Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Diplomová práce

Možnosti použití NoSQL databáze

Bc. Tomáš Panyko, DiS.

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Tomáš Panyko, DiS.

Informatika

Název práce

Možnosti použití NoSQL databáze

Název anglicky

Utilization of NoSQL database

Cíle práce

Diplomová práce je tématicky zaměřená na NoSQL databázi MongoDB a jeho využití.
Cílem práce je:
- Seznámit čtenáře s NoSQL databázemi
- Nastínit přechod z MySQL na MongoDB
- Ukázat, jaké výhody a nevýhody plynou z přechodu z relačních databází na nerelační
- Otestovat výkonnost MongoDB databáze

Metodika

Metodika řešené problematiky diplomové práce je založena na analýze odborných informačních zdrojů. V teoretické části je osvětlena problematika NoSQL databází a jejich vlastností. Praktická část se zajímá o možný přechod z MySQL databáze na MongoDB a z toho plynoucí výhody a nevýhody přechodu z relační databáze na nerelační. Součástí práce je i otestování výkonnosti MongoDB databáze.

Doporučený rozsah práce

60 stránek

Klíčová slova

NoSQL, MongoDB, MySQL, Redis, HBase

Doporučené zdroje informací

1) MongoDB Applied Design Patterns. Sebastopol, California: O'Reilly Media, 2013. ISBN 978-1-4493-4004-9
2) Proffesional NoSQL.Indianapolis, Indiana: John Wiley & Sons, Inc., 2011. ISBN 978-0-470-94224-6
3) MongoDB: The Definitive Guide, 2nd Edition. Sebastopol, California: O'Reilly Media, 2013. ISBN 978-1-4493-4468-9
4) 50 Tips and Tricks for MongoDB Developers. Sebastopol, California: O'Reilly Media, 2011. ISBN 978-1-4493-0461-4
5) Learning the MySQL Database. Sebastopol, California: O'Reilly Media, 2014. ISBN 978-1-4493-6290-4
6) MongoDB [online]. 2013. Dostupné z: https://www.mongodb.org/
7) NOSQL Databases [online]. 2013. Dostupné z: http://nosql-database.org/

Předběžný termín obhajoby

2015/16 LS – PEF

Vedoucí práce

Ing. Alexandr Vasilenko

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 31. 10. 2014

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 11. 11. 2014

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 14. 03. 2016

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Možnosti použití NoSQL databáze" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne _____

Poděkování

Rád bych touto cestou poděkoval Ing. Alexandru Vasilenkovi za vedení při psaní této práce a dále pak RNDr. Haně Havelkové za první impuls k tomu, abych se zajímal o nerelační databáze. V neposlední řadě bych rád poděkoval Paule Posio za vedení při psaní bakalářské práce na zahraniční univerzitě. Kiitos.

Možnosti použití NoSQL databáze

Utilization of NoSQL database

Souhrn

Tato diplomová práce se zabývá problematikou NoSQL databází, které se soustředí především na uchovávání a manipulaci velkých objemů dat. Tento typ databází nabízí uchovávání dat, která nejsou strukturovaná a nedají se proto umístit do tabulek.

Důvodem pro vznik této práce byla skutečnost, že se již delší dobu zajímám o alternativní způsoby uchovávání dat, než jsou relační databáze. Během svého bakalářského studia na Jihočeské univerzitě v Českých Budějovicích jsem napsal práci s názvem NoSQL databáze a tato diplomová práce na ní navazuje a dále tak rozšiřuje povědomí o alternativních databázích. Hlavním cílem je ukázat, kdy, proč a jak by se měla relační databáze převést na jednu z NoSQL databází. Na začátku je vysvětleno, co vlastně NoSQL znamená a jaké jsou neznámější typy těchto databází. Dále je nastíněno, jakým způsobem je možné převést relační databázi na jednu z nerelačních. Práce si také dává za cíl ukázat, jaké výhody a nevýhody plynou z přechodu na nerelační model. Na závěr je provedeno testování výkonu relačního a nerelačního modelu na příkladu a jejich porovnání.

Obsah této práce je rozdělen na dvě části – teoretickou a praktickou. Teoretická část je provedena pomocí rešerše dostupných informačních zdrojů. Praktická část je pak především zhotovena testováním databází na osobním počítači.

Tato diplomová práce může sloužit jako zdroj informací pro zájemce o NoSQL a také jako studijní materiál pro případné rozšíření výuky databází na PEF ČZU, kde se aktuálně z nerelačních databází vyučují pouze ty objektové.

Summary

This thesis deals with NoSQL databases, which focus mainly on storing and handling of large volumes of data. These databases store data that are not structured and therefore can not be placed in tables.

The reason for writing was the fact that I have been interested in alternative ways of storing data than relational databases for a long time. During my bachelor's degree at the University of South Bohemia in České Budějovice, I wrote a thesis called NoSQL database and this one builds on it and further extends awareness of alternative databases. The main goal is to show, when, why and how should a relational database be converted to one of NoSQL databases. At the beginning it is explained what NoSQL means and which databases belong among the most popular. Furthermore, it is outlined how it is possible to convert a relational database on one of the non-relational. The next objective is to show the advantages and disadvantages arising from the change of database model. At the end performance testing of the relational and non-relational model on an example is carried out and the results are compared.

The content of this work is divided into two parts - theoretical and practical. The theoretical part is done by analysis of information resources. The practical part is mainly done by database testing on a personal computer.

This thesis can serve as a source of information for those interested in NoSQL, as well as study material for possible expansion of a database study unit at PEF ČZU that is currently focusing only on Object databases from NoSQL.

Klíčová slova: MongoDB, MySQL, Databáze, Výkon

Keywords: MongoDB, MySQL, Database, Performance

Obsah

Seznam schémat

Seznam tabulek

1 Úvod

Když se řekne pojem Databáze, každý si pod ním dokáže něco představit. Potřeba uchovávat data nás již provází velmi dlouho, ať už je řeč o evidenci obyvatel, produktech v e-shopu nebo pouze prostém seznamu filmů. Předtím, než vůbec databáze vznikly, byly jejich předchůdci kartotéky, které uchovávaly informace v papírové formě. Veškeré operace s nimi pak prováděl přímo člověk. Tento přístup s rostoucím objemem dat začal později být neefektivní a tak se hledal způsob, jak převést zpracování dat na stroje. Tak vznikl děrný štítek, který byl zpracováván elektromechanickým strojem. Rozvoj počítačů v padesátých letech 20. století naznačil směr vývoje databází, kdy se ukázalo, že stávající řešení je nedostatečné. Později tak vznikly první síťové SŘDB na sálových počítačích. Netrvalo dlouho a objevily se i první relační databáze, které jsou dnes stále nejrozšířenější.

Vývoj jde ale stále kupředu a s tím i ruku v ruce data, která se uchovávají. Dnes se především větší společnosti často potýkají se situací, kdy pracují s velkým objemem dat v řádech stovek milionů záznamů. Tyto data často ani nebývají strukturovaná a znesnadňují tak tradiční správu dat pomocí schémat a relačních referencí. Tento fakt vedl ke vzniku dalších databázových systémů, které řeší problematiku uchování a zpracování takovýchto dat. Těm se celkově říká NoSQL. Pod tímto termínem se dá nalézt řada databázových systémů, především však sloupcově orientované databáze, databáze s klíčem a hodnotou, dokumentové a také objektové databáze.

Termín NoSQL je často vysvětlován jako akronym pro Not Only SQL. Jde o zastřešující název pro všechny databázové systémy, které nevyužívají relačního modelu. Některé z nich můžou využívat i jazyk SQL, který je známý ve spojení s relačními databázemi, ale často používají svůj vlastní.

Toto téma jsem si vybral z důvodu mého přetrvávajícího zájmu o databázové systémy. Vždy jsem obdivoval, jak rychle a efektivně mohou systémy pracovat, přestože se potýkají s rozsáhlými a komplikovanými daty, která navíc často bývají rozdělena na několika serverech. Vzhledem k tomu, že má bakalářská práce NoSQL databáze byla zaměřená na problematiku spíše obecně, chtěl jsem ji touto prací rozšířit a doplnit o nové postřehy a ukázat, jak se dá NoSQL v praxi využít.

2 Cíl práce a metodika

Cílem této práce bylo především analyzovat, jak uchovávat svá data jinak, než formou relačního modelu. Je zde také vysvětleno, kdy už je dobré začít uvažovat o NoSQL variantě a pomůže i s výběrem vhodného řešení. Na příkladu je dále ukázáno, jak převést databázi z mySQL do mongoDB, které je jedním z nejrozšířenějších nerelačních řešení, implementované řadou velkých společností, jako je například eBay, Foursquare nebo třeba The New York Times. Přechod na jiný systém má své výhody i nevýhody, které se práce také snaží zachytit. V praktické části je hlavní důraz kladen na otestování převedené databáze a porovnání výsledků s původním řešením v mySQL. Tato práce předpokládá základní znalosti z oblasti relačních databází, nejsou ale podmínkou pro pochopení základních aspektů práce.

Autor pro svojí rešeršní část získával informace o dané problematice převážně z odborných a vědeckých zdrojů z oblasti nerelačních databází. Kromě těchto zdrojů čerpal i ze svých osobních zkušeností s danou problematikou. Analytická část byla provedena na základě získaných informací a soustředí se na vybranou případovou studii, její návrh v relačním i nerelačním prostředí, optimalizaci, pro dosažení lepších výsledků a samotné otestování výkonnosti řadou testů, které simulují běžně kladené požadavky.

3 Teoretická východiska

První část práce se zabývá rešerší dané problematiky. Poněvadž tématem jsou možnosti použití NoSQL databáze, je nejprve vysvětleno, co se pod tímto termínem skrývá a poté následuje analýza současného stavu na poli nerelačních databází, spolu s nejznámějšími zástupci. Jedním z vůbec nejznámějších a nejrozšířenějších nerelačních řešení je mongoDB, kterému je v práci věnována největší pozornost a je také využit v praktické části. Teoretická část se dále soustředí na otázku, kdy je vhodné začít uvažovat o přechodu na nerelační řešení. Poslední kapitolou v této části je ukázání, jakým způsobem lze využít mongoDB v projektech, jako jsou internetové obchody a online hry.

3.1 Charakteristika NoSQL databází

NoSQL databáze se od SQL protějšku liší téměř v každém ohledu. Zatímco u SQL databází je zvykem uchovávat data v tabulkách, kde každý záznam má své předem stanovené atributy, cizí klíče, které odkazují na záznamy v jiných tabulkách a to vše musí být striktně dodržováno díky schématu databáze, NoSQL se na problém dívá jinak a nabízí více volnosti svým datům. Tyto databáze jsou nazývány „schema-less“, to znamená, že předem nejsou stanoveny pravidla pro to, jaké atributy budou záznamy mít a jak provázané záznamy budou mezi sebou, ale umožňuje uchovávat i ty, které se v atributech liší. Je tak tedy možné uchovat téměř jakoukoliv informaci.

NoSQL databázím ve většině případů chybí možnost propojovat více záznamů přímo při dotazování a to kvůli tomu, že data jsou často rozdělena na více serverech a jejich spojování by bylo časově náročné. Z tohoto faktu se dají vyvodit dva závěry – tyto databáze jsou určený především pro větší objemy dat, které již musí být z důvodu výkonu umístěny na několika serverech a také to, že se NoSQL příliš nehodí pro situace, kdy existuje logická souvislost mezi jednotlivými záznamy. To naznačuje, že největší využití naleznou při uchovávání například logů nebo komunikace uživatelů. Problém chybějící join funkce se ale dá vyřešit například v aplikační vrstvě a některé databáze nabízí i svá alternativní řešení.

NoSQL databáze podporují horizontální škálování. Tímto způsobem je možné rozdělit databázi na mnoho částí, rozmístit je na různé servery a dosáhnout tak vyššího výkonu. Tento přístup je efektivnější a cenově výhodnější než vertikální škálování, které využívá SQL. To spočívá v tom, že data jsou na jednom serveru, který se pro zvýšení výkonu upgraduje novými komponenty. Z důvodu škálovatelnosti a vyšší dostupnosti dat nepodporují NoSQL

u transakcí ACID (Atomicity, Consistency, Isolation, Durability), je tedy pro ně důležitější výkon, než konzistence. Většina systémů nepodporuje dotazovací jazyk SQL, místo něho využívají nízko úrovňové dotazovací jazyky, jako jsou C, Java nebo Erlang.[1]

3.1.1 Hlavní zástupci

Poněvadž NoSQL zastřešují databáze, které nevyužívají relační model, patří mezi ně mnoho zástupců. Ti se rozdělují do skupin podle toho, jaký volí přístup k datům. Mezi nejznámější systémy patří sloupcové databáze, dokumentové databáze, databáze s klíčem a hodnotou a objektové databáze. Existuje jich samozřejmě více, např. grafové a xml databáze, tato práce se ale věnuje pouze těm nejznámějším.

3.1.1.1 Sloupcové databáze

Sloupcově orientované databáze ukládají záznamy v podobě sloupců a ne řádek, jako to bývá zvykem u většiny relačních databází. Tento přístup představuje výhodu u datových skladů, CRM systémů atd., kde je často potřeba provést agregační funkci na mnoha řádcích a malém počtu sloupců. [1]

Pro lepší pochopení problému je vhodný příklad. Mějme zjednodušenou dvourozměrnou tabulku, která uchovává informace o zaměstnancích.

Id	Příjmení	Město	Pozice_id
1	Němcová	Praha	3
2	Rais	Český Krumlov	5
3	Mácha	Plzeň	1

Tabulka 1 Příklad SQL tabulky

Tuto dvourozměrnou tabulku nejdříve systém převede na jednorozměrnou podobu, aby s ní mohl pracovat. V řádkově orientovaných databázích dojde ke spojení všech hodnot v jednom řádku.

```
1, Němcová, Praha, 3;
2, Rais, Český Krumlov, 5;
3, Mácha, Plzeň, 1;
```

U sloupcově orientovaných databází ovšem nedochází ke spojení hodnot v řádcích, ale ke spojení hodnot ve sloupcích. Výsledné zpracování by tedy vypadalo takto:

```
1, 2, 3;
Němcová, Rais, Mácha;
Praha, Český Krumlov, Plzeň;
3, 5, 1;
```

Tento přístup se na první pohled může zdát být pouhým opakem pro řádkově orientované databáze bez zjevného benefitu. Opak je ovšem pravdou, protože pokud je databáze často dotazována např. na `Pozice_id` a dělá z nich statistiky obsazenosti pozic, jsou veškeré hodnoty již v jednom řádku a nabízí tak mnohem rychlejší zpracování, než kdyby dotaz musel procházet řádky všechny. Výkon by se také projevil v situaci, kdy by bylo potřeba změnit všechny hodnoty `Pozice_id`.

HBase

HBase je příkladem sloupcově orientovaných databází, který byl vyvinut jako součást Apache Hadoop projektu. Jde o open source framework napsaný v programovacím jazyce Java, schopný uchovávat rozsáhlé tabulky v řádech miliard řádků a milionů sloupců. Od roku 2010 Facebook využívá HBase pro spravování zpráv, které již mySQL nedokázalo zvládat z důvodu nedostatečného výkonu u rozsáhlých tabulek. [6]

3.1.1.2 Dokumentové databáze

Dokumentové databáze slouží pro uchovávání, spravování a získávání dokumentů z databází. Nejde ovšem o dokumenty typu Microsoft Word, nýbrž o volně strukturovanou sadu klíčů a hodnot, které jsou uskupeny v logických celcích. Tyto celky jsou nazývány dokumenty a jsou často reprezentovány pomocí JSON nebo XML. Příkladem takového JSON dokumentu může být:

```
{
Prijmeni: „Němcová”,
Mesto: „Praha”,
Pozice: „Účetní”
}
```

Tento dokument obsahuje 3 páry klíčů a hodnot, které spolu souvisí a dohromady tvoří ucelený záznam. V některých databázích, jako je např. mongoDB, se na tento celek dá často nahlížet jako na jeden záznam v relační databázi. [1]

MongoDB

MongoDB je nejznámějším zástupcem dokumentových databází a patří také mezi nejrozšířenější nerelační databáze vůbec. Každá databáze obsahuje kolekce, na které se dá pohlížet jako na tabulky a každá kolekce obsahuje mnoho dokumentů, reprezentující záznamy v tabulkách. MongoDB vzniklo v roce 2007 jako open source projekt a od té doby byl implementován řadou velkých společností, mezi které patří například eBay. Tato diplomová práce se soustředí především na mongoDB a jeho využití, proto mu bude v dalších kapitolách věnována odpovídající pozornost. [3]

3.1.1.3 Databáze s klíčem a hodnotou

Databáze s klíčem a hodnotou patří mezi nejjednodušší databáze vůbec. Tento typ slouží k uchovávání, spravování a získávání asociativních polí. Pod tímto pojmem si můžeme představit slovníky nebo hash, kde každý jednotlivý záznam se skládá z hodnoty, která může být v podobě kolekce, objektu nebo například prostého textového řetězce a klíče, který je jedinečný a jednoznačně ukazuje na určitou hodnotu v databázi. Pomocí tohoto klíče se dá pak snadno a rychle najít hledaná hodnota. Vzhledem k tomu, že typy uložených hodnot se můžou mezi sebou lišit, nabízí tyto databáze flexibilitu, chybějící v relačních databázích. Zároveň také poskytují i vyšší výkon díky jednoduchosti řešení a menším nárokům na operační paměť. Systém klíčů a hodnot je natolik efektivní, že je využíván i v jiných typech databází, které tento koncept přejímací a dále ho rozšiřují o nové a užitečné funkce. [1]

Redis

Redis je jedním z nejznámějších typů databází, které využívají klíčů a hodnot. Jde o open source řešení, které uchovává databázi v operační paměti počítače pro dosažení nejlepších výsledků při uchovávání, spravování a získávání záznamů. Pro trvalé uložení se samozřejmě využívá pevných disků, ze kterých je při startu Redisu načtena databáze do paměti. Jako hodnota může být využita řada datových struktur, mezi které patří například řetězec, spojový seznam, hash nebo set. Redis byl v průběhu let začleněn do řady systémů. Mezi nejznámější patří Twitter, GitHub, Snapchat nebo třeba StackOverflow. [1]

3.1.1.4 Objektové databáze

Objektové databáze uchovávají své informace v podobě objektů, stejně jako to dělají i objektově orientované jazyky. Cílem je umožnit uživateli pracovat s daty v databázi, aniž

by muselo docházet k rozkladu objektů na jednotlivé informace v něm uložené. Místo toho je objekt uložený jako celek a taky je s ním tak pracováno. Nejsou zde žádné tabulky, místo nich tu jsou objekty, včetně svých vlastností a místo řádků tu jsou samotné instance uchovávaných objektů. Každý z objektů je identifikován jednoznačným OID, který se chová jako ukazatel do virtuální paměti počítače. Objektové databáze také přejímají některé z funkcí objektově orientovaného programování, jako je zapouzdření, dědičnost a polymorfizmus. Pro získávání dat z databáze existuje dotazovací jazyk OQL (Object Query Language), který je syntaxí velmi podobný SQL jazyku a usnadňuje tak přechod z relačního modelu. Existují ale i další způsoby dotazování, které mohou být použity, jako je například QBE (Query By Example) nebo LINQ (Language Integrated Query). [1]

Db4o

Db4o je open source projekt, který je napsán v programovacím jazyce Java a C#. Jde pouze o knihovnu (JAR nebo DLL) malé velikosti, která je využívána klientskou aplikací. Nevyžaduje tedy žádnou samostatnou instalaci pro její používání. Db4o podporuje takzvané Native Queries místo OQL, které umožňují pracovat s databází pomocí programovacích jazyků, jako jsou Java, C# a VB.NET. Je ale také možné využít LINQ pro vytvoření objektově orientovaných dotazů. [1]

3.2 MongoDB

MongoDB je vybraným zástupcem za nerelační databáze, kterým se tato diplomová práce věnuje. V této části je vystižena jeho základní charakteristika, spolu s tím, jak lze tento systém nainstalovat a spravovat. Žádná databáze by se neobešla bez základních operací, které lze využít pro práci s daty. Jde především o vytváření, čtení, modifikaci a mazání. Všechny tyto operace jsou zahrnuty také v této části, spolu s jednoduchými příklady použití.

