Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

Studijní program: Aplikovaná informatika

Obor: Informatika

Srovnání řešení správy dat v MySQL a MongoDB při použití Doctrine 2 ve frameworku Symfony 2

BAKALÁŘSKÁ PRÁCE

Student : Dominik Firla

Vedoucí : RNDr. Helena Palovská, Ph. D.

Oponent : Ing. Jan Mittner

2016

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze kterých jsem čerpal.

V Praze dne 22. 4. 2016 . . . . . .

Dominik Firla

Poděkování

Děkuji RNDr. Heleně Palovské, PhD. za odborné vedení práce a vstřícnost. Také bych rád poděkoval svým blízkým za cenné rady a komentáře.

Abstrakt

Tato práce se zabývá porovnáním relačního databázového systému a dokumentové databáze na teoretické i praktické úrovni. V teoretické části práce jsou vysvětleny principy relačních a dokumentových databází. Blíže jsou představeny databáze MySQL aMongoDB. V praktické části je realizováno řešení správy dat pro modelovou situaci. Oba systémy jsou porovnány jak z hlediska implementace, tak výkonu. Nabyté poznatky jsou shrnuté v závěru práce a měly by čtenáři napomoci při výběru správného databázového systému pro webovou aplikaci nejen v PHP frameworku Symfony. Součástí praktické části práce je také autorem vytvořený projekt, ve kterém jsou obě databáze pro modelovou situaci implementovány.

Klíčová slova

Databáze, MySQL, MongoDB, Symfony, Doctrine, ORM, ODM, Vagrant.

Abstract

This thesis focuses on comparing a relational database system with a document database on both theoretical and practical levels. In theoretical part of this thesis principles of relational and document databases are explained. Closer attention is given to MySQL and MongoDB databases. In practical part a solution to data management is realized for a model situation. Both systems are compared in terms of implementation and performance. Knowledge gained is summarized in conclusion of this thesis and should help a reader when choosing the right database system not only for a web application implementing PHP framework Symfony. Practical part of the thesis also includes an author-made project containing implementation of both databases in the model situation.

Keywords

Database, MySQL, MongoDB, Symfony, Doctrine, ORM, ODM, Vagrant.

Obsah

1 Úvod
1.1 Cíl práce ... 1
1.2 Rešerše podobně zaměřených prací ... 2

2 MySQL ... 4
2.1 Principy relačních databází ... 4
2.1.1 Vysvětlení základních pojmů ... 4
2.1.2 Asociace ... 5
2.1.3 Index ... 7
2.1.4 ACID ... 7
2.1.5 Normální formy ... 8
2.2 Charakteristika MySQL ... 9
2.2.1 Vznik a vývoj ... 9
2.2.2 Práce se systémem ... 10
2.2.3 Úložné formáty ... 11

3 MongoDB ... 12
3.1 Principy NoSQL databází ... 12
3.1.1 Vznik a vývoj ... 12
3.1.2 Kategorizace ... 13
3.1.3 Úložiště klíč – hodnota ... 13
3.1.4 Sloupcová úložiště ... 14
3.1.5 Grafové databáze ... 15
3.1.6 Dokumentové databáze ... 16
3.2 MongoDB ... 17
3.2.1 Vznik a vývoj ... 17
3.2.2 Dokumentový datový model ... 17
3.2.3 Asociace ... 18

4 Implementace databáze v modelové aplikaci ... 21
4.1 Popis prostředí ... 21
4.1.1 Platforma ... 21
4.1.2 Zprovoznění projektu ... 22
4.1.3 Adresářová struktura Symfony projektu ... 23
4.1.4 Doctrine 2 ... 24
4.1.5 Modelová situace ... 25
4.1.6 Kompletní projekt ... 26
4.2 Implementace MySQL ... 27
4.2.1 Konfigurace ... 27

4.2.2 Implementace schématu 28
4.2.3 Práce s daty 33
4.3 Implementace MongoDB 35
4.3.1 Konfigurace 35
4.3.2 Definice dokumentů 37
4.3.3 Práce s daty 39
4.4 Srovnání implementace obou řešení 41
4.5 Srovnání výkonu obou řešení 42
4.5.1 Generování testovacích dat 42
4.5.2 Monitorování výkonu 44
4.5.3 Vybrané dotazy 45
4.5.4 Výsledky testu 46
4.5.5 Odvozené závěry 47

