hodnoty. Pokud je zvoleno tlačítko *Cancel* nedojde k žádné akci.



Obrázek A.9: Ukázka okna pro přidávání a úpravu hodnot klíče typu Hash.

Tlačítko *Delete key* vymaže z databáze vybrané klíče. Klíč se musí nejprve označit kliknutím na jeho řádek. V případě výběru více klíčů pro smazání stačí přidržet klávesu *Ctrl* a pomocí myši udělat hromadný výběr. Po vybrání klíčů stačí kliknout na tlačítko *Delete key* a požadované klíče jsou z databáze smazány.

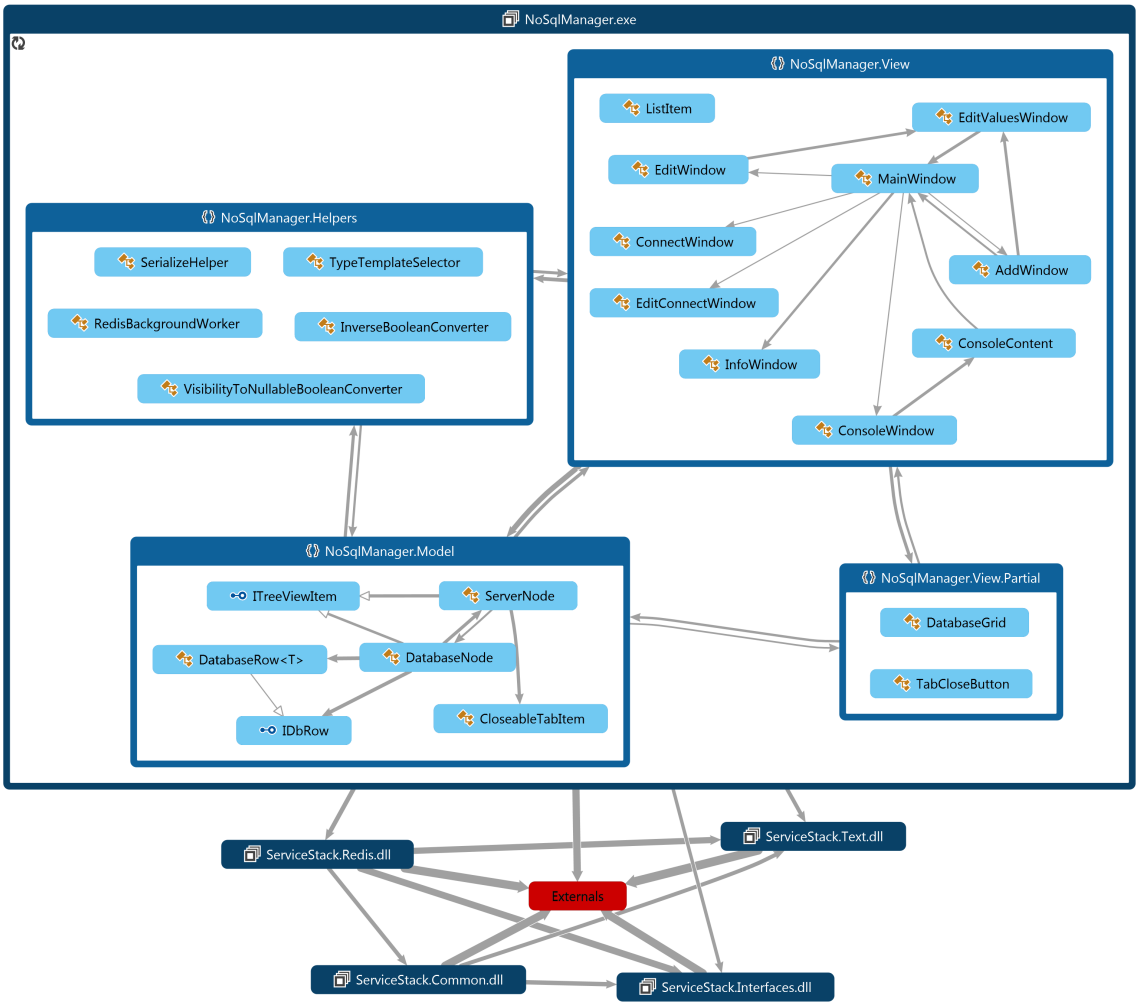


Obrázek A.10: Ukázka okna pro zobrazení náhledu klíče.

Pro úpravu dat je možné užít tlačítko *Edit key* nebo dvojklik na požadovaný řádek v tabulce. Zobrazí se náhledové okno, které je zobrazeno na obrázku A.10, kde jsou uvedeny podrobnější informace, jako je typ klíče, jméno klíče a náhled na hodnoty. Pro úpravu je nutno stisknout tlačítko se symbolem tužky. Otevře se stejná tabulka, která je při přidávání dat do databáze, viz A.9. Hlavním rozdílem oproti přidání nové hodnoty je, že již v této tabulce dochází k dotazům do databáze. Například v případě, že pomocí klávesy *Delete* smažeme nějakou hodnotu, dojde rovnou k provedení odstranění hodnoty v databázi. Při úpravě je možné pouze přidávat nové hodnoty nebo mazat stávající.

Tlačítko *Refresh* umožňuje aktualizovat tabulku s daty. Pomocí tlačítka *Export* je umožněno uložit aktuální data uvedená na zvolené záložce do souboru ve formátu JSON. Po kliknutí na tlačítko je zobrazeno klasické dialogové okno pro uložení souboru. Obdobně je tomu i při kliknutí na tlačítko *Import*, které poskytuje možnost nahrát data do databáze ze zvoleného JSON souboru. V případě, že se již v databázi nacházejí nějaká data, jsou při importu doplněna o data uložená v souboru. Pokud dojde k duplicitě klíčů, je stávající klíč přepsán importovaným klíčem a jeho hodnotami. Pouze v případě klíče typu *List* dojde k dvojitému přidání hodnot. Jedná se o očekávané chování, které je způsobeno jednoduchým přístupem Redisu k datům.

B Graf závislostí



Obrázek B.1: Graf závislostí jednotlivých jmenných prostorů vygenerovaný Visual Studiem.

C Obsah DVD

Přílohou této diplomové práce je DVD, které má následující strukturu:

- **Čti mě.txt** – Textový soubor, obsahující popis jednotlivých adresářů uložených na médiu.
- **Dokumentace** – Obsahuje elektronickou verzi této diplomové práce.
- **Program NoSqlManager** – Obsahuje všechny součásti programu, jako jsou spustitelný soubor a zdrojové kódy.
  - **bin** – Obsahuje zazipovanou spustitelnou verzi programu s přiloženými knihovnami.
  - **doc** – Vygenerovaná programátorská dokumentace.
  - **src** – Obsahuje zdrojové kódy programu a graf závislosti jmenných prostorů.
- **Programové vybavení** – Obsahuje nástroje pro spuštění databázového systému Redis.
  - **Instalátor .NET Framework 4.5.1** – Obsahuje instalační program Microsoft .NET Framework 4.5.1.
  - **MSOpenTech.Redis** – Obsahuje verzi databázového systému Redis, která je spustitelná na operačním systému Windows.
  - **VirtualBox** – Obsahuje virtualizovaný obraz operačního systému Debian s nainstalovaným systémem Redis