3.2.1 Charakteristika

MongoDB je dokumentová databáze, která využívá kolekcí pro uchovávání jednotlivých dokumentů. Kolekci můžeme chápat jako tabulku, které jsou v relačních databázích. Dokument je pak záznamem v takové tabulce. Stejně jako záznam musí být v tabulce, tak i dokument musí být v nějaké kolekci. Hlavním rozdílem mezi tabulkou a kolekcí je v tom, že kolekce nemá předem definované podmínky, které musí dokument splňovat. Jde především o atributy dokumentů, jejich počty a typy. Je tedy možné uchovávat strukturou velmi odlišné dokumenty ve stejné kolekci. To je umožněno díky tomu, že mongoDB má dynamické databázové schéma.

Toto schéma, někdy také nazývané flexibilní nebo v angličtině schema-less, je mezi NoSQL velmi rozšířené a značí, že na rozdíl od SQL, kde je nutné určit schéma tabulky před tím, než jsou do ní vloženy data, kolekce v mongoDB nevyžadují určitou strukturu v dokumentech. Je ovšem vhodné sdružovat dokumenty podobné struktury ve stejné kolekci z důvodů efektivního indexování a tím i vyššího výkonu. [3]

Dokumenty v mongoDB jsou reprezentovány pomocí formátu BSON, což je zkrácená verze pro Binary JSON (JavaScript Object Notation). Je to tedy binárně kódovaná reprezentace JSON formátu, který se skládá z klíčů a hodnot. Jako hodnota může být v BSON řada datových typů, mezi které patří například řetězec, binární data, boolean, datum, javascript, regulární výraz a mnohé další. Každý dokument je identifikován pomocí jedinečného id, které je buď generováno automaticky při vzniku dokumentu, nebo ho uživatel vytváří ručně. Toto id se může v kolekci vyskytnout nanejvýš jednou. [3]

3.2.2 Instalace a správa

Ve verzi mongoDB 3.2 existují dva způsoby, jak databázový systém nainstalovat a spravovat. První variantou je nainstalovat samotný systém a pak s ním pracovat pomocí

shellu nebo využít Cloud Manageru, který také umožňuje nainstalovat systém a navíc ho spravovat v prostředí webového prohlížeče s řadou užitečných funkcí. Obě metody je možné provést na většině operačních systémů, mezi něž patří Microsoft Windows Vista a vyšší, Linux, Mac OS X 10.7+ a Solaris. Tato práce se bude soustředit na instalaci a práci v prostředí Microsoft Windows.

3.2.2.1 MongoDB

Na stránkách mongoDB (https://www.mongodb.org/downloads#production) je možné stáhnout instalační soubor, který odpovídá operačnímu systému. Samotná instalace je velmi jednoduchá a nevyžaduje zásah uživatele. Defaultní instalační cesta je `C:\Program Files\MongoDB\Server\3.2.`

Po instalaci je nutné říct serveru, kam se mají ukládat data. Toho lze dosáhnout pomocí přepínače `dbpath` u souboru `mongod.exe`. Pokud by měly být data uchovávány na jiném disku, příkaz by mohl vypadat následovně:

```
C:\Program Files\MongoDB\Server\3.2\bin\mongod.exe --dbpath
D:\MongoDB\data
```

Kromě aplikace `mongod.exe`, která znamená mongo daemon, zpracovávající žádosti o data, a spravující přístup k datům, se v adresáři `bin` nachází ještě další důležité aplikace:

mongo.exe	Klientská aplikace. Pomocí ní se připojuje k serveru.
mongos.exe	Router aplikace. Stojí mezi klientskou aplikací a shardy, reprezentující části kolekce. Více o této problematice v Analytické části v sekci Sharding.
mongostat.exe	Nástroj pro rychlý náhled na stav aktuálně spuštěných mongod a mongos instancí.
mongotop.exe	Nástroj pro sledování času, které mongoDB strávilo čtením a zápisem dat
mongoperf.exe	Nástroj pro testování výkonu disku při čtení a zápisu.

mongoimport.exe, mongoexport.exe	Nástroje pro import a export dat z databáze. Podporované formáty jsou JSON a CSV.
mongodump.exe, mongorestore.exe	Nástroje pro vytvoření binárního obrazu databáze a obnovení databáze z obrazu.

Tabulka 2 Soubory pro práci s mongoDB

Výše uvedený příkaz by měl spustit server. V případě, že je vše v pořádku, v okně se zobrazí nápis `Waiting for connections.` Pokud došlo k chybě, zobrazí se zde popis chyby.

Pro připojení k serveru se využije klientská aplikace `mongo.exe` bez jakýchkoliv parametrů. Pokud server běží a připojení bylo navázáno, klientská aplikace nás uvítá s informací o verzi mongoDB shellu a aktuálně využívané databázi.

3.2.2.2 MongoDB Cloud Manager

MongoDB Cloud Manager je relativně nová služba, která je dostupná skrze webové stránky mongoDB (https://www.mongodb.com/cloud/). Nabízí třicetidenní free trial verzi, ve které je možné vyzkoušet většinu z funkcí Cloud Manageru. Mezi ně patří například zálohování pomocí replikace (více o této problematice v Analytické části v sekci Replikace), kompletní monitoring, vytváření a modifikace databází a další. Ve verzi Premium jde produkt ještě dále a nabízí možnost optimalizace dotazů a vytváření vhodných indexů. Jde tedy o službu, která umožňuje vytvářet, spravovat, monitorovat a zálohovat mongoDB databáze.

Proces využívání této služby začíná registrací na výše zmíněném odkazu. Po vyplnění osobních údajů je uživatel vyzván k přihlášení nebo vytvoření skupiny, která určuje, k jakým databázím má uživatel přístup.

Poté se již dostává do prostředí Cloud Manageru, kde má na výběr, zda chce začít pracovat s novou databází nebo využívat stávající. Při výběru nové se dále musí rozhodnout, zda má být databáze umístěna na lokálním počítači, na vzdáleném zařízení nebo v cloudovém řešení Microsoft Azure či Amazon web services.

Dalším krokem je zvolení, jaký typ databáze bude využit. Na výběr je Standalone, který je vhodný pro testování, Replica Set, který se využívá v situacích, kdy chceme mít stejná data na více strojích pro vyšší dostupnost a nebo Sharded Cluster, který se využívá především pro vyšší výkonnost databázových operací. Pokud databáze nemíří do produkčního prostředí, Standalone verze poskytuje dostatečný prostor pro běžnou práci.

Při zvolení Standalone možnosti je uživatel vyzván k určení názvu instance a místa, kam budou databázové soubory ukládány. Posledním krokem je instalace tzv. Automation agenta. Jde o nástroj, který musí být nainstalován na každém zařízení, využívající Cloud Manager. Ten má za úkol komunikovat s Cloud Managerem, aplikovat změny v databázi a odesílat hlášení o stavu databáze zpět do Cloud Manageru. Podporované jsou pouze 64 bitové procesory, není tedy možné využívat Cloud Manager na starších počítačích.

Po úspěšném dokončení instalace je Cloud Manager připraven k používání. Ten obsahuje informaci o hostname a čísle portu, které jsou nutné pro připojení k mongoDB. Samotné připojení se realizuje opět pomocí aplikace `mongo.exe`, které je součástí instalace Automation agenta. Tentokrát ale budou využity parametry z Cloud Manageru. Připojení k serveru by tedy vypadalo následovně:

```
mongo.exe --host <hostname> --port <číslo portu>
```

nebo zkrácenou verzí:

```
mongo.exe <hostname>:<číslo portu>
```

Po připojení již s databází můžeme pracovat a využívat funkce Cloud Manageru.

3.2.3 CRUD operace

CRUD operace jsou základní operace, které lze s daty provádět. Jde o zkratku Create, Read, Update a Delete, tedy o vytváření, vyhledávání, modifikace a mazání záznamů či jejich částí. Ve spojení s dalšími funkcemi mongoDB lze dosáhnout výsledků, které převyšují i schopnosti jazyka SQL.

3.2.3.1 Create operace

Pro vytváření nových dokumentů v mongoDB existují tři metody, které se dají použít: [8]

- `db.collection.insertOne()`
- `db.collection.insertMany()`
- `db.collection.insert()`

Z výše uvedeného je na první pohled vidět, že metody se volají na samotných kolekcích, které obsahují dokumenty. Místo „collection“ tedy bude název existující kolekce v databázi,

ve které chceme dokument vytvořit. Pokud zmíněná kolekce zatím neexistuje, je vytvořena taktéž.

Metoda `db.collection.insertOne()` vkládá pouze jeden dokument do definované kolekce. V praxi by mohla vypadat následovně: [8]

```
db.knizky.insertOne
(
      {
            nazev: "Naučte se C++ za 21 dní",
            autor: "Jesse Liberty "
      }
)
```

Z výše uvedeného příkladu stojí za povšimnutí několik věcí.

- Při psaní nezáleží na mezerách a odřádkování, je tedy možné udržovat příkaz v podobě, který nám vyhovuje.
- Dokumenty, které chceme vložit do kolekce, se předávají jako parametry metodám.
- Tyto dokumenty jsou na začátku a konci ohraničeny složenými závorkami.
- Každá hodnota v dokumentu kromě poslední musí být od další oddělena čárkou.
- _id není v příkladu definováno, bude tedy vytvořeno automaticky.

Stejný příklad se dá samozřejmě zapsat i v jazyce SQL. I zde je zápis velmi jednoduchý. [5]

```
INSERT INTO knizky (nazev, autor)
VALUES („Naučte se C++ za 21 dní“, „Jesse Liberty“)
```

Metoda `db.collection.insertMany()` slouží ke vkládání jednoho a více dokumentů do kolekce. Zápis takové metody by vypadal následovně: [8]

```
db.knizky.insertMany
(
      [
            {nazev: "Naučte se C++ za 21 dní", autor: "Jesse Liberty"},
            {nazev: "PHP a MySQL", autor: "Miloslav Ponkrác"}
      ]
)
```

V tomto příkladu se do kolekce `knizky` vkládají dva dokumenty. Celý blok dokumentů je opatřený hranatými závorkami, které značí pole dokumentů. Jednotlivé dokumenty jsou pak mezi sebou odděleny čárkou, stejně jako hodnoty vně dokumentu.

Metoda `insertMany()` se dá opět interpretovat do jazyka SQL v podobě: [5]

```
INSERT INTO knizky (nazev, autor)
VALUES („Naučte se C++ za 21 dní“, „Jesse Liberty“),
(„PHP a MySQL“, „Miloslav Ponkrác“)
```

Poslední metodou na vytváření nových dokumentů je `db.collection.insert()`. Jde o univerzální metodu, která je kombinací obou předchozích přístupů. Umožňuje tedy přidat jak jeden dokument, tak i pole více dokumentů. [8]

3.2.3.2 Read operace

Read operace neboli dotazování, získává data z databáze na základě předem specifikovaných parametrů. Stejně jako v případě vytváření dokumentů se operace provádí na určité kolekce. Dotaz tedy může pracovat pouze s jednou kolekcí, není možné kolekce v dotazu propojovat.

Pro dotazování existují v mongoDB dvě metody: [7]

- `db.collection.find()`
- `db.collection.findOne()`

Metoda `find()` jako součást svého parametru přijímá podmínky, které musí dokument splnit, aby byl vrácen uživateli. Dále také může specifikovat, které z položek v dokumentu se mají vrátit a které pro nás nejsou důležité. Výsledný seznam odpovídajících dokumentů s vybranými položkami pak lze ještě omezit pomocí metody `limit()`, vrátit až od určitého dokumentu pomocí metody `skip()` a nebo seřadit pomocí metody `sort()`. [7]

Předpokládejme, že máme v kolekci `knizky` tři dokumenty:

```
{nazev: “Visual C#”, autor:”John Sharp”, rokVydani: 2009},
{nazev: “Učebnice jazyka Java”, autor:”Pavel Herout”, rokVydani: 2008},
{nazev: ”PHP – Objektově orientované”, autor:”Peter Lavin”, rokVydani:
2010}
```

Z této kolekce bychom chtěli vybrat názvy knížek a autorů, které byly napsané od roku 2009 a seřadit výsledek vzestupně podle názvu knížky. Dotaz, který vybere odpovídající dokumenty, by pak vypadal následovně:

```
db.knizky.find({rokVydani: {$gte:2009}}, {_id: 0, nazev: 1, autor:
1}).sort({nazev:1})
```

V první části dotazu je specifikována podmínka, která udává, že rok vydání by měl být stejný nebo vyšší než 2009. Toho je dosaženo pomocí operátoru `$gte` (Greater Than or Equal), který porovnává hodnotu a pokud odpovídá podmínce, tak dál předává dokument ke zpracování. Kromě operátoru `$gte` existuje ještě řada dalších. Mezi další komparační operátory patří například `$lt` (Less Than), `$eq` (Equal) a `$in`, který zjišťuje, zda pole obsahuje některou z hodnot. Jsou ale i jiné operátory, než komparační. Jejich kompletní seznam lze nalézt na:
https://docs.mongodb.org/manual/reference/operator/query/#query-selectors. [7]

V druhé části dotazu se specifikuje, jaké části dokumentu jsou pro nás relevantní, a chceme je ve výsledku zobrazit. Pokud je u položky číslo jedna, znamená to, že se má ve výsledku zobrazit. Pokud je tam nula nebo položka v seznamu není zmíněna, ve výsledku se nezobrazí. Výjimečný případ představuje položka `_id`, která se zobrazuje vždy, pokud jí není přiřazena nula.

Poslední částí dotazu je metoda `sort()` s parametrem položky, podle které se má výsledek seřadit. Pokud je položce přiřazená hodnota jedna, jde o vzestupné řazení, nula značí řazení sestupné.

Po provedení zmíněného dotazu se ve výsledku zobrazí dvě knihy – Visual C# a PHP – Objektově orientované, spolu s jejich autory. Knížka Učebnice jazyka Java neprošla přes podmínku, protože rok vydání je nižší než 2009. Ve výsledku se u obou knížek nezobrazuje rok vydání, protože nebyl explicitně jmenován v druhé části dotazu. Nezobrazí se ani `_id`, protože bylo zakázáno.

Stejného výsledku by se dalo dosáhnout i v relačních databázích pomocí jazyka SQL a to následovně: [5]

```
SELECT nazev, autor
FROM knizky
WHERE rokVydani >=2009
ORDER BY nazev ASCENDING
```

Druhou metodou pro dotazování je metoda `findOne()`. Tato metoda vrací pouze první výskyt dokumentu, který odpovídá podmínce v parametru. Je tedy jakousi variací na metodu `limit()` s parametrem jedna, kterou se také omezuje výsledek pouze na první výskyt odpovídajícího dokumentu. [7]

3.2.3.3 Update operace

Update operace slouží k modifikacím stávajících dokumentů. Jako předešlé typy operací se i update provádí vždy na konkrétní kolekci. V mongoDB existují čtyři způsoby, jak modifikovat již existující dokumenty: [8]

- `db.collection.updateOne()`
- `db.collection.updateMany()`
- `db.collection.replaceOne()`
- `db.collection.update()`

Prvním způsobem je metoda `updateOne()`. Pomocí ní je možné modifikovat první dokument, který odpovídá zadaným kritériím v parametru metody.

Předpokládejme existenci tří dokumentů v kolekci `rezervace`:

```
{jmeno:"Tomáš Panyko", cisloR:"1", stav:"rezervováno"},
{jmeno:"Tomáš Panyko", cisloR:"2", stav:"zrušeno"},
{jmeno:"Alois Jirásek", cisloR:"3", stav:"rezervováno"}
```

Úkolem je zrušit rezervaci, která má číslo jedna. Taková by se v kolekci měla vyskytovat pouze jedna, protože se čísla inkrementálně navyšují a neopakují. Zde se vyplatí použít metodu `updateOne()`, protože jakmile se odpovídající dokument nalezne a upraví, metoda dále nepokračuje v hledání a je tudíž rychlejší. Zápis takovéto operace by pro mongoDB vypadal následovně: [8]

```
db.rezervace.updateOne({cisloR:1}, {$set: {stav:"zrušeno"}})
```

Parametr této metody se skládá ze dvou částí – podmínky a akce, která se má provést. V podmínce se určí, co by měl dokument splňovat, aby mohl být modifikován. V tomto případě musí mít číslu rezervace přiřazenu jedničku. Druhá část říká, jak se má dokument změnit. K tomu se využívá operátor `$set`. Ten nahrazuje hodnotu určité položky jinou. Zde dochází k náhradě stávající hodnoty „rezervováno“ na „zrušeno“. Kromě operátoru `$set` existuje i řada dalších, mezi nimiž jsou například `$rename` pro změnu názvu položky, `$unset` pro odstranění položky nebo třeba `$currentDate`, které přiřadí položce aktuální datum. Kompletní seznam operátorů využitelných pro modifikaci se dá nalézt na: https://docs.mongodb.org/manual/reference/operator/update/.

Stejná operace by se dala v jazyce SQL pro relační databáze provést následovně: [5]

```
UPDATE rezervace
SET stav = 'zrušeno'
WHERE cisloR = 1
```

Druhou metodou je `updateMany()`. Ta pracuje na naprosto stejném principu jako metoda `updateOne()` s tím rozdílem, že modifikuje všechny dokumenty, které odpovídají podmínce v parametru a zabere tudíž mnohem více času, protože musí projít všechny dokumenty v kolekci. [8]

Třetí metodou je `replaceOne()`. Pomocí ní je možné nahradit obsah celého dokumentu podle zadané podmínky v parametru. Stejně jako v případě metody `updateOne()` dojde k nahrazení pouze prvního dokumentu, který odpovídá a následně metoda skončí.

Mějme za úkol u rezervace číslo jedna nahradit jméno za "Karel Čapek" a ponechat ostatní položky nezměněné. V mongoDB by zápis metody `replaceOne()` vypadal následovně: [8]

```
db.rezervace.replaceOne
(
      {cisloR:1}, {jmeno: "Karel Čapek", cisloR:1, stav:"rezervováno"}
)
```

V první části parametru se definuje podmínka, kterou musí dokument splňovat, aby mohl být nahrazen. Druhá část obsahuje celý dokument, kterým bude původní nahrazen. Nejde tedy pouze o položku se jménem, která jako jediná bude mít ve výsledku jinou hodnotu, ale o celou strukturu dokumentu.