5 Závěr 49

Terminologický slovník 51

Seznam literatury 53

Seznam obrázků, tabulek a výpisů 56
Seznam obrázků 56
Seznam tabulek 56
Seznam výpisů 56

Přílohy 58

1 Úvod

Při vývoji webových aplikací musí jejich architekt řešit otázku, jaké použít technologie. Jednou z nejběžnějších situací je volba Linuxového serveru s web serverem Apache2.[1] Nejpoužívanější je pak programovací jazyk PHP.[2] V dalším kroku je potřeba zvolit pro aplikaci správný framework, ve kterém bude vyvíjena, a systém pro správu dat. Pro účely této práce byl zvolen PHP framework Symfony2[3] (dále jen Symfony) a výběrem databázového systému se bude zabývat celý tento dokument.

1.1 Cíl práce

Cílem této práce je porovnat dva databázové systémy: MySQL a MongoDB. Důvod, proč byly vybrány právě tyto dva, je ten, že Symfony je standardně distribuována s mapovacím frameworkem Doctrine2 (dále jen Doctrine), a ten zatím oficiálně podporuje pouze zmíněné dva systémy. Ačkoliv lze v Symfony implementovat široká škála databázových systémů, je u těch ostatních (jiných než MySQL a MongoDB) obvykle potřeba spoléhat se na knihovny s nejistým budoucím vývojem a údržbou. Nasazení MySQL nebo MongoDB v Symfony je tedy spolehlivější řešení.

V první a druhé části práce bude vysvětleno teoretické pozadí obou databází do takové míry, která umožňuje pochopení rozdílu mezi nimi a orientaci v problematice v praktické části této práce. V první části budou obecně charakterizovány relační databáze, konkrétně MySQL, a budou vysvětleny výhody úložného formátu InnoDB. Následně budou ve druhé části podobným způsobem představeny NoSQL databáze. Větší pozornost bude věnována představiteli dokumentových databází – MongoDB.

---

[1] Operační systém z rodiny Unix systémů dnes využívá 68 % všech webových aplikací. [1] Z nich více než polovina je Linux [2] a na přibližně 55 % systémů s Linuxem se používá web server Apache2. [3]

[2] Když Google na konferenci v květnu 2013 oznámil podporu pro PHP v jejich cloudové službě Google App Engine, výrazně tím napomohl k dalšímu růstu tržního podílu PHP na dnešních přibližně 82 % všech webových aplikací. [4]

[3] Symfony2 je druhý (hned po Laravelu) nejpoužívanější PHP framework na světě. [5]

V třetí části bude nejprve popsáno testovací prostředí. V zápětí bude navržena modelová situace a pro ni budou v testovacím prostředí implementovány oba porovnávané databázové systémy. Obě implementace budou porovnány především z hlediska pohodlnosti pro vývojáře.

V závěru práce budou obě řešení porovnána z hlediska výkonu při vykonávání vybraných dotazů. Rozhodujícím kritériem při tomto porovnání bude rychlost zpracování dotazu.

1.2 Rešerše podobně zaměřených prací

Před vypracováním této práce byly prostudovány jiné práce na příbuzné téma. Přestože nebyla nalezena práce, která by řešila stejný problém jako tato, bylo nalezeno několik prací s blízkým zaměřením:

Důvody k nasazení NoSQL, a speciálně MongoDB (Vysoká škola ekonomická v Praze, 2015) – V této bakalářské práci její autor Miroslav Čermák obecně popisuje NoSQL databáze a následně se věnuje systému MongoDB. Součástí práce je také případová studie některých implementovaných řešení a z ní odvozené důvody pro nasazení MongoDB. Práce je zpracována kvalitně, ale výkonu systému MongoDB v porovnání s ostatními systémy se v ní autor věnuje jen povrchně. [6]

Tvorba webových aplikací pomocí frameworku Symfony (Vysoká škola ekonomická v Praze, 2015) – Jan Dočkál ve své bakalářské práci podává návod, jak vytvořit jednoduchou aplikaci ve frameworku Symfony2. Součástí práce je postup při implementaci MySQL databáze, což je součástí také této práce. [7]