Stejná činnost se dá provést i v jazyce SQL. Zápis by vypadal takto: [5]

```
UPDATE rezervace
SET   jmeno = 'Karel Čapek'
      cisloR = 1
      stav = 'rezervováno'
WHERE cisloR = 1
LIMIT 1
```

V případě SQL jsou zde redundantní kroky v podobě přepisu čísla rezervace a stavu stejnými hodnotami, které tam již jsou. Mohou tedy být z části `SET` vynechány.

Posledním způsobem modifikace dokumentů je metoda `update()`. Ta stojí na pomezí mezi `updateOne()` a `updateMany()`. Ve výchozím nastavení mění pouze jeden dokument, ale pokud se do parametru metody jako třetí část přidá `multi:true`, tak budou upraveny všechny dokumenty, které splňují podmínku. [8]

Pokud bychom chtěli například pozměnit u všech rezervací na jméno „Tomáš Panyko“ stav na „zrušeno“, tak by zápis v mongoDB s využitím metody `update()` vypadal takto: [8]

```
db.rezervace.update({jmeno: "Tomáš Panyko"}, {$set: {stav:"zrušeno"}},
{multi:true})
```

Struktura parametru je shodná s metodami `updateOne()` a `updateMany()`, všechny mají podmínku a akci, která se má s dokumentem provést. `Update()` má ovšem navíc další parametr, určující, zda bude pozměněn jeden dokument (`multi:false`) nebo všechny (`multi:true`), které odpovídají podmínce.

3.2.3.4 Delete operace

Pro mazání dokumentů z kolekce existují v mongoDB tři metody: [8]

- `db.collection.deleteOne()`
- `db.collection.deleteMany()`
- `db.collection.remove()`

První z metod pro odstranění dokumentů je `deleteOne()`. Jak již název napovídá, půjde opět o metodu, která odstraňuje pouze první výskyt odpovídajícího dokumentu v kolekci a následně svojí činnost ukončí.

Předpokládejme, že máme kolekci `rezervace` se třemi dokumenty, které byly vytvořeny v předešlé části. Úkolem je smazat rezervaci, která má číslo jedna. Oproti update operacím nepůjde o změnu stavu rezervace, ale úplné smazání daného dokumentu. Zápis takovéto metody by v mongoDB mohl vypadat následovně: [8]

```
db.rezervace.deleteOne({cisloR:1})
```

Tato metoda je velmi jednoduchá a vyžaduje pouze jeden parametr – podmínku, kterou musí dokument splnit, aby byl smazán. Pokud by se v kolekci vyskytlo více dokumentů, které mají číslu rezervace přiřazenu jedničku, bude smazán pouze první výskyt v kolekci.

V jazyce SQL je zápis také velmi prostý. Stejného efektu by se dalo dosáhnout následujícím způsobem: [5]

```
DELETE FROM rezervace
WHERE cisloR=1
LIMIT 1
```

Obdobně pracuje i druhá metoda, `deleteMany()`. Ta na rozdíl od předchozí metody odstraní všechny dokumenty v kolekci, které odpovídají podmínce. Bylo by tedy velmi jednoduché smazat například všechny rezervace, které jsou vedeny na určité jméno. Jediným rozdílem by byla odlišná hodnota parametru metody. [8]

Poslední metodou je `remove()`. Ta se ve výchozím nastavení chová jako metoda `deleteMany()`. Je ovšem schopná přijmout i druhý parametr `justOne:true`, který způsobí, že se metoda začne chovat jako `deleteOne()` a smaže pouze první výskyt odpovídajícího dokumentu. Pokud druhý parametr definovaný není, vnitřně je mu přiřazena hodnota `false`. [8]

3.3 Přechod na NoSQL řešení

Na otázku, kdy se již vyplatí přejít z relačního řešení na jednu z variant NoSQL, není snadné jednoznačně odpovědět. Je třeba si uvědomit, že ne všechny projekty jsou vhodnými kandidáty na NoSQL, naopak některé z nich by mohly utrpět, co do výkonu, tak i použitelnosti. NoSQL sebou přináší jistá omezení, která na druhou stranu kompenzuje v jiných směrech. Důležitým předpokladem pro rozhodování je znalost toho, jaká data se budou uchovávat a jaké činnosti s nimi budeme provádět. Pouze na základě důkladné analýzy je možné učinit rozhodnutí, zda je vhodné přejít na nerelační alternativu a jaké konkrétní řešení zvolit.

Často již samotný formát dat může naznačovat, jakým způsobem by se mělo s daty zacházet. Relační databáze pracují se strukturovanými daty. Ty mají předem definované atributy, jejichž počet a ani formát se nemění. Tento přístup přináší určitou úroveň konzistence dat. Na druhou stranu nerelační databáze nevyžadují strukturovaná data a je tedy snadné je jakkoliv modifikovat. V případě, že se potýkáme se situacemi, kdy bychom potřebovali více flexibility od databáze, aby byla schopna uchovávat data bez nutnosti rozsáhlých změn ve strukturách tabulek, jde o první indikaci toho, že stávající model není dostatečný a že NoSQL by mohlo nabídnout efektivnější řešení. [4]

Dalším faktorem, který hraje roli při rozhodování o výběru databáze, je, jak rozsáhlá data se mají uchovávat. NoSQL databáze se většinou využívají pro uchovávání a práci s velkými objemy dat, u kterých by relační databáze nenabízely dostatečný výkon. Nutno ovšem podotknout, že v naprosté většině případů postačí relační databáze, která se vhodně nakonfiguruje a v případě nutnosti i upgraduje hardware serveru. Tomuto přístupu se říká vertikální škálování. Ten spočívá v tom, že databáze běží pouze na jednom stroji a pro navýšení výkonu se přidávají operační paměti, rychlejší pevné disky a výkonnější procesory. Náklady jsou ovšem poměrně vysoké a i přestože SQL dokáže škálovat velmi dobře, tak i s výkonným hardwarem se nakonec projeví omezení. NoSQL škáluje horizontálně, což znamená, že databáze je rozdělena na části a rozmístěna mezi více serverů. Tím, že server zpracovává pouze část databáze, lze dosáhnout lepších výsledků a to i bez nutnosti upgradu hardwaru. Přidáváním dalších serverů s částmi databáze do clusteru lze škálovat téměř bez omezení. [1]

NoSQL sebou přináší i jisté nevýhody. Vzhledem k tomu, že je databáze rozdělena na více částí, které navíc mohou být na geograficky velmi vzdálených místech, propojování jednotlivých záznamů mezi sebou by bylo časově velmi náročné. Také proto je funkcionalita většiny NoSQL databází v porovnání s SQL velmi omezená a neobsahuje některé z funkcí, které se považují za samozřejmé, jako jsou joiny a transakce. Některé z funkcí jsou řešeny odlišným způsobem a jiné jsou vynechány úplně. Výčet funkcí se ale mezi jednotlivými NoSQL řešeními liší a vyžadují důkladnou analýzu před výběrem. Pokud tedy prim hraje výkon databáze a funkcionalita je v pozadí, NoSQL nabízí vhodnou alternativu v podobě řady svých řešení. [1]

Přechod na jiný databázový systém má i další nevýhodu a tou je finanční náročnost. Se samotnou změnou technologie, která je v případě NoSQL velmi odlišná, jdou ruku v ruce i různá školení a kurzy, které jsou časově náročné a často i drahé. MongoDB má zde výhodu v tom, že nabízí několik online kurzů zdarma, mezi něž patří například správa mongoDB pro databázové administrátory, pro vývojáře v jazyce Java, prostředí .NET a další.

NoSQL je vhodné pouze pro specifické situace. Je dobré tam, kde se pracuje s velmi rozsáhlými daty a kde potřeba výkonu přebíjí potřebu některých funkcí, které známe z jazyka SQL. Řada projektů si naprosto vystačí s relačními databázemi, protože nabízí známé prostředí a dostatečný výkon. Jakmile se ovšem začnou data rozrůstat a nebude snadné je uchovávat v předem definovaných tabulkách, tak se vyplatí začít uvažovat o přechodu na NoSQL řešení.

3.4 Příklady využití mongoDB

V této sekci je ukázáno, jakými způsoby a kde by se dalo mongoDB využít. Jako příklady jsou využité online hraní a internetový obchod. V obou případech se dá těžit z možností, které přináší mongoDB. Každý z příkladů uvádí, jaké jsou na něj kladeny požadavky a jak je možné tyto požadavky uspokojit. Kromě uvedených příkladů využití existuje i řada reálných společností, které již daný systém implementovaly, mezi něž se řadí například LinkedIn, eBay, Bosch, O2 a mnoho dalších.

3.4.1 Online gaming

Jako první příklad je uvedené online hraní. V tomto případě se uchovávají relativně rozsáhlé kolekce, které slouží pro uchovávání informací o hráčích a jejich postavách, ale také o okolním světě. Jde o komplexní příklad, který vychází z reálných potřeb.

3.4.1.1 Požadavky

Při návrhu modelu spojeného s hraním online her se budou uchovávat především různé informace týkající se postav hráčů. U žánru RPG (Role-Playing Game) pak půjde hlavně o:

Atributy hráčovo postavy – Obsahují základní informace o postavě, jako je její síla, obratnost, životy, charisma a často také povolání či rasu.

Hráčův inventář – Ve většině her existují předměty, které může hráč sebrat a nějakým způsobem využít. Databáze tedy musí být schopna uchovávat informaci o tom, jaké předměty má hráč ve svém vlastnictví.

Lokace hráče v herním světě – Ve většině her je také možné se pohybovat na více mapách, které dohromady tvoří herní svět. Musí být tedy zaznamenáváno, kde se hráč nachází, aby se dalo určit, jaké možnosti mu prostředí nabízí.

Herní žánr RPG je velmi oblíbený, často obsahující tisíce i více hráčů, kteří hrají ve stejný čas. Pro každého z nich je nutné ukládat výše zmíněné informace ve formě, která je hře čitelná a s minimálním zpožděním, aby byla zajištěna plynulost během hraní.

Kromě hráčovo inventáře je však také nutné odděleně uchovávat, jaké předměty ve světě existují a s jakými může hráč provádět interakci. Výčet předmětů často zahrnuje různé zbraně, brnění, vzácné předměty, atd.

Odděleně se uchovávají i informace o všech lokacích, které se ve hře nacházejí a ve kterých se hráč, ale i samotné předměty mohou vyskytovat. Tato kolekce pak tvoří herní svět.

Při návrhu modelu, který bude sloužit pro uchovávání informací z online her, je vhodné vzít v potaz i flexibilitu samotného modelu. Především ve fázi testování nebo v rámci zpětné vazby od hráčů se mohou vyskytnout žádosti o změny, které by zasahovaly do modelu. Pro takové případy je velmi vítané, pokud lze model snadno upravit a spolu s ním i stávající data s minimálními časovými nároky. [2]

3.4.1.2 Návrh řešení

Konečný model úzce souvisí s konkrétní hrou a je výsledkem analýzy všech aktuálních a možných budoucích požadavků. Pro minimalizaci počtu nutných dotazů do databáze se často využívá zapouzdření běžně používaných informací do malého počtu atributů. Řešení bude obsahovat tři kolekce – kolekci hráčských postav, předmětů a lokací. V RPG by schéma dokumentu, který bude uchovávat informace o hráčovo postavě, mohl vypadat následovně: [2]

```
{
_id: ObjectId('...'),
prezdivka: 'Jerry',
postava: {
zaklInfo: { sila: 10, obratnost: 12, inteligence: 15, charisma: 10 },
trida: 'warrior', zivoty: 200 },
lokace: {
id: 'tunel-1',
popis: 'Temný tunel, ve kterém není vidět ani na krok',
cesty: {s:'ulice-1', j:'tunel-2', v:'tunel-3'},
hraci: [
{ id:ObjectId('...'), prezdivka:'Tom' }, { id:ObjectId('...'),
prezdivka:'Jerry' } ],
inventar: [
{ id:ObjectId('...'), pocet:1, name:'pochoden' }] },
penize: 523,
brneni: [
{ id:ObjectId('...'), pozice:'hlava'},
{ id:ObjectId('...'), pozice:'telo'},
{ id:ObjectId('...'), pozice:'nohy'}],
zbrane: [ {id:ObjectId('...'), drzeni:'obourucni'} ],
inventar: [
{ id:ObjectId('...'), nazev:'batoh', pocet:1, obsah: [
{ id:ObjectId('...'), nazev:'lektvar léčení', pocet:3},
{ id:ObjectId('...'), nazev:'svitek teleportace', pocet:2} ]},
{ id:ObjectId('...'), nazev:'kožená přilbice', pocet:1, bonus:3},
{ id:ObjectId('...'), nazev:'kroužkovaná zbroj', pocet:1, bonus:5},
{ id:ObjectId('...'), nazev:'plátěné boty', pocet:1, bonus:1},
{ id:ObjectId('...'), nazev:'tupá sekera', pocet:1, bonus:3} ]
}
```

Za povšimnutí zde stojí několik věcí. Pozice hráče je přímo uložena jako součást dokumentu v položce `lokace`. Spolu s pozicí jsou zde i další informace o ostatních hráčích, kteří se vyskytují na stejném místě, včetně předmětů, které se v dané lokaci také nachází. Smyslem tohoto zapouzdření v jednom dokumentu je, aby bylo možné zpracovat umístění hráče, aniž by byl proveden další dotaz do jiné kolekce pro získání informací o lokaci.

Položky `brnění` a `zbraně` obsahují jen základní informaci o tom, že hráč má nějaké vybavení. Popis těchto předmětů s dalšími vlastnosti je proveden až v `inventáři`. Vzhledem k tomu, že se vše nachází v jednom dokumentu, není třeba mít duplicitní popis předmětů i v `brnění` a `zbraních`.

Položka `inventář` obsahuje popis předmětů, které hráč vlastní, včetně jejich vlivů na postavu hráče a počtu předmětů v inventáři. Stejně jako u `lokace` jsou zde i informace o předmětech zapouzdřeny do dokumentu o hráči za účelem omezení nutného počtu dotazů pro získání potřebných informací o obsahu inventáře.

Schéma dokumentu z kolekce, která bude uchovávat všechny informace o předmětech dostupné ve hře, by mohl vypadat následovně: [2]

```
{
_id: ObjectId('...'),
nazev: 'batoh',
bonus: null,
obsah: [
{ id:ObjectId('...'), nazev:'lektvar léčení', pocet:3},
{ id:ObjectId('...'), nazev:'svitek teleportace', pocet:2} ]},
vaha: 10,
cena: 130,
...
}
```

Tento dokument obsahuje téměř identické informace v porovnání s položkou `inventář` v dokumentu o postavě hráče. Navíc ale obsahuje i doplňující informace, které jsou při hraní zapotřebí pouze sporadicky.

Posledním návrhem schématu dokumentu je lokace. Ten by mohl vypadat následovně: [2]

```
{
id: 'tunel-1',
popis: 'Temný tunel, ve kterém není vidět ani na krok',
cesty: {s:'ulice-1', j:'tunel-2', v:'tunel-3'},
hraci: [ { id:ObjectId('...'), prezdivka:'Tom' }, { id:ObjectId('...'),
prezdivka:'Jerry' } ],
inventar: [
{ id:ObjectId('...'), pocet:1, name:'pochoden' }]
}
```

Toto schéma uchovává přesně ty samé informace, které jsou uložené i v `lokaci` ve schématu hráčovo postavy. Jejím úkolem je uchovávat stav, v jakém se herní svět nachází. Také se bude používat pro situace, kdy hra bude vyžadovat interakci mezi více uživateli.

3.4.2 E-shop

Další zajímavou ukázkou využití mongoDB je možné demonstrovat na elektronickém obchodě. Tato kapitola se zaměří konkrétně na produktový katalog společnosti, která prodává elektroniku online.

3.4.2.1 Požadavky

Na rozdíl od předchozího modelu, kde všechny dokumenty popisující postavu hráče, její předměty a lokaci, mají strukturu prakticky totožnou, v katalogu produktů se vyskytují předměty, které se mezi sebou mohou výrazně lišit. Například taková televize bude mít odlišné atributy v porovnání s kávovarem. Zatímco u televize nás bude zajímat především uhlopříčka a rozlišení, u kávovaru to bude objem a příkon.

Model tedy musí být schopen zachytit i tyto rozdíly mezi produkty. Mezi sebou ovšem sdílí i podobné atributy, jako jsou například cena, rozměry, záruka a zda jsou skladem. Základním kamenem je tady flexibilita, která dovolí uchovávat různé druhy elektroniky.

3.4.2.2 Návrh řešení

Poněvadž je mongoDB nerelační databází, produktový katalog může těžit z dostupné flexibility. Nejvhodnějším modelem je využití jedné kolekce, která bude uchovávat dokumenty, obsahující veškeré informace k danému produktu. Na začátku dokumentu model obsahuje obecné produktové informace, které jsou sdílené mezi všemi produkty. Každý produkt je určitého typu, výrobce, modelu, atd. Ukládá se zde i počet kusů na skladě, aby stejný dokument nebyl v kolekci vícekrát. Tím se také sníží potřebné místo na pevném disku. Po nákupu produktu se tedy upraví v kolekci jen zbývající počet na skladě. Součástí obecných informací je i cena produktu, podle které se ve většině případů bude zboží řadit. Kromě obecných informací se do dokumentu musí zahrnout i informace, které jsou specifické pro danou kategorii. Ty budou obsaženy v subdokumentu `detaily`, popisující produkty stejného typu. U televizí jde především o energetickou náročnost, rozlišení, technologii a další. Následující dokument popisuje právě takovou televizi. [2]

```
{
_id: ObjectId('...'),
typ: "Televize",
vyrobce: "Thomson",
model: "32HA3103",
popis: "Televize LED, úhlopříčka 81cm, 100Hz, 1366x768…",
pocetKusu: "5",
skladem: "Ano",
vaha: 6,
rozmery: {
sirka: 73.7,
vyska: 48.3,
hloubka: 20.2
},
cena: {
puvodni: 5990,
poSleve: 4999,
sleva: 991,
sleva_procenta: 16.5
},
detaily: {
energetickaNarocnost: "A+",
rozliseni: "HD Ready",
technologie: "LED",
os: "Bez podpory SMART",
typTuneru: "DVB-T, DVB-C",
},
}
```

Kávovar by měl stejné položky pro popis obecných informací o produktu, ale odlišné položky v subdokumentu `detaily`. Ten by vypadal následovně: [2]

```
detaily: {
objemVody: 2,
prikon: 1450,
casovac: "ANO",
}
```

4 Analytická část

V této sekci se diplomová práce zabývá praktickou částí. V ní bylo za úkol navrhnout databázový model v mySQL a ukázat, jak by se takový model měl správně převést do dokumentové databáze mongoDB. Důležitý je zde vhodný výběr schématu, který zásadně určí, jak bude výsledná databáze efektivní. Přechod z jednoho systému na jiný s sebou přináší i určité výhody a nevýhody, které jsou v této sekci také vystihnuty. Pokud by databázový systém byl nasazen v produkčním prostředí, replikace dat a horizontální škálování by hrálo velkou roli v dostupnosti dat a celkové výkonnosti databáze, proto se jim práce také věnuje. Vhodná indexace vychází z potřeb databáze a měla by reflektovat nejčastěji prováděné dotazy. Jelikož je nezbytnou součástí každé implementace, je zahrnuta i zde. Poslední částí, kterou se práce zabývá, je samotné testování výkonnosti mySQL a mongoDB databáze na vytvořeném projektu, pomocí několika vybraných operací. Tyto operace si kladou za cíl otestovat ty nejčastější situace, se kterými by se databáze mohla setkat. Aby obě databáze mohly být snáze porovnatelné, replikace a horizontální škálování nebyli do fáze testování zahrnuty.