NoSQL databáze (Vysoká škola ekonomická v Praze, 2014) – Autor Jakub Mrozek v této bakalářské práci velmi pečlivě vysvětlil, jak fungují NoSQL databáze. Podrobně se věnuje všem jejich typům a nemálo prostoru je věnováno i databázi MongoDB. Práce je teoretického charakteru. [8]

MongoDB (Vysoká škola ekonomická v Praze, 2013) – Bakalářská práce Martina Petery, která velmi podrobně popisuje MongoDB. Práce nejprve vysvětluje principy NoSQL databázi, podrobněji se věnuje dokumentovým databázím a právě MongoDB. Dále autor

uvádí důvody proč použít/nepoužít právě MongoDB. Součástí práce je také praktická ukázka použití databáze. [9]

2 MySQL

V této části budou nejprve představeny relační databáze. Blíže bude popsán systém MySQL a jeden z jeho úložných formátů – InnoDB. Ten bude později využit v praktické části této práce.

2.1 Principy relačních databází

Autorem relačního modelu je Edgar Frank Codd, který jej představuje ve svém pojednání z roku 1970. Během následujících let bylo v IBM založeno několik projektů. Výsledkem jednoho z nich byl dotazovací jazyk Sequel a z něj později vznikl jazyk SQL (Structured Query Language), který se používá dodnes.

Struktura relační databáze vždy odpovídá nějakému relačnímu modelu. Datový model obecně lze chápat jako souhrn nástrojů, které slouží k popisu dat, vazeb mezi nimi, jejich významu a omezení, zajištující jejich konzistenci. Data a jejich vazby v relačním modelu jsou znázorněny pomocí tabulek (relací).[10]

2.1.1 Vysvětlení základních pojmů

Jeden z běžných scénářů, na kterých se dají vysvětlit principy relačních databází je popis prostředí školy. Ve škole existují různé **entity** (např. učitelé, studenti, vyučované předměty, učebny,…). Entity, které jsou si svojí podstatou blízké, se dají dobře sdružovat do tabulek. Tedy např. všechny učitele je možné sdružit do tabulky s názvem Učitel. Pro každou entitu lze definovat výčet vlastností, které je potřeba v popisované realitě (tedy ve škole) zaznamenat v databázi. U učitelů může být zajímavé třeba jejich jméno, titul, plat a pohlaví. Těmto vlastnostem se říká **atributy** a v tabulce se projeví jako názvy sloupců. Konkrétní učitelé pak budou představovat jednotlivé řádky v tabulce Učitel. Jak by taková tabulka mohla vypadat je znázorněno v tabulce *Tabulka 1*.

V tabulce si lze všimnout ještě jednoho dalšího atributu – ID. Každá entita musí mít v rámci své tabulky jednoznačný unikátní identifikátor – **primární klíč**. U některých entit lze nalézt tzv. **přirozený primární klíč**. U učitelů by to mohlo být rodné číslo.

V popisované realitě však v tabulce Učitel atribut "rodné číslo" není a u žádného jiného atributu (nebo skupiny atributů) nelze zajistit, že bude pro každou entitu této tabulky unikátní. To je důvod, proč bylo potřeba vytvořit **umělý primární klíč** – atribut ID. [10]

Tabulka 1: *Učitel (zdroj: autor)*

ID	jméno	titul	plat	pohlaví
1	Dan Myška	Ing.	28000	muž
2	Jan Rychlý	*NULL*	22000	muž
3	Milan Pomalý	RNDr., Ph.D.	27500	muž

Dalším pojmem, který je potřeba zmínit, je **doména**. Doména je množina všech hodnot, kterých může daný atribut nabývat. Prvky domény lze považovat za **atomické**, pokud jejich hodnoty již nelze dále rozdělit tak, aby dávaly smysl. Ve vzorové tabulce Učitel třeba doména atributu *jméno* není atomická, protože by se dala rozdělit na křestní jméno a příjmení (a obě jména by tedy byla ve vlastním sloupci). Atomicita je podmínkou splnění jedné z normálních forem, které budou popsány později v této kapitole.[10] Za povšimnutí ještě stojí klíčové slovo NULL, které se vyskytuje u druhého záznamu ve sloupci titul. Hodnota tohoto atributu není v této tabulce povinná a NULL vyjadřuje, že tato hodnota u tohoto záznamu nebyla zadána.

2.1.2 Asociace

Mezi entitami v relačním modelu databáze často vznikají asociace (používají se také termíny *vztah* nebo *vazba*). Na příkladu se školním prostředím by to mohla být třeba vazba mezi učitelem a třídou, ve které je tento učitel třídním. Každá třída má přiřazeného právě jednoho učitele jako třídního. Tato situace je znázorněna v tabulce *Tabulka 2*. Atribut *třídní_učitel* v tabulce *Třída,* je **cizí klíč** odkazující na primární klíč *ID* v tabulce *Učitel*. Tímto způsobem je vytvořena vazba 1–1 mezi entitami učitelů a tříd, kde platí, že každý učitel je třídním učitelem jedné nebo žádné třídy. Třeba učitel Jan Rychlý není třídním učitelem žádné třídy, zatímco Milan Pomalý je třídním učitelem ve třídě 1. A.

Tabulka 2*: Třída (zdroj: autor)*

ID	jméno	třídní_učitel
1	1. A	3
2	1. B	1

Tato vazba by se dala rovněž realizovat tím způsobem, že v tabulce Učitel by se nacházel cizí klíč "třída", odkazující na tabulku Třída. Zatímco všechny třídy musí mít právě jednoho třídního učitele, ne každý učitel musí být v nějaké třídě třídním. Je tedy vhodnější umístit cizí klíč způsobem naznačeným v tabulce. Zároveň u této vazby platí, že dvě různé třídy nesmí mít stejného třídního učitele.

Dalším možným (a zároveň nejběžnějším) vztahem mezi entitami je vazba *1–n*. V popisovaném prostředí to může být například přiřazenost studenta ke třídě. Platí, že v jedné třídě může být *n* studentů a každý je vždy přiřazen k právě jedné třídě. Jak by mohla vypadat tabulka Student, je znázorněno v tabulce *Tabulka 3*. Na rozdíl od vazby *1–1*, v tomto případě více entit relace Student může mít cizí klíč odkazující na stejnou třídu, jako je tomu například u studentů Zeleného, Modrého a Hranaté.

***Tabulka 3**: Student (zdroj: autor)*

ID	jméno	třída
1	Pavel Zelený	1
2	Martin Modrý	1
3	Eliška Lišková	2
4	Eva Hranatá	1

Posledním možným typem asociace je vazba *m–n*. V prostředí školy lze takovou vazbu nalézt mezi učiteli a vyučovanými předměty. Každý učitel může vyučovat více předmětů a každý předmět může být vyučován více učiteli. Tento typ vazby je realizován pomocí vazební tabulky, kterou lze nazvat například *Vyučuje*, což reflektuje význam vazby.[4] Použitím vazební tabulky je vazba *m–n* ve skutečnosti reprezentována jako dvě vazby *1–n*. Vazba je znázorněna v tabulkách *Tabulka 4* a *Tabulka 5*. Jedna z vazeb je mezi relacemi *Učitel* a *Vyučuje*, a ta druhá mezi relacemi *Předmět* a *Vyučuje*. Na rozdíl od ostatních tabulek, v tabulce *Vyučuje* není potřeba zvlášť sloupec pro primární klíč, ten je v této tabulce složený z cizích

***Tabulka 4**: Vyučuje (zdroj: autor)*

učitel	předmět
1	**1**
1	**3**
2	**1**
3	**2**

***Tabulka 5**: Předmět (zdroj: autor)*

ID	jméno
1	Matematika
2	Angličtina
3	Tělocvik

[4] Jiným vhodným názvem tabulky by byl *UčitelPředmět*. Z tohoto způsobu pojmenování je ihned patrné, mezi kterými tabulkami vazba existuje.

klíčů *učitel* a *předmět*. I v tomto případě musí být zachována unikátnost primárního klíče a nemohou tedy existovat dva řádky, které by měly stejnou kombinaci hodnot u atributů *učitel* a *předmět*.