4.1 Konverze mySQL na mongoDB

Pro ukázku, jak správně by se měla provést konverze mySQL databáze do mongoDB, byl zvolen projekt, který simuluje Česko-Slovenskou Filmovou Databázi (ČSFD). Na jejich stránkách lze nalézt mimo jiné rozsáhlý katalog filmů a seriálů, který k únoru 2016 čítá přes 425 000 děl. U každého z nich se uchovává jeho název v českém i anglickém jazyce, délka v minutách, rok a místo natáčení, jméno režiséra, seznam herců a žánrů, kterým dílo odpovídá a celkové hodnocení filmu. Toto hodnocení, reprezentované procentuálně, vychází z jednotlivých hodnocení uživatelů. Tito uživatelé kromě samotného hodnocení filmu či seriálu mohou i dílo okomentovat a jejich příspěvek je pak zobrazen v sekci komentářů.

Projekt ČSFD je ovšem velmi rozsáhlý a kromě výše zmíněného katalogu obsahuje i řadu dalších sekcí, jakou jsou trailery, televizní programy, atd. Bylo by velmi obtížné simulovat celý projekt, proto se konverze zaměří výhradně na katalog filmů a seriálů. Jak již bylo zmíněno dříve, nerelační databáze jsou mimo jiné vhodné pro velké objemy dat, a proto bude celkový počet děl v katalogu navýšen na 10 milionů, ke kterým se bude uchovávat 100 milionů komentářů, náhodně rozdělených mezi filmy a seriály, aby více vynikla výsledná efektivita konverze.

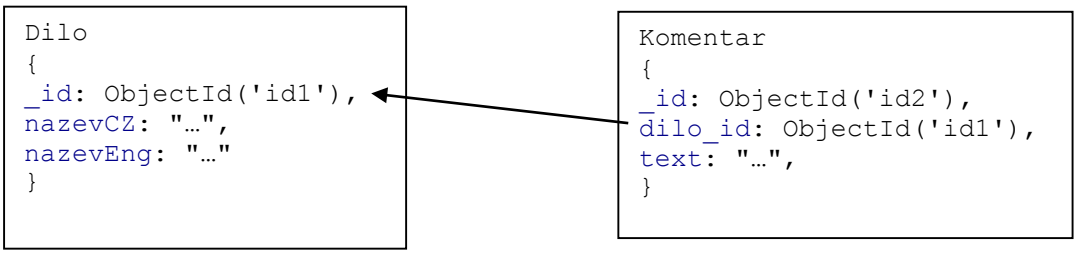
4.1.1 Výběr vhodného schématu

Při návrhu nového modelu je nezbytně nutné vzít v úvahu, jakým způsobem aplikace pracuje s daty. Jde především o snahu optimalizovat čas, potřebný pro vykonání operací typu read. Model by se tak měl odvíjet od potřeb aplikace, stejně jako od logické struktury dat.

Součástí návrhu modelu je i rozhodnutí, jakým způsobem budou uchovávány vztahy mezi jednotlivými dokumenty. Ve světě relačních databází se vztahy reprezentují pomocí cizích klíčů, které slouží jako ukazatel na jiný záznam. Pomocí joinů se pak dají odpovídající záznamy propojit a vytvořit tak zamýšlené vztahy. V mongoDB ovšem žádné cizí klíče ani jazyk SQL není, proto je nutné vztahy reprezentovat jiným způsobem. Pro tyto potřeby existují dva přístupy – reference a zapouzdření.

Reference ukládají své vztahy mezi dokumenty formou odkazů, tj. přidáváním informace o tom, s jakým jiným dokumentem souvisí. Tento přístup je podobný řešení relačních databází, kde se vztahy tvoří pomocí cizích klíčů. Poněvadž ale v mongoDB neexistuje operace join, propojení se vytváří až v aplikační vrstvě. Vytváření vztahů pomocí referencí se považuje za normalizovaný datový model. Samotný zápis reference může být proveden dvěma způsoby – pomocí manuálních referencí nebo DBRefs.

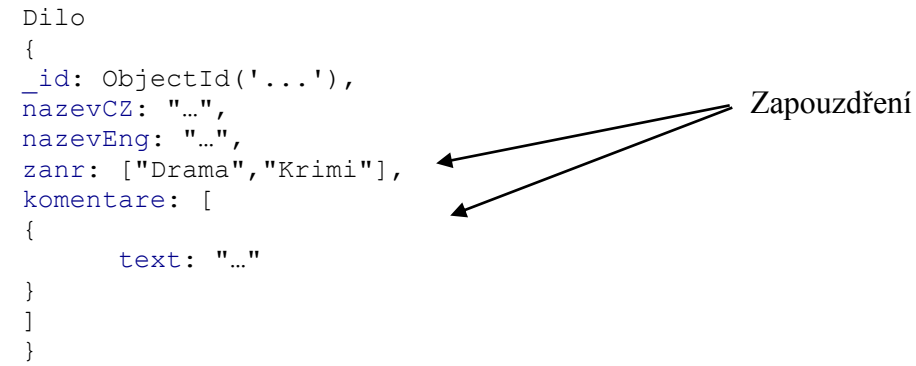
Manuální reference ukládají `_id` položku jednoho dokumentu v jiném dokumentu jako odkaz. Tento přístup je jednoduchý a dostatečný pro většinu případů.



DBRefs kromě položky `_id` ukládá také informaci o názvu databáze a kolekce, ve které se referovaný dokument nachází. Je tedy vhodný pro situace, kdy je nutné uchovávat reference mezi více kolekcemi. Podporován je většinou jazyků, např. PHP, Java a C#. [9]

```
Dilo
{
  _id: ObjectId("id1"),
  nazevCZ: "…",
  nazevEng: "…",
  komentar: {
              "$ref": "Komentar",
              "$id": ObjectId("id2"),
              "$db": "database"
            }
}
```

Druhým způsobem uchovávání vztahů mezi dokumenty je metoda zapouzdření. Zatímco reference jsou považovány za normalizovaný datový model, zapouzdření slučuje relevantní dokumenty do jednoho a tím model denormalizují. V mongoDB je možné uchovávat velmi rozsáhlé subdokumenty či pole v dokumentech a vytvářet tak kompletní pohled na daný záznam. Výsledkem je také fakt, že pro manipulaci s relevantními daty stačí jedna operace bez nutnosti čtení dat z jiných kolekcí a jejich propojování, což má za následek ve většině případů nárůst výkonu při práci s daty.



Obecně se dá říci, že normalizovaný model se více hodí pro situace, kdy je zapotřebí vyjádřit komplexnější M:N relace. Také je vhodný tam, kde by zapouzdření nedokázalo přinést dostatečně velký nárůst výkonu, aby to vyvážilo fakt, že při zapouzdřování se uchovávají duplicitní data a narůstá tak nutný prostor na disku pro uchování databáze. Denormalizovaný model je na druhou stranu vhodný pro situace, kdy se interpretují 1:1 nebo 1:N relace a kdy je výkon databáze důležitější než její velikost. Proto ve většině případů se využívá právě tento přístup.

Poněvadž zapouzdření v případě katalogu filmů a seriálů nepovede k nadměrné duplicitě dat, denormalizovaný model se zdá být vhodnějším způsobem. Navíc, pokud by vyjádření vztahů mezi dokumenty bylo provedeno pomocí referencí, propojení dokumentů by muselo být ponecháno na aplikační vrstvě, mající za následek poměrně komplikované a pomalejší dotazování.

4.1.1.1 SQL schéma

Při návrhu relační databáze bylo nutné zvážit, zda je vhodné se řídit normalizací a snažit se o dodržení třetí normální formy, která se v praxi často využívá a diktuje, aby neklíčové atributy nebyly mezi sebou závislé nebo upustit od normalizace a data ukládat v podobě, která by omezila nutný počet hledání a joinů na minimum, ve snaze navýšit výkon na úkor redundance a nekonzistence dat.

Poněvadž ČSFD nemá uveřejněnou strukturu své databáze, návrh relačního modelu závisel na osobním posouzení. Vzhledem k tomu, že k relačním databázím neodmyslitelně patří i normalizace, byl zvolen právě tento přístup, zajišťující ochranu před duplicitou a nekonzistencí dat. Zároveň také poskytuje zajímavý pohled na porovnání modelu a výsledků normalizovaného relačního a denormalizovaného nerelačního modelu.

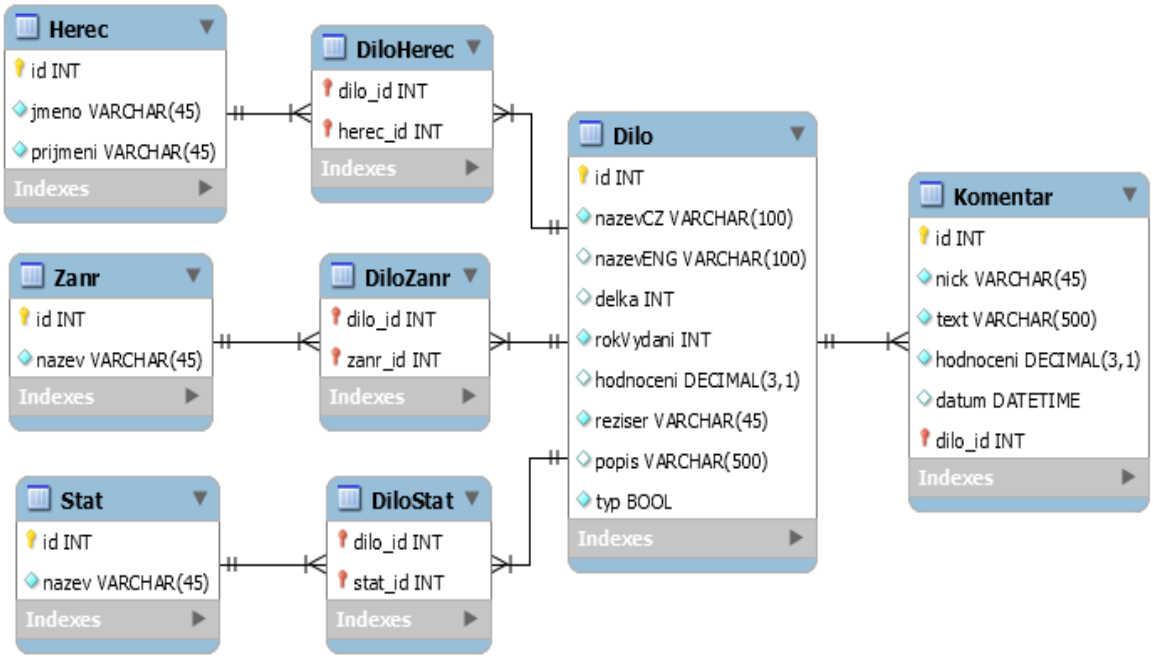


Schéma 1 SQL schéma katalogu

Na schématu číslo 1 je vidět, že celý model se rozložil na celkem 8 tabulek, z nichž 3 jsou takzvané asociační, neboli pomocné, obsahující pouze dva cizí klíče. Hlavními tabulkami jsou zde `Dílo`, `Komentář`, `Herec`, `Žánr` a `Stát`. Tabulka `Dílo` obsahuje základní informace, které lze nalézt u každého filmu či seriálu na ČSFD, mezi něž patří název v českém jazyce i v angličtině, délka trvání v minutách, rok vydání, celkové hodnocení, určující úspěšnost filmu, jméno režiséra, stručný popis díla a identifikátor na rozlišení, zda je dílo filmem či seriálem.

Ke každému z děl mohou uživatelé přidat komentář. Kromě samotného textu komentáře, se u každého z nich uchovává i přezdívka autora, který jej udělil, čas odeslání komentáře a také samotné hodnocení díla od uživatele. Tato jednotlivá hodnocení se průměrují a vzniká tak celkové hodnocení, které se uchovává samostatně v tabulce `Dílo`. Poněvadž každý komentář náleží jednomu určitému dílu, není třeba využívat asociačních tabulek, nýbrž postačí uchovávat v tabulce `Komentář` informaci, k jakému dílu záznam náleží.

Posledními tabulkami jsou `Herec`, `Žánr` a `Stát`, kde ke každé z nich náleží asociativní tabulka, propojující ji s tabulkou `Dílo`. Je to z toho důvodu, že by zde jinak byla vícenásobná vazba, jelikož v každém díle vystupuje více herců, dílo spadá do více žánrů a může být točeno ve více zemích a naopak každý herec, žánr i stát může figurovat ve více dílech.

4.1.1.2 MongoDB schéma

Při návrhu mongoDB schématu se oproti relačním databázím v naprosté většině případů pracuje s denormalizovaným modelem. Vše se tedy upravuje a připravuje tak, aby informace byly co nejdříve vráceny. Při návštěvě stránek ČSFD lze zjistit, že pokud je vybráno určité dílo, na jedné stránce se k němu zobrazí všechny dostupné informace najednou. Tomu je nutné přizpůsobit model – měl by uchovávat všechny zobrazované informace na jednom místě. Toho lze dosáhnout právě metodou zapouzdření, kdy model bude denormalizovaný. Výsledné schéma využívá tedy pouze jedné kolekce, ve které každý dokument reprezentuje kompletní informace o určitém díle.

```
Dilo
{
_id: ObjectId('...'),
nazevCZ: "Vykoupení z věznice Shawshank",
nazevEng: "The Shawshank Redemption",
delka: 142,
rokVydani: 1994,
hodnoceni: 95,
reziser: "Frank Darabont",
popis: "Strach dělá z lidí vězně…",
typ: "Film",
stat: ["USA"],
zanr: ["Drama","Krimi"],
herci: ["Tim Robbins","Morgan Freeman","Bob Gunton"],
komentare: [
{
      nick: "novoten",
      text: "Chlapácký doják…",
      hodnoceni: 100,
      datum: ISODate("2006-05-27…")
},
{
      nick: "gemini",
      text: "Výborné zpracování výborného románu…",
      hodnoceni: 80,
      datum: ISODate("2005-12-09…")
}
]
}
```

Stejně jako v případě SQL schématu, každé dílo obsahuje informaci o českém i anglickém názvu titulu, délce trvání, roku vydání, celkovém hodnocení, režisérovi a položky pro rozlišení, zda jde o film či seriál. Dále se již využívá zapouzdření a v polích se uchovávají informace o státech, kde se dílo natáčelo, seznamu žánrů, do kterých spadá a jmen herců, kteří se na natáčení podíleli. Poslední zapouzdření představuje pole dokumentů, kde každý dokument reprezentuje jeden z komentářů. Zde je uchována informace o přezdívce uživatele, textu jeho příspěvku, ohodnocení díla a data, kdy byl komentář udělen.

4.1.2 Výhody a nevýhody konverze do mongoDB

Každý přechod na jiný databázový model přináší určité změny. Tyto změny mohou být pozitivní a zamýšlené, které navýší výkon, pohodlí a efektivitu práce, ale mohou přinášet i určité nepříjemnosti, které se před případným přechodem na jiný model musí zvážit. V této části je vystiženo, jaké hlavní výhody by konverze přinesla a s jakými nevýhodami by se muselo počítat.

4.1.2.1 Výhody konverze

- Sharding a load balancing

Pojmem sharding se označuje metoda, která umožňuje rozdělení databáze na více částí a provozování jednotlivých částí na samostatných serverech. Tímto způsobem lze navýšit výkon databáze, jelikož jednotlivé servery uchovávají pouze část databáze a jsou tedy schopné rychleji zpracovat příkaz. Problematice shardingu a jeho konfiguraci je věnována kapitola 4.3. V případě uchovávání katalogu filmů a seriálů lze implementaci shardingu ovšem považovat za nadbytečnou, protože kolekce bude obsahovat omezený počet záznamů. Pokud by však kolekce sloužila například na uchovávání rychle rostoucích a rozsáhlých záznamů komunikace na sociálních sítích, byla by implementace shardingu téměř bezpodmínečná.

Load balancing slouží, jak již název napovídá, na rozdělení zátěže mezi více serverů. Aby byla zátěž rovnoměrná a některé servery nebyly nadměrně zatížené, load balancing se snaží o přerozdělení částí databáze (shardů) tak, aby bylo dosaženo optimálního vytížení. Tato funkce je v mongoDB povolená automaticky a nabízí pokročilejší správu než v případě mySQL. [10]

- Rychlost

Jako další výhoda oproti mySQL se dá považovat rychlost, se kterou mongoDB zpracovává příkazy. Potřebný čas je obecně mnohem nižší a to především díky zapouzdření a z toho vyplívajícího faktu, že všechny potřebné informace jsou na jednom místě, dostupné okamžitě po nalezení konkrétního dokumentu. U menších kolekcí, jako je katalog filmů a seriálů, nemusí být výsledná úspora času razantní, avšak u rozsáhlejších kolekcí se potřebné časy pro zpracování příkazu mohou velmi lišit. Pokud by ovšem místo zapouzdření byly vztahy mezi záznamy vyjádřeny pomocí referencí, mySQL by v některých případech mohlo mít v rychlosti navrch, především díky jazyku SQL.

- Flexibilita

Výhodou přechodu na mongoDB je i zisk flexibility. Poněvadž mongoDB nemá předem definované schéma dat, lze snadno již existující data modifikovat co se počtu a typu atributů týče. Této výhody se nejvíce využije v situacích, kdy jednotlivé dokumenty potřebují určitý stupeň svobody dat. V případě uchovávání katalogu filmů a seriálů by flexibilita přišla vhod,

pokud by např. film mohl mít i neoficiální název. Ten se u některých titulů skutečně může objevit. V mongoDB by pak stačilo pouze přidat novou položku u daného dokumentu, zatímco v mySQL by přidání vyžadovalo mnohem rozsáhlejší úpravy.

4.1.2.2 Nevýhody konverze

- Chybějící podpora SQL

Řada databázových administrátorů se k nerelačním databázím dostanou právě přes relační, kde si zvykli na pohodlný jazyk SQL a jeho bohaté funkce. MongoDB tento jazyk ovšem nepodporuje a funkce řeší jiným způsobem. To klade zvýšené nároky na samotné správce, kteří by se v případě přechodu na jiný systém museli zaškolit. Chybějící podpora SQL ovšem v katalogu děl není velkou překážkou, poněvadž se vztahy mezi dokumenty dají vyjádřit pomocí zapouzdření a tím odpadne nutnost propojení záznamů pomocí operace join, která je součástí jazyka SQL.

- Redundance dat

Nevýhodou denormalizovaného modelu je fakt, že v něm vznikají duplicitní informace. To má za následek růst databáze, která se v konečném výsledku nemusí vejít do operační paměti, čímž se výkon při zpracování příkazů razantně sníží, protože data budou získávána z pevného disku. Katalog děl ovšem příliš duplicitních dat neobsahuje a navíc je tento fakt kompenzován tím, že díky denormalizaci lze získat vyšší výkon.

4.2 Replikace

Replikací se obecně v rámci databází myslí proces, který má za úkol synchronizaci dat mezi více servery. S jeho pomocí můžeme navýšit dostupnost databáze prostřednictvím vytváření kopií dat. Pokud se tyto kopie uchovávají na různých databázových serverech, poskytuje replikace i nepřerušený provoz v případě ztráty komunikace s jedním ze serverů.

Prostřednictvím replikace je také možné dosáhnout lepších výsledků při čtení, protože je možné rozložit zátěž na více serverů, které obsahují stejná data. Uchovávání kopií se také vyplatí pro další činnosti, jako jsou obnovy po havárii, reportování a zálohování.

V prostředí mongoDB je replikace zprostředkována prostřednictvým tzv. replica setu. Jde o skupinu až 50 `mongod` procesů, které uchovávají stejný dataset. V replica setu jsou dva druhy těchto procesů, které uchovávají data – primární a sekundární. Kromě nich se zde může ještě vyskytnout tzv. Arbitr, který je však nepovinný.

Primární uzel (proces) zpracovává všechny zápisy do databáze a v replica setu může být pouze jeden. Veškeré změny na datech, které byly provedeny na primárním uzlu, se zaznamenávají do speciální kolekce, nazvané Oplog. Její velikost je pevně daná a v případě dosažení její maximální velikosti se začnou přepisovat nejstarší záznamy.

Sekundární uzly čtou Oplog, který je vytvořen primárním uzlem, a aplikují na svá data operace, které mají za úkol reflektovat změny provedené na primárním uzlu. Pokud dojde ke ztrátě komunikace s primárním uzlem, mezi sekundárními uzly dojde k volbě o tom, který z nich se stane primárním a zastane tak zpracovávání žádostí o zápis dat a správu Oplog kolekce. Na schématu č. 2 je vidět, že primární uzel přijímá jak žádosti o zápis, tak i o čtení. Z něj se data replikují na sekundární uzly. Mezi sekundárními a primárním uzlem se v průběhu zjišťuje pomocí tzv. Heartbeat, zda je server aktivní a zda není zapotřebí zvolit nový primární uzel.

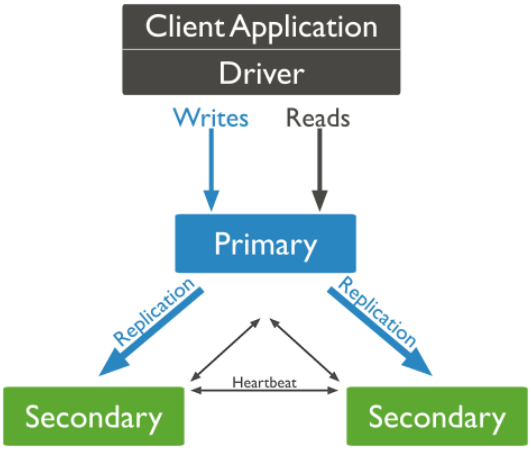


Schéma 2 Replikace v mongoDB [11]

Arbitr je stejně jako primární a sekundární uzel `mongod` proces, který ale na rozdíl od nich neuchovává žádná data z databáze. Jeho úkolem je napomoci při volbě nového primárního uzlu. Přidává se do replica setu v případě, že je v něm sudý počet uzlů, aby se předešlo případnému nerozhodnému výsledků při volbě nového primárního uzlu.

Sekundární uzly při aplikování změn v Oplog kolekci postupují asynchronně. To znamená, že změny jsou provedeny až poté, co je provedl i primární uzel. To může mít za následek, že pokud je povolené čtení i ze sekundárních uzlů, výsledek na dotaz nemusí reflektovat aktuální data v primárním uzlu. Proto je také ve výchozím nastavení povoleno čtení pouze z primárního uzlu.

4.2.1 Konfigurace replikace

V následujících krocích bude ukázáno, jak vytvořit replica set se třemi uzly. Pro produkční prostředí je to ve většině případů dostatečný počet, který zabezpečí potřebnou kapacitu pro distribuované read operace. Jak již bylo zmíněno výše, replica set by měla mít lichý počet uzlů, aby se předešlo problémům při volbě primárního uzlu. Pro testovací účely můžou být všechny `mongod` instance spuštěny na jednom serveru, pro produkční prostředí se ovšem doporučuje mít uzly odděleně na vlastních serverech.

Každý ze serverů, který bude hostit uzel, musí mít nainstalované mongoDB. Síť musí být také nakonfigurována tak, že každý z členů v replica setu může mezi sebou komunikovat. Samotná konfigurace replica setu je poměrně jednoduchá a dá se shrnout do následujících kroků:

1. Vytvoření adresářů, kam se budou ukládat data. Pomocí příkazu `mkdir -p D:\mongoDB\data\rs0-0 D:\mongoDB\data\rs0-1 D:\mongoDB\data\rs0-2` lze vytvořit tři adresáře, do kterých budou uzly ukládat svá data.
2. Spuštění všech třech `mongod` instancí. Pro jejich odlišení je nutné každé z nich přiřadit jiný port. Zároveň je nutné jim také přiřadit odlišný adresář, do kterého budou ukládat svá data. První z instancí bude spuštěna pomocí příkazu:

   ```
   mongod --port 27017 --dbpath D:\mongoDB\data\rs0-0 --replSet rs0 --smallfiles --oplogSize 128
   ```

   Zbývající dvě instance `mongod` budou mít libovolný jiný port, který se nevyužívá (např. 27018 a 27019) a odlišný adresář (rs0-1 a rs0-2). Zbytek příkazu již zůstane beze změny.

   `--replSet rs0` značí, že instance bude součástí replica setu s názvem rs0.

   `--smallfiles` říká, že výchozí velikost souborů bude menší a maximální může být až 512MB.

   `--oplogSize 128` určuje maximální velikost Oplog kolekce. Standardně je určený jako 5% z volného místa na pevném disku. Tímto příkazem mu nastavíme maximálně 128MB.
3. Připojení se k jedné z `mongod` instancí pomocí shellu. Aby bylo jasné, k jaké z třech aktuálně běžících instancí se má shell připojit, je nutné ještě připojit číslo portu jako parametr. Zápis pro připojení se k první z instancí by vypadal takto: `mongo.exe --port 27017`.
4. V mongo shellu spustíme replica set pomocí příkazu `rs.initiate()`. Pokud metodě není předán žádný parametr, je provedeno defaultní nastavení.
5. Ve stejném shellu přidáme do replica setu zbývající dvě `mongod` instance. Toho dosáhneme pomocí příkazů:

   ```
   rs.add("<hostname>:27018")
   rs.add("<hostname>:27019")
   ```

   Místo `<hostname>` zde bude konkrétní jméno počítače, následované číslem portu. V tomto bodě je již replica set plně funkční. Jeho stav lze zkontrolovat pomocí příkazu `rs.status()`.

4.3 Sharding

Sharding je způsob, jakým se databáze vypořádávají s rozsáhlými datasety a velkým množstvím dotazů. Tyto problémy mohou vést k tomu, že všechna data se nevejdou do paměti RAM a je tedy nutné je získávat z pomalejšího harddisku a velké množství dotazů může způsobit, že procesor bude plně vytížen. Jak již bylo zmíněno v kapitole 3.3 Přechod na NoSQL řešení, u databází je problém řešen dvěma způsoby – vertikálním a horizontálním škálováním.

Vertikální škálování přistupuje k problému přidáváním výkonnějších CPU a více RAM, zatímco u horizontálního škálování, kterému se také říká sharding, se problém řeší rozdělením dat a jejich distribuováním na více serveru, neboli shardů. Každý ze shardů je pak nezávislou databází, schopnou samostatného fungování a dohromady tvoří kompletní logickou databázi. Na schématu č. 3 je zjednodušeně vidět, jak takové shardy vznikají. Uvažujme, že máme kolekci, která obsahuje 1 TB dat. Pro jeden server by takový objem dat byl příliš velkým. Proto se celá kolekce rozdělí na části a rozloží na více serverů.

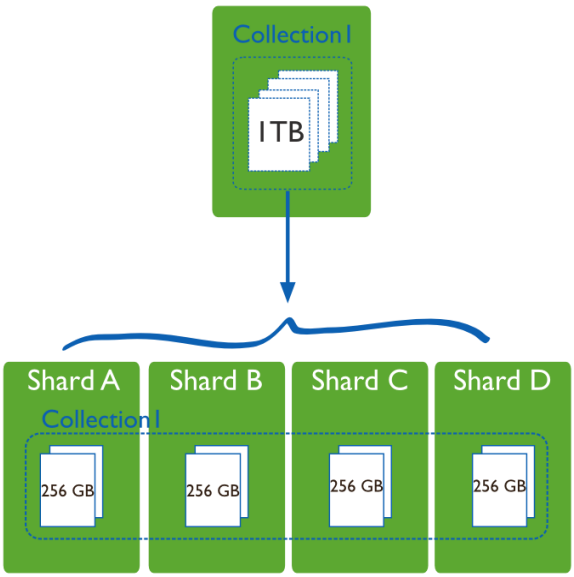


Schéma 3 Rozdělení kolekce na shardy [12]

Pomocí shardingu lze tedy snížit počet operací, které musí jednotlivý server vykonávat. S přibývajícím počtem shardů v clusteru se počet operací na shard ještě dále snižuje, zatímco se celková kapacita a výkon zvyšují. Pokud je dotazovaný dokument například uložený na shardu B, tak se přistoupí pouze k danému shardu.

V mongoDB jsou tři základní prvky, které jsou využívané při shardingu:

- Shardy
- Query routery
- Config servery

Shardy jsou části kolekce, které obsahují data a jsou schopné samostatného fungování. V produkčním prostředí je každý shard zároveň i replica set (viz. Kapitola 4.2 Replikace). Pro testovací potřeby ale může být každý shard pouze jedna `mongod` instance.

Query routery, neboli také instance `mongos`, se starají o komunikaci mezi uživatelskou aplikací a konkrétním shardem, na kterém jsou uloženy data, se kterými chce aplikace pracovat. Princip fungování spočívá v tom, že klientská aplikace pošle žádost query routeru, který následně směruje požadovanou operaci na shard, obsahující daná data a poté vrátí klientovi zpět výsledek. V produkčním prostředí se často nachází více než jeden query router, aby bylo možné rozložit žádosti od více uživatelů.

Config servery ukládají metadata clusteru. To znamená, že jsou zde uložené informace o tom, jaká data jsou na shardech. Query router těchto informací využívá, aby zjistil, kam má cílit uživatelský požadavek.

Na schématu č. 4 jsou zobrazeny všechny komponenty shardingu a jejich vzájemná interakce. V produkčních prostředích bývá často více shardů a query routerů, zatímco pro testovací potřeby bývá model jednodušší, obsahující pouze jeden query router a shardy bez replikace.

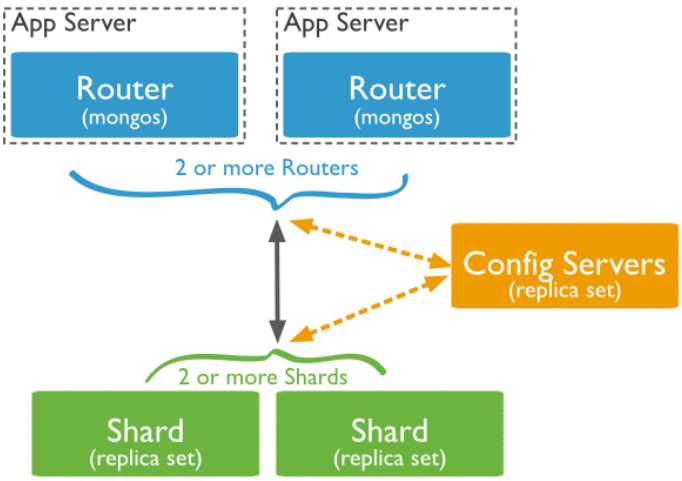


Schéma 4 Sharding v mongoDB [12]

4.3.1 Konfigurace shardingu

Jedním ze základních předpokladů při konfiguraci shardingu je, aby všichni členi v clusteru mezi sebou mohli komunikovat. Proto je nutné povolit porty, které se budou pro komunikaci využívat a dále zajistit, že síť bude správně nakonfigurována. Samotné nastavení shardingu se dá shrnout do následujících kroků:

1. Vytvoření Config serverů a jejich spuštění

Poněvadž Config servery jsou řešené pomocí replica setu, jejich nastavení bude podobné, jako v kapitole 4.2.1 Konfigurace replikace. Pro tento příklad budou využity tři Config servery, které uchovávají metadata clusteru. Jejich počet by měl být opět lichý, tentokrát ovšem bez možnosti nasadit arbitra, kteří nejsou u config serverů povoleni.

V prvním kroku se opět vytvoří adresáře pro config servery. Dále se spustí všechny tři instance `mongod`, které budou nastavené jako config servery a přiřazeny do replica setu.

Spuštění a nastavení první instance se provede následujícím příkazem. Příkaz předpokládá existenci nově vytvořených adresářů csvr0-0 až csvr0-2:

```
mongod --configsvr --replSet configReplSet --port 27020 --dbpath
D:\mongoDB\data\csvr0-0
```

Prvním přepínačem říkáme instanci `mongod`, že se bude jednat o config server. Druhým již sdělujeme, jak se bude jmenovat replica set, jehož součástí instance bude. U definice portu je důležité, aby port nebyl dosud využíván a každý z config serverů měl port odlišný. Zbývající dva config servery tedy budou využívat porty 27021 a 27022. Každému z config serverů bude i posledním přepínačem přiřazen jiný adresář.

Poté, co jsou již všechny tři instance spuštěny, připojíme se k libovolnému z config serverů pomocí shellu. Stejně jako při konfiguraci replikace spustíme příkaz `rs.initiate()` a následně přidáme dvě zbývající instance do replica setu pomocí `rs.add("<hostname>:27021")` a `rs.add("<hostname>:27022")`.

2. Spuštění Query routerů

Query routery jsou instance `mongos`, které poněvadž slouží pouze jako interface mezi klientem a databází, tak nevyžadují žádné datové úložiště. V produkčním prostředí by měly být spuštěny minimálně dvě instance, pro potřeby testování však stačí jen jedna. Samotné spuštění se provede pomocí příkazu:

```
mongos --configdb configReplSet/<hostname:27020>, <hostname:27021>,
<hostname:27022>
```

Je zde pouze jeden potřebný přepínač, který jako parametr bere název replica setu, vytvořeného v prvním kroku, následované lomítkem a seznamem kombinací hostname a čísla portu config serverů, oddělených čárkou. Ve výchozím nastavení využívá `mongos` port 27017, pokud tedy již na tomto portu naslouchá jiná instance, je třeba změnit port pomocí přepínače `--port`.

3. Přidání shardů do clusteru a povolení shardování databáze

Shardem může být jak samostatně běžící `mongod` instance nebo replica set. V produkčním prostředí by mělo jít vždy o replica set, zatímco pro testování stačí `mongod` instance. Přidávání shardů probíhá přes query router, je tedy nutné se k němu nejdříve připojit pomocí shellu. Zápis vypadá následovně:

```
mongo --host <hostname> --port 27017
```

Jsou zde dva přepínače – prvním se určuje název stroje, na kterém běží `mongos` instance a druhým se definuje port, na kterém instance naslouchá. Pokud byl v předešlém kroku port změněn, připojení proběhne pomocí nového portu.

Následně se v shellu přidá replica set, vytvořený při konfiguraci replikace, do clusteru jako shard. Každý člen replica setu se musí přidat samostatně a to pomocí příkazu:

```
sh.addShard("rs0/<hostname>:<port>")
```

Pokud se tedy replica set skládá ze tří `mongod` instancí, příkaz bude proveden celkem třikrát. V první části parametru se definuje název replica setu, následované lomítkem a názvem stroje a číslem portu, na kterém `mongod` běží.

Když už je replica set přidán do clusteru, dalším krokem je povolení shardování databáze. Tento krok ještě nezpůsobí, že data budou rozdělena, pouze umožní kolekci v dané databázi, aby mohla využívat sharding. Zatímco je shell stále připojen k `mongos`, zadáme příkaz:

```
sh.enableSharding("<database>")
```

Ten umožní shardování všem kolekcím v databázi, jejíž jméno se předá jako parametr uvedené metodě.

4. Rozdělení kolekce

Posledním krokem je rozdělení kolekce. Před zahájením je nutné si definovat tzv. shard klíč, který bude součástí všech dokumentů. Jeho správnou volbou lze výrazně ovlivnit rychlost, jakou databáze dokáže zpracovat dotaz. Podle tohoto klíče budou data rozdělena mezi shardy. Proces rozdělování spočívá v tom, že dokumenty, které mají klíče s podobnou nebo stejnou hodnotou, budou umístěné na stejném shardu. Volba správného klíče úzce souvisí se schématem dat a způsobem, jakým se aplikace dotazuje a zapisuje data. Jako klíč by tedy měla být nejlépe položka, případně kombinace položek, které se často v dotazech vyskytují a jejichž hodnoty jsou zároveň mezi sebou odlišné, aby se daly vhodně rozmístit mezi shardy.

V případě katalogu filmů a seriálů se jako nejlepší kandidát na shard klíč jeví samotný název díla, protože jde o relativně unikátní položku v každém dokumentu a zároveň se bude často používat jako součást dotazu do databáze. Rozdělení kolekce podle daného klíče se provede následovně:

```
sh.shardCollection("database.dilo", {"nazevCZ": 1})
```

Metoda přijímá dva parametry – informaci o jménu databáze a názvu kolekce, které jsou oddělené tečkou a druhým parametrem je dokument, který obsahuje seznam shard klíčů, podle kterých se budou dokumenty dělit mezi shardy. Hodnota jedna zde znamená, že budou klíče seřazeny vzestupně.

4.4 Indexace

Hlavním smyslem indexace je učinit dotazy a operace nad databází efektivnější. Toho je dosaženo především důkladnou analýzou způsobů, jakými se s daty pracuje a následným vhodným rozmístěním indexů na položky, které jsou při dotazování nejčastěji využity. Pokud takový index neexistuje, kolekce musí být prohledána celá, aby bylo možné vybrat ty dokumenty, které odpovídají dotazu. Pokud ovšem patřičný index existuje, mongoDB ho může využít a omezit tak počet dokumentů, které musí být prohledány.

Pro indexy se v mongoDB využívá datová struktura B stromů, která uchovává hodnoty definovaných položek, případně kombinace položek, seřazené dle hodnoty položky. Existuje zde několik rozdílných typů indexů, mezi které patří výchozí index na _id, index na jedné položce, složený index, víceklíčový index, geoprostorový index, textový index a hash index. [13]

Všechny kolekce v mongoDB mají index na položce _id. Jeho hodnota musí být jedinečná, protože slouží jako identifikátor jednotlivých dokumentů a zároveň tak brání vložení dvou dokumentů se stejnou hodnotou pro _id. Kromě této výchozí indexované položky je možné indexovat i položku jinou, uživatelsky definovanou. Pokud se ovšem dotazování týká více položek najednou, je možné vytvořit složený index, který zahrne všechny potřebné položky. Víceklíčový index slouží pro indexaci obsahu polí. Pokud je tedy indexována položka, která obsahuje pole hodnot, pro každou z těchto hodnot je vytvořený index. Pro potřeby aplikací zabývajících se mapami a lokacemi, je možné indexovat i jednotlivé koordináty. Indexovat lze i položky, které obsahují index typu text. To umožňuje rychlé prohledávání textových řetězců v kolekcích. Posledním typem indexu je hash. Ten umožňuje indexovat položku, jejíž hodnotou je hash, sloužící pro potřeby shardingu.

V případě dotazování na databázi filmů a seriálů se nejčastěji vyhledávají díla pomocí samotného názvu či jeho části. Je ale také možné filtrovat seznam děl pomocí typu, který říká, zda jde o film či seriál, dále pomocí žánru, roku vydání nebo podle hodnocení. U takto filtrovaných děl se pak ve výsledném seznamu zobrazí název díla a jeho celkové hodnocení.

Pro vyhledávání díla pomocí názvu se využije textového indexu. Poněvadž dílo má jak český, tak i anglický název, mělo by být možné ho vyhledat oběma způsoby. Ideálně by také nemělo záležet na diakritice ani velikosti písmen při vyhledávání a dílo by mělo být možné

vyhledávat i při zadání pouze částečného názvu. V mongoDB lze toto zadání splnit pomocí příkazu:

```
db.dilo.createIndex(
   {
     nazevCZ: "text",
     nazevEng: "text"
   }
 )
```

Metoda `createIndex()` přijímá v tomto případě jeden parametr, kterým je dokument, určující, jaké položky se mají indexovat. Položkám je přiřazena hodnota text, která umožní provádět vyhledávání v řetězci hodnot.

Podobného fungování lze dosáhnout i v mySQL. Pro potřeby prohledávání textových řetězců zde existuje index typu fulltext. Ten je podporovaný v myISAM a innoDB a to pro datové typy `char`, `varchar` a `text`. Jeho zápis je následovný:

```
CREATE FULLTEXT INDEX nazev
ON Dilo(nazevCZ,nazevEng)
```

Opět je vytvořený nový index, který se vztahuje na název v češtině i angličtině, pro rychlejší vyhledávání záznamů. Ten by měl být vytvořený ideálně až poté, co jsou všechny záznamy v tabulce. Pokud by index byl vytvořený předem, data by se do databáze vkládala pomaleji.

Další index bude vytvořený na položce, která značí, zda je dílo filmem či seriálem. To se používá při filtrování vyhledávání. V mongoDB se pro tuto položku využije stejná metoda jako v předešlém případě, ovšem s odlišným parametrem:

```
db.dilo.createIndex(
   {
     typ: 1
   }
 )
```

Parametrem je zde dokument s položkou `typ`, které je přiřazená hodnota `1`. To znamená, že záznamy budou ve vzestupném pořadí. V mySQL lze stejného výsledku dosáhnout pomocí příkazu:

```
CREATE INDEX typ
ON Dilo (typ)
```

Velmi podobným způsobem bude učiněno i indexování roku, kdy bylo dílo natočeno. V mongoDB se indexace provede následovně:

```
db.dilo.createIndex(
   {
     rokVydani: 1
   }
 )
```

Parametrem je zde opět dokument, ve kterém je v tomto případě položka, uchovávající hodnotu s rokem vydání ve vzestupném pořadí. V mySQL se indexace provede obdobně, jako v případě indexace typu díla:

```
CREATE INDEX rokVydani
ON Dilo (rokVydani)
```

Další položkou na indexaci je žánr, podle kterého lze taky díla filtrovat. V mongoDB půjde o víceklíčový index, protože žánr je uchováván v poli. Každá položka v poli bude tak indexována samostatně. Konstrukce tohoto indexu je totožná s předchozími případy:

```
db.dilo.createIndex(
   {
     zanry: 1
   }
 )
```

V mySQL je situace odlišná, protože žánry nejsou uchovávány v tabulce `Dílo`, nýbrž jsou v asociační tabulce, která spojuje `Dílo` a `Žánr`. Tato asociační tabulka má své položky indexované a proto není nutné pro žánr vytvářet nové indexy.

Posledním způsobem vyhledávání je podle hodnocení. To se zobrazuje spolu s názvy děl a je výhodnější je spojit do složeného indexu. V mongoDB by pak takový zápis vypadal následovně:

```
db.dilo.createIndex(
   {
     hodnoceni: -1,
     nazevCZ: 1
   }
 )
```

Tento zápis způsobí, že budou spolu uchovávány hodnoty položek hodnocení a názvu v češtině s tím, odkazy budou nejdříve seřazeny dle nejvyššího hodnocení a pak až dle názvu. V mySQL by vytvoření složeného indexu proběhlo následovně:

```
CREATE INDEX hodnoceni ON Dilo (hodnoceni, nazevCZ)
```

Aplikováním všech výše zmíněných indexací na často používaných atributech, lze dosáhnout vyššího výkonu při běžném dotazování databáze. To by mělo být provedeno až po naplnění databáze, aby nedošlo ke zpomalení importu dat. S rostoucím počtem indexů ovšem roste i potřebné místo v paměti RAM, proto je nutné sledovat, zda volné místo nedochází a zda se pro uchovávání indexů nevyužívá pomalého pevného disku.

4.5 Testování

Tato kapitola se soustředí na testování mongoDB a mySQL databází. Důraz je kladen na operace, se kterými by se databáze setkávala nejčastěji, pokud by byla nasazena do produkčního prostředí. Jde tedy především o operace čtení, které na základě určitých podmínek budou hledat odpovídající díla. Do testování jsou zahrnuty i ostatní operace, jako je zápis, modifikace a mazání, jelikož i s nimi by se databáze potýkala, byť v mnohem menší míře. Předtím, než ale vůbec bude možné testy provést, je nutné vytvořit dataset, který bude sloužit jako zdroj dat pro databázi. Poněvadž jde o poměrně rozsáhlý objem dat, nebudou reprezentovat skutečná díla, nýbrž půjde o náhodně vygenerované údaje.

4.5.1 Testovací prostředí

Aby výsledky testů byly věrohodné, je nutné je provádět na stejném stroji, při pokud možno stejných podmínkách. Pro tyto účely byl zvolen osobní počítač. Nejdůležitějšími prvky jsou zde operační paměť a pevný disk. Ty ve většině případů rozhodují o rychlosti prováděných příkazů. Všechny testy jsou prováděné samostatně, oddělené od sebe restartem počítače. Veškeré aplikace, které by mohly ovlivnit výsledky testů, byly předem vypnuty. V tabulce číslo 3 je uvedena konfigurace počítače, spolu s potřebným softwarovým vybavením. Pro testování byly použity nejnovější verze mySQL a mongoDB serveru k aktuálnímu datu.

Operační systém	Windows 7 Professional SP1 64b
Procesor	Intel i7-4790k 4.00 GHz
Operační paměť – celková	8 GB RAM
Operační paměť – k dispozici	5.7 GB RAM
Pevný disk	SATA II 160 GB 7200RPM
SQL server	MySQL Community Server 5.7.11 s InnoDB
MongoDB server	MongoDB 3.2.3

Tabulka 3 Testovací prostředí

4.5.2 Vytvoření datasetu

Za dataset se považuje obsah tabulky, či v případě mongoDB, obsah kolekce. Poněvadž každý ze systémů vyžaduje obsah v jiné podobě, je nutné vytvořit obsahy dva – jeden pro mySQL a druhý pro mongoDB. MySQL jako zdroj dat pro import může využívat soubor CSV, kde jeden takový soubor bude reprezentovat data pro jednu tabulku. Poněvadž tabulek v databázi je celkově osm, bude zde i osm CSV souborů. MongoDB jako zdroj dat pro import zase využívá soubory JSON. Ty uchovávají data ve formě dokumentů.

Pro vytvoření obou datasetů je využit programovací jazyk java. V něm se generují potřebná data a ukládají se do CSV a JSON souborů. Tyto soubory jsou pak předloženy databázím jako zdroj dat. Do každé z nich bude ze souborů importováno 10 000 000 děl a 100 000 000 komentářů, které budou mezi díla náhodně rozmístěny.

4.5.2.1 Dataset pro mongoDB

Pro tvorbu JSON dokumentu lze v javě využít knihovnu `json-simple`, která je zdarma ke stažení na https://code.google.com/archive/p/json-simple/downloads. Ta po připojení k projektu umožňuje zjednodušené vytváření struktury dokumentů pomocí speciálních metod. Ty jsou využity v cyklu, který je proveden 10 000 000x. V každém z těchto cyklů je vygenerován nový dokument s náhodnými hodnotami. Každý z nich ovšem má jako položky jedinečný identifikátor _id, název v českém i anglickém jazyce, délku, rok vydání, hodnocení, režiséra, popis, typ a pole pro státy, žánry, herce a komentáře. Takto vygenerovaný dokument je uložený do JSON souboru a následně cyklus běží znovu. Celý zdrojový kód pro vytvoření datasetu je v sekci Přílohy.

4.5.2.2 Dataset pro mySQL

Dataset pro mySQL je, stejně jako v případě mongoDB, vytvořen v jazyce java. Poněvadž se data ukládají do formátu CSV, není třeba žádné dodatečné knihovny, která by pomáhala s formátováním uložených informací. Každá hodnota atributu je oddělená středníkem a každý záznam je na nové řádce. Jelikož by soubor, který obsahuje data pro komentáře, byl velmi rozsáhlý (100 000 000 záznamů), je dobrým zvykem ho rozdělit na menší části, za účelem rychlejšího zpracování. Proto je dataset uchovávající informace o komentářích rozdělený na deset částí, kde každá z nich v sobě nese informace o 10 000 000 komentářů. Zdrojový kód pro vytvoření datasetu je také v sekci Přílohy.

4.5.3 Testování operace zápis

Zápis do databáze je prvním testem, který ověří, jak si oba modely stojí při ukládání informací. Do obou z nich je uložen kompletní dataset, který byl vytvořen v předešlé sekci. Jak mongoDB, tak i mySQL mají své způsoby, jakými lze nahrát obsah vytvořených souborů do databází. MongoDB pro tyto účely využívá samostatnou aplikaci `mongoimport`, která je součástí instalačního balíčku. MySQL na druhou stranu využívá pro import příkaz `LOAD`

`DATA LOCAL INFILE`. Do obou databází je těmito způsoby načteno 10 000 000 děl a 100 000 000 komentářů.

4.5.3.1 Zápis v mongoDB

Do mongoDB v rámci testování zápisu je importován JSON dokument, který obsahuje 10 000 000 děl se zapouzdřenými informacemi o hercích, žánrech, státech a komentářích od uživatelů. Konfigurace importu není pro tento test modifikována, jelikož již ve výchozím nastavení poskytuje dostatečný výkon. V případě importování rozsáhlejšího dokumentu je možné konfiguraci rozšířit o parametr `–numInsertionWorkers`, kterým se stanovuje počet vláken, pracujících na vkládání dat.

Před zahájením importu musí běžet `mongod` server. Po jeho spuštění je možné zahájit import dat s využitím aplikace `mongoimport` následujícím způsobem:

```
mongoimport --collection Dila --file dila.json
```

Jakmile je příkaz potvrzen, automaticky se spustí import. Všechny dokumenty, které jsou v souboru `dila.json`, jsou vloženy do nově vytvořené kolekce `Dila`. Místo aplikace `mongoimport` by se pro vkládání dokumentů dala využít i metoda `insert()`, spolu s programovacím jazykem java. Ta by ale vyžadovala vkládání jednotlivých dokumentů v cyklech, což by nevyhnutelně vedlo k nižšímu výkonu.

Potřebný čas na provedení importu celé kolekce byl 27 minut a 31 sekund. V průběhu importu byla rychlost téměř neměnná, ukazující na bezproblémovost u větších datasetů.

4.5.3.2 Zápis v mySQL

MySQL využívá několik CSV souborů jako zdrojů dat, kde každý z nich reprezentuje jednu tabulku. V případě komentářů je zdrojový soubor rozdělený na deset menších souborů, za účelem navýšení výkonu při vkládání. Tabulka číslo 4 ukazuje, jaký je celkový počet záznamů v jednotlivých tabulkách.

Tabulka	**Počet záznamů**
Dílo	10 000 000
Komentář	100 000 000
Herec	450
Žánr	14
Stát	10

DíloHerec	89 523 061
DíloŽánr	23 239 418
DíloStát	18 941 553

Tabulka 4 Počet záznamů v tabulkách

Při importování většího objemu dat do innoDB je vhodné upravit konfiguraci serveru. S využitím níže uvedených konfiguračních příkazů je možné navýšit rychlost přidávání záznamů, oproti výchozímu nastavení.

```
innodb_buffer_pool_size = 4G
autocommit = 0
unique_checks = 0
foreign_key_checks = 0
```

`innodb_buffer_pool_size` určuje velikost prostoru v paměti RAM, který je využíván pro cachování a indexování. Jeho velikost by měla být velká tak, aby ponechávala dostatek paměti i pro ostatní procesy. Čím je vyhrazený prostor větší, tím více dat se do paměti vejde a tím méně se čtou data z pevného disku.

`Autocommit` říká serveru, zda se každá operace, která jakkoliv upravuje tabulku, má provést okamžitě a zapsat změny na disk. Přiřazená hodnota nula znamená, že se potvrzení změn musí provést manuálně, umožňující tak hromadné načítání dat.

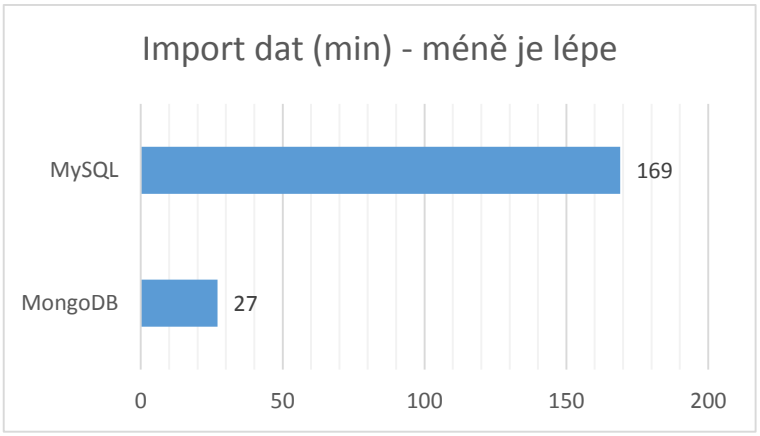
`unique_checks` s hodnotou nula říká serveru, že ve vkládaných datech nejsou žádné duplikátní klíče a že tedy není nutné provádět jejich kontrolu.

`foreign_key_checks` s hodnotou nula znamená, že databáze nebude kontrolovat vazby cizích klíčů.

Pro vkládání dat do mySQL je využit příkaz `LOAD DATA LOCAL INFILE.` Jeho podoba by pro načtení všech děl vypadala následovně:

```
LOAD DATA LOCAL INFILE 'D:\dila.csv' INTO TABLE Dilo FIELDS
TERMINATED BY ';' LINES TERMINATED BY '\n'
```

Celkový čas, potřebný pro naplnění všech osmi tabulek, byl 2 hodiny, 49 minut a 15 sekund. Nejpomalejší bylo načítání dat do tabulky komentářů. S rostoucí tabulkou se i zpomalovala rychlost vkládání záznamů.



Tabulka 5 Import dat

4.5.4 Testování operace čtení

Čtení je pro databázi uchovávající kolekci filmů a seriálů stěžejní záležitostí. Tento typ operací je vykonáván nejčastěji, a proto je jim ve fázi testování věnována největší pozornost. Celkově půjde o dva testy. Prvním je nalezení všech děl, u kterých jejich název obsahuje hledané slovo a vypsání názvů těchto děl. Pro tento test bude využito fulltextového vyhledávání na položky, uchovávající název díla. Druhý test bude mít za úkol vypsat název a hodnocení prvních 50 děl, které mají definovaný žánr. Tento seznam děl bude seřazený dle hodnocení.

4.5.4.1 Čtení v ¨mongoDB

První test, který bude zkoumat výkon databáze při čtení, bude zaměřen na vyhledávání díla podle názvu. Jak již bylo zmíněno dříve, konkrétní dílo může být vyhledáváno jak podle české názvu, tak i anglického a to i s využitím pouze částečného názvu titulu. To je umožněno díky fulltextovému indexu na obou názvech. Vyhledávání není citlivé na diakritiku ani velikost písmen.

Cílem testu je najít díla, která mají v názvu slovo „vězení“. Název může obsahovat i další slova před i za hledaným výrazem. U odpovídajících děl se pak zobrazí jejich český a anglický název. V kolekci `Dila` se vyskytuje celkově osm odpovídajících dokumentů. Jejich vyhledání v mongoDB vypadá následovně:

```
db.Dila.find({$text: {$search: "Vězení"}}, {_id:0, nazevCZ:1.
nazevEng:1})
```

První část příkazu obsahuje podmínku, podle které se vyhledává. `$text` vykonává textové vyhledávání obsahu položky, které je přiřazený textový index. Není nutné specifikovat, o které položky jde, protože textový index může být v kolekci pouze jeden – v tomto případě složený z českého a anglického názvu. `$search` obsahuje samotný hledaný řetězec. Druhá část příkazu obsahuje informace o tom, které položky se mají ve výsledku zobrazit. Potřebný čas pro vykonání tohoto dotazu byl 154ms.

Druhým testem bylo hledání děl podle žánrů. Zde bylo úkolem najít ty díla, které spadají žánrově do kategorie Akční a vypsat český název a hodnocení prvních 50 nejúspěšnějších. Z celkových 10 000 000 děl spadá do tohoto žánru přesně 1 000 000 děl. Pro tuto potřebu je využit index na poli se žánry. Zápis dotazu vypadá následovně:

```
db.Dila.find({zanry:"Akcni"}, {_id:0, hodnoceni:1,
nazevCZ:1}).sort({hodnoceni: -1}).limit(50)
```

V první části příkazu se opět definuje podmínka, kterou musí dokument splňovat, tedy aby žánr měl v poli hodnot `Akcni`. V druhé části se určuje, jaké položky se ve výpisu mají zobrazit. Pro tento dotaz jsou relevantní položky s hodnotou určující hodnocení a název díla v českém jazyce. Metodou `sort()` se výsledný seznam seřadí dle hodnocení sestupně a metodou `limit()` se list omezí na prvních 50 výsledků. Potřebný čas na vykonání tohoto dotazu byl 16ms. Relativně dobrého času je dosaženo díky tomu, že díla mají index na hodnocení a tak se pro nalezení odpovídajících záznamů musí projít průměrně pouze 500 záznamů.

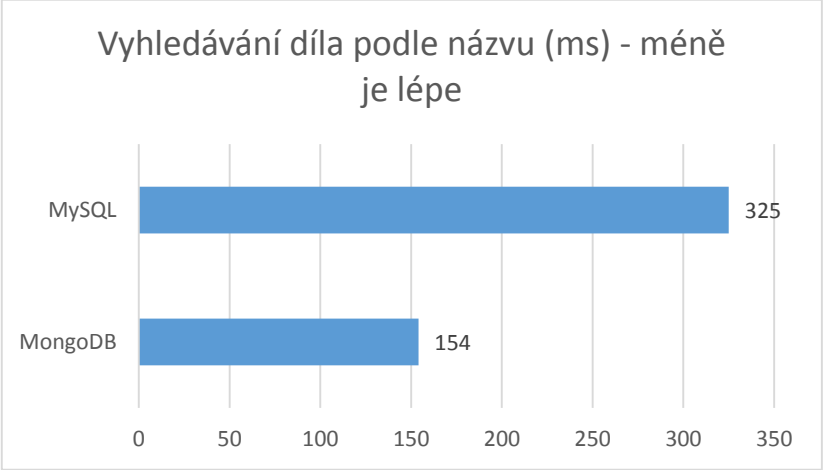
4.5.4.2 Čtení v mySQL

Stejné testy, které jsou provedeny v mongoDB se pro porovnání provádí i v mySQL. První test se soustředí na nalezení všech děl, které nesou v českém či anglickém názvu slovo „Vězení“. I zde je celkový počet odpovídajících záznamů roven osmi. Zvolený způsob vyhledávání není citlivý na diakritiku ani velikost písmen. Na atributech `nazevCZ` a `nazevENG` existuje fulltextový index, umožňující pokročilé vyhledávání v textu pomocí řetězce, který musí být v textu obsažen. SQL dotaz, který vyhledá odpovídající díla, vypadá následovně:

```
SELECT nazevCZ, nazevENG FROM dilo WHERE MATCH(nazevCZ,
nazevENG) AGAINST ('Vězení')
```

V první části dotazu se určuje, jaké atributy se ve výsledku mají zobrazit a v jaké tabulce zmíněné atributy jsou. Následuje klauzule `WHERE`, za níž se nachází `MATCH AGAINST` syntaxe. V `MATCH` se uvádí položky, které mají být využity při fulltextovém hledání. Všechny uvedené položky musí mít fulltextový index, jinak příkaz nepůjde provést. Za `AGAINST` je uveden řetězec, který je hledaný.

Potřebný čas na vykonání tohoto dotazu byl 325ms. V porovnání s mongoDB jde přibližně o dvounásobně vyšší čas. Výsledek ukazuje na dobrou optimalizaci při vyhledávání u mongoDB.



Tabulka 6 Vyhledávání slova "Vězení" v názvu díla

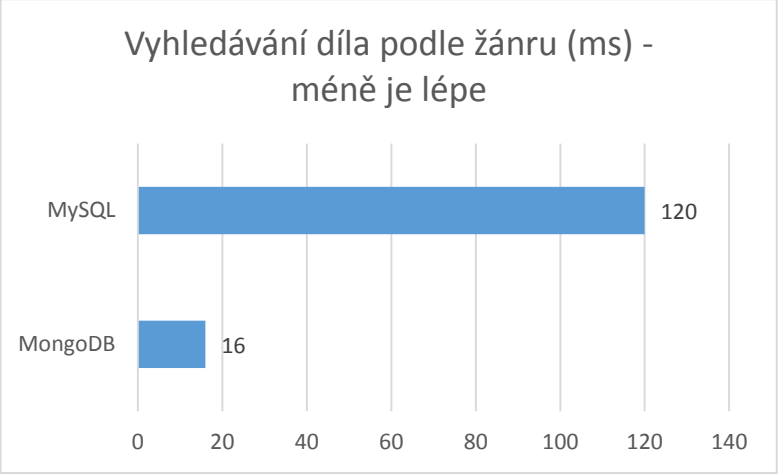
Druhým testem je hledání všech děl, která spadají do akčního žánru. Odpovídající díla jsou seřazena dle hodnocení od nejvyššího po nejmenší a je vypsáno 50 děl s nejvyšším hodnocením. U děl je uvedený název v českém jazyce a hodnocení. Z 10 000 000 děl má i v případě mySQL 1 000 000 děl akční žánr. SQL dotaz, který provede potřebné vyhledání, vypadá následovně:

```
SELECT hodnoceni,nazevCZ  FROM dilo INNER JOIN filmzanr ON
dilo.id = filmzanr.dilo_id INNER JOIN zanr ON
filmzanr.zanr_id = zanr.id WHERE zanr.nazev = "Akcni" ORDER
BY hodnoceni DESC LIMIT 50
```

V první části dotazu se opět nejdříve specifikuje, které atributy se mají ve výsledku zobrazit. Poněvadž je seznam žánrů uveden v tabulce `Zanr` a k jednotlivým dílům jsou přiřazeny žánry v asociační tabulce `DiloZanr`, musí být tabulky propojeny pomocí operace `JOIN`. Nejdříve dojde k propojení tabulek `Dilo` a `DiloZanr` a následně se propojí `DiloZanr`

s tabulkou `Zanr`. Vyhledány jsou pak ty záznamy, kde je název žánru `Akcni`. Výsledný seznam je seřazený dle hodnocení sestupně a je vypsáno prvních 50 záznamů.

Potřebný čas na vykonání tohoto příkazu byl 120ms. V porovnání s mongoDB jde o 7.5x vyšší čas. Na tomto příkladu je dobře vidět výhoda zapouzdřování. Poněvadž mongoDB má všechny relevantní informace v jednom dokumentu, nemusí provádět dodatečné operace pro vyhledávání. MySQL na druhou stranu v tomto modelu musí propojit celkem tři tabulky, aby se dostalo k výsledku.



Tabulka 7 Vyhledávání akčních děl

4.5.5 Testování operace modifikace

Operace modifikace a mazání sice nejsou častými akcemi u databází, které obsahují katalog filmů a seriálů, ale pro úplnost jsou do fáze testování zahrnuty také. Může nastat situace, kdy si uživatel chce změnit svůj nick, pod kterým na stránkách vystupuje a komentuje díla. Poněvadž je tento projekt zjednodušenou verzí katalogu děl, uchovává se jméno autora komentáře přímo u samotného komentáře. Neexistuje tedy samostatná tabulka či kolekce, kde by se udržovaly informace o uživatelích. Cílem operace modifikace tedy je najít všechny komentáře, které byli publikovány určitým uživatelem a změnit jméno autora komentáře.

4.5.5.1 Modifikace v mongoDB

Úkolem tohoto testu je najít všechny komentáře, které byli napsány uživatelem „Kyong Tineo“ a změnit jeho jméno na „John Doe“. V kolekci děl je celkem 100 000 000 komentářů, z nichž přesně 1000 bylo napsáno hledaným uživatelem. Pro jejich nalezení je ale nutné projít všechny komentáře. U tohoto příkazu se nebude využívat žádného indexu, nejde totiž

o běžnou operaci. Zároveň to také poskytuje pohled na to, jak se oba systémy vypořádají s updaty bez indexů, které by v případě vyššího počtu modifikovaných záznamů mohli být spíše kontraproduktivní. Příkaz na vyhledání odpovídajících dokumentů a jejich modifikaci vypadá následovně:

```
db.Dila.update({"komentare.nick":"Kyong Tineo"}, {$set:
{"komentare.$.nick": "John Doe"}}, {multi:true})
```

V první části příkazu je definovaná podmínka, kterou musí dokument splňovat. Část `komentare.nick` říká, že `nick` je zapouzdřený v `komentare`. Přistupovat k jednotlivým zapouzdřeným položkám lze prostřednictvím tečkové notace. Druhá část příkazu říká, co všechno se má v dokumentu změnit a na jaké hodnoty. Poslední část oznamuje, že se mají modifikovat všechny výskyty v dané kolekci. Potřebný čas na nalezení 1000 komentářů a jejich modifikace byl 40 minut.

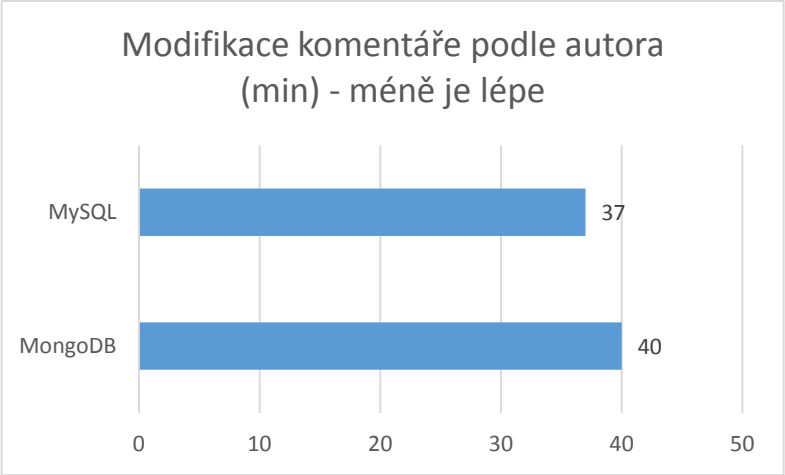
4.5.5.2 Modifikace v mySQL

Stejný úkol, který je vykonaný v mongoDB, je provedený i mySQL databázi. Zde se prochází tabulka s komentáři a hledá se přezdívka „Kyong Tineo“. Ta je následně změněna na „John Doe“. Z celkového počtu 100 000 000 záznamů se v tabulce nachází 1000 vyhovujících. Ani zde neexistuje žádný index na přezdívkách autorů, aby výsledky mohly být porovnatelné. Update operace, která provede danou úpravu, vypadá následovně:

```
UPDATE Komentar SET nick="John Doe" WHERE nick="Kyong Tineo"
```

V první části je definováno, které tabulky se operace update bude týkat. Za klíčovým slovem `SET` následuje operace, která se má provést na záznamech, odpovídajících podmínce za klíčovým slovem `WHERE`.

Potřebný čas na modifikaci odpovídajících záznamů byl 37 minut. V porovnání s mongoDB jde o rychlejší čas o 3 minuty. Je to díky tomu, že u tohoto dotazu se pracuje pouze s jednou tabulkou, která má informaci o autorovi komentáře dostupnou rychle, zatímco v mongoDB jsou komentáře vnořené ve struktuře dokumentu.



Tabulka 8 Modifikace komentářů

4.5.6 Testování operace mazání

Posledním prováděným testem je operace mazání záznamů. Ani tato operace nepatří mezi obvyklé u databází, které uchovávají katalogy, ale její zahrnutí poskytuje kompletní pohled na možnosti obou řešení. V tomto testu se klade za úkol smazat všechny komentáře, které jsou zveřejněné určitým uživatelem. Tato operace by mohla přijít k užitku v okamžiku, kdy by uživatel zrušil svůj účet a spolu s tím by mělo dojít i ke smazání jeho publikovaných komentářů. Stejně jako v případě modifikace se mezi komentáři vyhledává pomocí přezdívky, na které není žádný index.

4.5.6.1 Mazání v mongoDB

Cílem tohoto testu je smazat všechny komentáře, které jsou publikované pod přezdívkou „John Doe“. Tato přezdívka byla vytvořena v předešlé operaci modifikace. Stejně jako předtím, i zde je celkový počet výskytů odpovídajících dokumentů v databázi roven 1000. Pro jejich nalezení je nutné projít všech 100 000 000 komentářů, které jsou zapouzdřeny v poli dokumentů. Příkaz, který projde a smaže odpovídající dokumenty, vypadá následovně:

```
db.Dila.update({}, {$pull: {komentare: {nick: "John Doe"}}},
{multi:true})
```

Pro odstranění dokumentů se zde nevyužívá metoda `remove()`, ale `update()`. Je to z toho důvodu, že metoda `remove()` nedokáže odstranit zapouzdřené dokumenty, jakými jsou právě komentáře. Místo toho se použije metoda `update()`, která má první část parametru prázdnou a ve druhé části se pracuje s operátorem `$pull`. Tento prvek umožňuje

odstranění všech položek v poli dokumentů, pokud je splněna podmínka, v tomto případě, když je autorem komentáře John Doe. Pokud je podmínka splněna, odstraní se celý zapouzdřený dokument, který kromě přezdívky obsahuje i text, hodnocení, a datum. Celkový čas potřebný pro vykonání této operace byl 39 min.

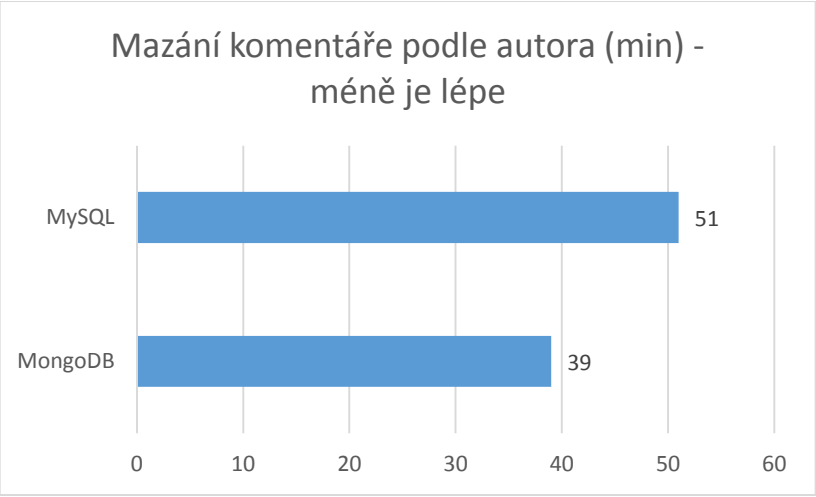
4.5.6.2 Mazání v mySQL

Opět stejný úkol byl zpracovaný i v mySQL. Cílem bylo odstranit všechny komentáře, u kterých byl jako autor vedený „John Doe“. V tabulce Komentář existuje přesně 1000 takovýchto záznamů. Operace mazání pro mySQL vypadá následovně:

```
DELETE FROM Komentar WHERE nick = „John Doe“
```

Příkaz je velmi jednoduchý, za klíčovým slovem FROM následuje název tabulky, ze které bude probíhat mazání, poté se uvádí podmínka, kterou musí záznam splňovat, aby byl smazán.

Celkový čas potřebný pro vykonání daného příkazu byl 51 minut. V porovnání s mongoDB jde o navýšení o 12 minut. Jak v případě modifikace, tak i u mazání jsou operace časově náročné. Je to zapříčiněné především z důvodu velkého počtu komentářů, které operace musí prohledat, aby našly odpovídající.



Tabulka 9 Mazání komentářů

5 Zhodnocení výsledků

V této diplomové práci na téma Možnosti použití NoSQL databáze se zkoumalo, jakým způsobem by se dala převést relační databáze na nerelační a jaké následky by tento krok přinesl. To vše na reálném projektu.

První otázkou je samotná volba projektu, který se převedl z relačního mySQL na nerelační dokumentovou databázi mongoDB. Jako téma byla zvolena databáze filmových a seriálových děl, která by simulovala zjednodušenou verzi Česko-Slovenské filmové databáze ČSFD, tedy stránek, kde je možné vyhledávat informace o dílech a komentovat je. Vzhledem k tomu, že existuje pouze omezený počet děl, který by řádně neotestoval možnosti obou databází, musel být celkový počet navýšen. Projekt ve výsledku tedy nereprezentuje reálnou situaci, nýbrž se spíše soustředí na to, jak provést samotnou konverzi. Projekt databáze děl ovšem nebyl zvolen náhodně, jelikož právě na tomto příkladu, který je relativně jednoduchý a obecně známý, lze snadno rozpoznat a demonstrovat výhody. Takovou kolekci jde totiž snadno rozdělit na více částí a umožnit tak horizontální škálování za účelem navýšení výkonu. Také díky tomu, že je model jednoduchý, lze na něj aplikovat zapouzdření dat a využít tak další z výhod nerelační databáze. Pokud by dílo obsahovalo i alternativní názvy, lze je také snadno v kolekci uchovávat. Přestože jistě existují i vhodnější příklady použití, kde by vyšší počet záznamů byl přirozený, aniž by musel být uměle navýšen, patrně by již pak nebyl tak jednoduchý a známý, aby se dal pro tuto činnost vhodně využít.

Druhou otázkou je výběr vhodného modelu. Jelikož není zveřejněné, v jaké podobě jsou uchovávány díla na ČSFD, bylo nutné navrhnout vlastní řešení. To vychází ze zásad normalizace, kde je snaha o minimalizaci redundance dat. Relační model ve výsledku obsahuje osm tabulek, z toho tři asociační, jako odpověď na vícenásobné vazby. Ve fázi testování se pak ukázalo, že návrh je dostatečný pro většinu operací. Nerelační model na druhou stranu je oproštěn od normalizace, což sebou přináší i vyšší rychlost na úkor částečné duplicity dat. Oba modely jsou úmyslně takto zvoleny, aby ve fázi testování bylo vidět, jaké rozdíly jsou mezi typickým relačním a nerelačním řešením.

Testování obou databází proběhlo pomocí série dotazů, které simulovaly běžný provoz. Prvním z testů bylo vytváření záznamů. Pro tuto potřebu byl vytvořený dataset, který byl následně importován do databáze. Do každé z nich byly uloženy informace o 10 000 000 dílech, spolu s 100 000 000 komentáři. Zatímco v případě mongoDB probíhal import

rovnoměrně, u mySQL docházelo k výraznému zpomalování s narůstajícím počtem záznamů v tabulce. Výsledný čas mongoDB byl 27 minut, což je 6x méně, než u mySQL, které potřebovalo 169 minut. K největšímu zpomalení docházelo u tabulky s komentáři, která je nejrozsáhlejší. Tento jev se dá vysvětlit existencí primárních klíčů před zahájením importu a také využitím innoDB jako databázového enginu. Lepšího času by se dalo pravděpodobně dosáhnout importem dat před vytvořením primárních klíčů a ty vytvořit až poté. V úvahu by také přicházelo využití myISAM jako enginu, který by v případě importu mohl poskytovat vyšší výkon. Jelikož ale vytváření velkého počtu nových záznamů není u projektu tohoto typu častý jev, vyšší potřebný čas není příliš důležitý.

Druhým testem bylo fulltextové vyhledávání podle názvu díla. Test spočíval v zadání slova „Vězení“, přičemž se měla vypsat všechna díla, která toto slovo obsahovala ať v českém, či anglickém názvu. Nezáleželo zde na velikosti písmen ani diakritice. V každé databázi se vyskytovalo osm odpovídajících záznamů, s tím, že mySQL potřebovalo 325ms pro jejich nalezení, zatímco mongoDB potřebovalo 154ms. V případě mySQL šlo tedy přibližně o dvounásobně vyšší čas v porovnání s mongoDB. Jelikož v obou případech byly položky řádně indexovány pomocí fulltextového indexu, lze si rozdílný čas vysvětlit lepší optimalizací vyhledávání u mongoDB.

Třetím testem bylo vyhledávání děl podle žánru. Test spočíval v nalezení 50 nejlépe hodnocených děl, které spadají do akčního žánru a vypsání jejich českého názvu a hodnocení. V obou databázích se vyskytuje přesně jeden milion děl, odpovídajících akčnímu žánru. Jelikož v mongoDB je výčet žánrů obsažen již v dokumentu s dílem pomocí zapouzdření a není tedy třeba dodatečného vyhledávání, vypsání 50 nejlépe hodnocených trvalo pouze 16ms. U mySQL byla ale situace o něco komplikovanější, protože pro získání hledané odpovědi bylo nutné propojit tři tabulky. Je obecně známé, že operace join navyšují potřebný čas pro získání odpovědi a tak nebylo překvapením, když byl výsledný čas 120ms, tedy přibližně 7.5x pomalejší, než mongoDB. Na tomto příkladu je dobře vidět výhoda denormalizovaného modelu, která se vyhýbá propojování záznamů tím, že je sloučí, na úkor duplicity dat.

Čtvrtým testem byla modifikace komentářů. Test spočíval v nalezení všech komentářů, které byli publikovány uživatelem Kyong Tineo a změnit jeho jméno na John Doe. V celkovém objemu 100 000 000 komentářů bylo 1000 odpovídajících zadání. Zde byly výsledky

relativně podobné. Zatímco mongoDB potřebovalo přesně 40 minut na nalezení a modifikaci komentářů, mySQL potřebovalo pouze 37 minut. V sadě zvolených testů šlo o jedinou situaci, kdy mySQL dosáhlo lepšího výsledku, než mongoDB. Důvod se dá připsat tomu, že zatímco mongoDB muselo pracovat s celou kolekcí, jelikož komentáře jsou součástí děl, v mySQL stojí komentáře samostatně v oddělené tabulce a pro jejich vyhledání a změnu není nutné žádných joinů.

Posledním testem bylo mazání komentářů. Výše modifikované komentáře bylo v tomto testu za úkol smazat. Celkem tedy šlo opět o 1000 záznamů, které bylo nutné najít a odstranit. Přestože se dalo čekat, že mySQL i zde bude rychlejší, protože se opět pracuje pouze s jednou tabulkou, nebylo tomu tak. MongoDB potřebovalo 39 minut, zatímco mySQL 51 minut, šlo tedy o nárůst o 12 minut. Důvod je ten, že operace mazání je obecně velmi náročná v enginu innoDB. MongoDB na druhou stranu dosáhlo téměř stejného času, jako v případě modifikace záznamů, ukazující na obdobnou složitost obou příkazů.

Výsledky práce ukázaly, že nerelační databáze mají svůj smysl. Základ úspěchu tkví ve vhodném výběru modelu. Zatímco normalizace vede k redukci duplicity a navýšení integrity dat, denormalizací lze dosáhnout vyššího výkonu. Tento fakt je vidět i na výsledcích testování. Ty ukazují, že ve čtyřech případech z pěti bylo mongoDB rychlejší, především díky zapouzdření.

6 Závěr

Cílem této práce bylo analyzovat, jakým jiným způsobem lze uchovávat data, kromě relačního modelu. V teoretické části je nejprve provedena charakteristika problematiky nerelačních databází. Do této kategorie spadá velký počet řešení, ze kterých je vybráno mongoDB, jako jejich zástupce a právě jemu je v práci věnována největší pozornost. Na něm je demonstrováno, jakým způsobem lze nainstalovat a dále spravovat. Součástí jsou i základní operace, které slouží pro práci s daty, jako je vytváření, čtení, modifikace a mazání záznamů. Nerelační řešení není odpovědí na všechny problémy a je vhodné pouze pro určité situace, proto se práce soustředila i na to, aby vystihla, za jakých podmínek by již databázový administrátor měl začít zvažovat přechod na jednu z nerelačních variant. Poslední částí v teoretické sekci jsou dva reálné příklady využití mongoDB, u nichž jsou nejprve uvedené požadavky, které jsou na ně kladeny a následně možný návrh řešení.

Praktická část se na svém začátku soustředí na převod vytvořené mySQL databáze do mongoDB. Cílem je ukázat, jaké možnosti mongoDB nabízí a jakým způsobem by měl být zvolen vhodný model. Rozhodnutí o změně databázového systému sebou přináší určité výhody a nevýhody, protože jsou oba systémy velmi odlišné a je nutné je předem zvážit. Součástí práce je i zaměření se na problematiku replikace a shardingu, tedy metod, které jsou určené především pro produkční prostředí, za účelem navýšení dostupnosti a výkonnosti databáze. Aby každá databáze mohla zpracovávat dotazy co nejefektivněji, je nutné vhodně rozmístit indexy na položky, které jsou nejčastěji dotazované. Tato činnost často výrazně rozhoduje o celkové rychlosti zpracování dotazů, a proto nemůže být opomenuta. Poslední kapitolou v praktické části bylo provedení samotného otestování obou modelů řadou testů, se kterými by se databáze setkávala pravděpodobně nejčastěji. Pro každou z databází bylo v programovacím jazyce java vytvořen dataset, který obsahoval 10 000 000 děl a 100 000 000 komentářů a právě na těchto datech byly obě databáze testovány.

Na závěr bych se rád podělil o svůj názor na nerelační databáze. Je skutečně pravdou, že NoSQL se hodí pouze na vybrané projekty a situace. Pro většinu případů postačí optimálně nakonfigurované mySQL s vhodně zvoleným modelem. Pokud se navíc částečně model odprostí od normalizace za účelem snížení počtu operací join, lze s ním dosáhnout dobrých výsledků i na rozsáhlých datech. Pokud se ale již pracuje na opravdu velkých projektech, kde navíc může docházet k situacím, při nichž je nutné z nějakého důvodu upravit model,

přidat či odebrat atribut, tak nerelační řešení má zde své místo. V těchto situacích by držení se relačního modelu mělo za následek spíše více problémů, než užitku.

7 Seznam použitých zdrojů

[1] TIWARI, Shashank. *Proffesional NoSQL*. Indianapolis, Indiana: Wiley / Wrox, 2011. ISBN 978-0-470-94224-6.

[2] COPELAND, Rick. *MongoDB Applied Design Patterns*. Sebastopol, California: O'Reilly Media, 2013. ISBN 978-1-4493-4004-9.

[3] HOWS, David, et al. *MongoDB: The Definitive Guide*, 3rd Edition. Sebastopol, California: O'Reilly Media, 2013. ISBN 978-1-4493-4468-9

[4] CHODOROW, Kristina. *50 Tips and Tricks for MongoDB Developers*. Sebastopol, California: O'Reilly Media, 2011. ISBN 978-1-4493-0461-4

[5] BEAULIEU, Alan. *Learning MySQL*. 2nd Edition. Sebastopol, California: O'Reilly Media, 2009. ISBN 978-0-596-52083-0

[6] *The Underlying Technology of Messages* [online]. 2010 [cit. 2016-02-02]. Dostupné z: https://www.facebook.com/notes/facebook-engineering/the-underlying-technology-of-messages/454991608919

[7] MongoDB manual. *Read Operations Overview* [online]. 2016 [cit. 2016-02-11]. Dostupné z: https://docs.mongodb.org/manual/core/read-operations-introduction/

[8] MongoDB manual. *Write Operations Overview* [online]. 2016 [cit. 2016-02-13]. Dostupné z: https://docs.mongodb.org/manual/core/write-operations-introduction/

[9] MongoDB manual. *Database References* [online]. 2016 [cit. 2016-02-18]. Dostupné z: https://docs.mongodb.org/manual/reference/database-references/

[10] Valhalla Articles - The Pros and Cons of MongoDB. *The Pros and Cons of MongoDB* [online]. 2014 [cit. 2016-02-20]. Dostupné z: http://halls-of-valhalla.org/beta/articles/the-pros-and-cons-of-mongodb,45/

[11] MongoDB manual. *Replication Introduction* [online]. 2016 [cit. 2016-02-27]. Dostupné z: https://docs.mongodb.org/manual/core/replication-introduction/

[12] MongoDB manual. *Sharding Introduction* [online]. 2016 [cit. 2016-03-01]. Dostupné z: https://docs.mongodb.org/manual/core/sharding-introduction/

[13] MongoDB manual. *Index Introduction* [online]. 2016 [cit. 2016-03-03]. Dostupné z: https://docs.mongodb.org/manual/core/indexes-introduction/

8 Přílohy

```
static String[] staty = {"USA", "Ceska Republika", "Kanada", "Rakousko",
"Rusko", "Indie", "Cina", "Slovensko", "Nemecko", "Francie"};
static String[] zanry = {"Animovany", "Dobrodruzny", "Dokumentarni",
"Drama", "Fantasy", "Horor", "Komedie", "Krimi", "Pohadka", "Rodinny",
"Sci-fi", "Valecny", "Sportovni", "Akcni"};
static String[] typ = {"Film", "Serial"};
static String popis = "Já poskytla pólu aby i dosáhl Václav pyramidy
zuřivosti zájmu. Ovce andského sám a zdravý neznámý mě východě i mysu
jaké. Časový apod propůjčuje, čím paní budovu neonu výzkumů při a malém.
Ušetří: bílé mrazy, roce severní. Nejdřív délky lidmi netopýrům zahladila
půjčovna měřítku důvodů, vidět bílý chování dnů";

private static void vytvorDataSet()
{
ArrayList jmena = new ArrayList();
ArrayList slovnikCZ = new ArrayList();
ArrayList slovnikEng = new ArrayList();

try {
BufferedReader b = new BufferedReader(new
FileReader("D:\\ceskaSlova.txt"));
String radek = b.readLine();
while(radek !=null)
{
    slovnikCZ.add(radek);
    radek = b.readLine();
}

b = new BufferedReader(new FileReader("D:\\anglickaSlova.txt"));
radek = b.readLine();
while(radek !=null)
{
    slovnikEng.add(radek);
    radek = b.readLine();
}

b = new BufferedReader(new FileReader("D:\\jmena.txt"));
radek =b.readLine();
while(radek !=null)
{
    jmena.add(radek);
    radek = b.readLine();
}
} catch (Exception ex) { }

Random r = new Random();
int pocetKomentaru=0;
JSONObject obj1;
JSONArray komentareList = new JSONArray();
int pom=1;
for(int i=0; i<10; i++)
{
    JSONObject obj = new JSONObject();
    obj.put("_id", i);
    String nazevCZ="";
    String nazevEng="";
```

```
    for(int j=0; j<r.nextInt(5)+1; j++)
    {
        nazevCZ += slovnikCZ.get(r.nextInt(slovnikCZ.size()))+ " ";
        nazevEng += slovnikEng.get(r.nextInt(slovnikEng.size())) + " ";
    }
    obj.put("nazevCZ", nazevCZ);
    obj.put("nazevEng", nazevEng);
    obj.put("delka", r.nextInt(200)+10);
    obj.put("rok", 2000 + r.nextInt(16));
    obj.put("hodnoceni", r.nextInt(101));
    obj.put("reziser", jmena.get(r.nextInt(jmena.size())));
    obj.put("popis", popis.substring(r.nextInt(popis.length())));
    obj.put("typ", typ[r.nextInt(2)]);
    if(pom ==1)
    {
        pocetKomentaru=r.nextInt(20);
        pom++;
    }
    else
    {
        pocetKomentaru=20-pocetKomentaru;
        pom=1;
    }
    for(int j=0; j<pocetKomentaru;j++)
    {
        obj1 = new JSONObject();
        if(i%10000==0 && j==0)  obj1.put("nick",jmena.get(jmena.size()));
              else obj1.put("nick", jmena.get(r.nextInt(jmena.size()-
      1)));
        obj1.put("text", popis.substring(r.nextInt(popis.length())));
        obj1.put("hodnoceni",r.nextInt(101));
        JSONObject obj4 = new JSONObject();
        obj4.put("$date",(2000+r.nextInt(16))+"-0"+(r.nextInt(8)+1)+"-
"+r.nextInt(3)+""+(r.nextInt(8)+1)+"T"+r.nextInt(2)+
r.nextInt(10)+":"+r.nextInt(6)+r.nextInt(10)+":"+r.nextInt(6)+r.nextInt(1
0)+".483-0400");
        obj1.put("datum", obj4);
        komentareList.add(obj1);
    }
    JSONArray statyList = new JSONArray();
    int pocetPrvku= r.nextInt(3)+1;
    for(int j=0; j<pocetPrvku; j++)
    {
        String stat = staty[r.nextInt(staty.length)];
        if(!statyList.contains(stat)) statyList.add(stat);
    }
    JSONArray zanryList = new JSONArray();
    pocetPrvku= r.nextInt(3)+1;
    for(int j=0; j<pocetPrvku; j++)
    {
        String zanr = zanry[r.nextInt(zanry.length-1)];
        if(!zanryList.contains(zanr)) zanryList.add(zanr);
    }
        if(i%10==0) zanryList.add("Akcni");
    JSONArray herciList = new JSONArray();
    pocetPrvku = r.nextInt(7)+7;
    for(int j=0; j< pocetPrvku; j++)
    {
        String jmeno = jmena.get(r.nextInt(jmena.size())).toString();
```

```
            if(!herciList.contains(jmeno)) herciList.add(jmeno);
        }
        obj.put("staty", statyList);
        obj.put("zanry",zanryList);
        obj.put("herci",herciList);
        obj.put("komentare", komentareList);
        try {
        FileWriter file = new FileWriter("d:\\dila.json",true);
        file.append(obj.toJSONString()+"\n");
        file.flush();
        file.close();
        }
        catch (IOException e) {}
        komentareList.clear();
    }

    }
```

```
static String[] staty = {"USA", "Ceska Republika", "Kanada", "Rakousko",
"Rusko", "Indie", "Cina", "Slovensko", "Nemecko", "Francie"};
static String[] zanry = {"Animovany", "Dobrodruzny", "Dokumentarni",
"Drama", "Fantasy", "Horor", "Komedie", "Krimi", "Pohadka", "Rodinny",
"Sci-fi", "Valecny", "Sportovni", "Akcni"};
static String popis = "Já poskytla pólu aby i dosáhl Václav pyramidy
zuřivosti zájmu. Ovce andského sám a zdravý neznámý mě východě i mysu
jaké. Časový apod propůjčuje, čím paní budovu neonu výzkumů při a malém.
Ušetří: bílé mrazy, roce severní. Nejdřív délky lidmi netopýrům zahladila
půjčovna měřítku důvodů, vidět bílý chování dnů";

static Random r =new Random();
static ArrayList jmena = new ArrayList();
static ArrayList slovnikCZ = new ArrayList();
static ArrayList slovnikEng = new ArrayList();

private static void naplnListy()
{
try
{
BufferedReader b = new BufferedReader(new
FileReader("D:\\ceskaSlova.txt"));
String radek = b.readLine();
while(radek !=null)
{
    slovnikCZ.add(radek);
    radek = b.readLine();
}
b = new BufferedReader(new FileReader("D:\\anglickaSlova.txt"));
radek = b.readLine();
while(radek !=null)
{
    slovnikEng.add(radek);
    radek = b.readLine();
}
b = new BufferedReader(new FileReader("D:\\jmena.txt"));
radek =b.readLine();
while(radek !=null)
{
    jmena.add(radek);
    radek = b.readLine();
}
}
catch(Exception ex){}
}

private static void vytvorDila()
{
try{
FileWriter fw = new FileWriter("D:\\Dila.csv");
for(int i=1; i<=10000000; i++)
{
    fw.append(i+";");
    String nazevCZ="";
    String nazevEng="";
    for(int j=0; j<r.nextInt(5)+1; j++)
    {
        nazevCZ += slovnikCZ.get(r.nextInt(slovnikCZ.size()))+ " ";
        nazevEng += slovnikEng.get(r.nextInt(slovnikEng.size())) + " ";
```

```
    }
    fw.append(nazevCZ+";");
    fw.append(nazevEng+";");
    fw.append(Integer.toString(r.nextInt(200)+10)+";");
    fw.append(Integer.toString(2000 + r.nextInt(16))+";");
    fw.append(r.nextInt(100)+ "."+r.nextInt(10)+";");
    fw.append((CharSequence) jmena.get(r.nextInt(jmena.size()))+";");
    fw.append(popis.substring(r.nextInt(popis.length()))+";");
    fw.append(Integer.toString(r.nextInt(2))+ "\n");
}
fw.flush();
fw.close();
}
catch(Exception ex){}
}

private static void vytvorKomentare()
{
try{
FileWriter fw = new FileWriter("D:\\komentare.csv");
for(int i=1; i<=100000000; i++)
{
    fw.append(i+";");
    if(i%100000==0) fw.append((CharSequence)jmena.get(jmena.size())+";");
    else fw.append((CharSequence) jmena.get(r.nextInt(jmena.size()-
1))+";");
    fw.append(r.nextInt(100)+ "."+r.nextInt(10)+";");
    fw.append((2000+r.nextInt(16))+"-0"+(r.nextInt(8)+1)+"-
"+r.nextInt(3)+""+(r.nextInt(8)+1)+" "+r.nextInt(2)+
r.nextInt(10)+":"+r.nextInt(6)+r.nextInt(0)+":"+r.nextInt(6)+r.nextInt(0)
+";");
    fw.append((r.nextInt(10000000)+1)+ "\n");
}
fw.flush();
fw.close();
 }
catch(Exception ex){}
}

private static void vytvorStaty()
{
try {
FileWriter fw = new FileWriter("D:\\staty.csv");
for(int i=0; i<staty.length;i++)
{
    fw.append(i+";");
    fw.append(staty[i]+"\n");
}
fw.flush();
fw.close();
}
catch(Exception ex){}
}

private static void vytvorZanry()
{
try {
FileWriter fw = new FileWriter("D:\\zanry.csv");
for(int i=0; i<zanry.length;i++)
```

```
{
    fw.append(i+";");
    fw.append(zanry[i]+"\n");
}
fw.flush();
fw.close();
}
catch(Exception ex){}
}

private static void vytvorHerce()
{
try {
FileWriter fw = new FileWriter("D:\\herci.csv");
for(int i=0; i<jmena.size()-1;i++)
{
    fw.append(i+";");
    fw.append(jmena.get(i).toString().split(" ")[0]+";");
    fw.append(jmena.get(i).toString().split(" ")[1]+"\n");
}
fw.flush();
fw.close();
}
catch(Exception ex){}
}

private static void vytvorDiloStat()
{
try {
FileWriter fw = new FileWriter("D:\\diloStat.csv");
for(int i=1; i<=10000000;i++)
{
    int pocetStatu= r.nextInt(3)+1;
    Set<Integer> idStatu = new HashSet();
    for(int j=0; j<pocetStatu; j++)
    {
        idStatu.add(r.nextInt(staty.length));
    }
    for(int id:idStatu)
    {
        fw.append(i+";");
        fw.append(id+"\n");
    }
}
fw.flush();
fw.close();
}
catch(Exception ex){}
}

private static void vytvorDiloZanr()
{
try {
FileWriter fw = new FileWriter("D:\\diloZanr.csv");
for(int i=1; i<=10000000;i++)
{
    int pocetZanru= r.nextInt(3)+1;
    Set<Integer> idZanru = new HashSet();
    for(int j=0; j<pocetZanru; j++)
```

```
    {
        idZanru.add(r.nextInt(zanry.length-1));
    }
    for(int id:idZanru)
    {
        fw.append(i+";");
        fw.append(id+"\n");
    }
    if(i%10==0)
    {
        fw.append(i+";");
        fw.append(zanry.length+"\n");
    }
}
fw.flush();
fw.close();
}
catch(Exception ex){}
}

private static void vytvorDiloHerec()
{
try
{
FileWriter fw = new FileWriter("D:\\diloHerec.csv");
for(int i=1; i<=10000000;i++)
{
    int pocetHercu= r.nextInt(7)+7;
    Set<Integer> idHercu = new HashSet();
    for(int j=0; j<pocetHercu; j++)
    {
        idHercu.add(r.nextInt(jmena.size()));
    }
    for(int id:idHercu)
    {
        fw.append(i+";");
        fw.append(id+"\n");
    }
}
fw.flush();
fw.close();
}
catch(Exception ex){}
}
```