2.1.3 Index

Index lze chápat jako ukazatele na záznam, který obsahuje konkrétní hodnotu určitého atributu (nebo skupiny atributů). Pokud by aplikace nad databází školy vyhledávala učitele podle titulu, bylo by vhodné vytvořit index pro atribut *titul*. Databáze by si tímto vytvořila rejstřík, ve kterém by pro všechny použité tituly měla přiřazené odkazy na řádky v tabulce *Učitel*, kde se tato hodnota vyskytuje. Při vyhledávání podle titulu potom není potřeba číst celý záznam, ale stačí prohledat tabulku indexů. U indexů je potřeba pamatovat na to, že zrychlují vyhledávání nad tabulkou, ale za cenu zpomalení vytváření, úprav a mazání záznamů kvůli režii s indexy spojené, protože při tomto typu operací musí být systémem vždy aktualizován také indexový rejstřík. [10]

2.1.4 ACID

Zkratka ACID je tvořena čtyřmi pojmy: atomicity, consistency, isolation a durability. Je to skupina vlastností, které by relační databáze měla splňovat, mají-li být data v ní uložená spolehlivá. [10]

- Atomicity (atomicita) je vlastnost, která zaručuje, že se určitá skupina příkazů úspěšně vykoná celá, nebo se nevykoná žádný (v takovém případě dojde k takzvanému *rollbacku*, kdy se databáze vrátí do stavu, jenž byl zaznamenán na začátku transakce, ještě než byly provedeny jakékoli změny). Důležitost atomicity transakcí lze předvést na příkladu bankovních účtu. Existuje předpoklad, že jsou dva účty, A (na kterém je 1 000 Kč) a B (na kterém jsou 2 000 Kč). Je potřeba převést 100 Kč z účtu A na účet B. Nejprve je z účtu A odečteno 100 Kč, ale při pokusu o přičtení 100 Kč na účet B dojde v systému k chybě. Pokud tato operace neprobíhala v transakci, bude nyní stav na účtech 900 Kč (A) a 2 000 Kč (B), čímž „zmizelo“ 100 Kč. Pokud tato operace v transakci probíhala, bude při chybě účet A navrácen do stavu před transakcí, což je žádoucí.

- Consistency (konzistence) zaručuje zachování integrity dat po ukončení transakce. Ukončení transakce nastane, když je proveden buďto *commit* (tedy potvrzení změn) nebo *rollback*.

- Isolation (izolace) – tato vlastnost zabraňuje vzniku nekonzistentních stavů, pokud by se naráz vykonávaly různé transakce. Vlastnost konzistence totiž zaručuje konzistenci dat na konci transakce, nikoliv však během jejího vykonávání. Pokud by během vzorové transakce s bankovními účty, popsané u jednoho z předchozích bodů, druhá transakce současně pracovala se stejnými daty, tedy například by se snažila zjistit součet částek na účtech A a B, v okamžiku, kdy první transakce odečetla částku z účtu A, ale dosud ji nepřičetla na účet B, rovnal by se součet částce 2 900 Kč. Tato data jsou však nekonzistentní, protože po skončení první transakce bude celková částka, stejně jako před jejím začátkem, 3 000 Kč. Z toho tedy plyne, že transakce pracující se stejnými daty nesmí běžet současně.

- Durability (trvanlivost) – pro platnost této vlastnosti musejí být splněny dvě podmínky:

    - Před ukončením transakce jsou již jakékoliv změny trvalé, tedy zapsány na disk.

    - Jsou uchovány všechny potřebné informace o transakci, aby bylo možné databázi načíst ve stavu po transakci, pokud by bylo systém potřeba restartovat.

U některých databází není spolehlivost dat prioritou a naopak je kladen důraz na rychlost zpracování. Takové databáze nesplňují ACID vlastnosti a zajištění konzistence dat musí být prováděno na aplikační vrstvě. [10]

2.1.5 Normální formy

Normalizace je metoda úpravy schématu takovým způsobem, aby bylo zamezeno zejména redundanci dat. Stupeň normalizace lze vyjádřit tím, kolik normálních forem je splněno. Normálních forem existuje více a při návrhu schématu by obvykle měly být splněny alespoň první tři normální formy:

1. 1NF (první normální forma) je splněna, pokud domény všech atributů jsou atomické.
2. 2NF (druhá normální forma) je splněna, pokud každý neklíčový atribut je přímo závislý na celém primárním klíči (v případě složeného primárního klíče není závislý pouze na jeho části). Zároveň musí být splněna 1NF.
3. 3NF (třetí normální forma) je splněna, pokud jsou všechny neklíčové atributy navzájem nezávislé. Zároveň musí být splněna 2NF.

Někdy je normalizované schéma úmyslně upraveno do nenormalizované podoby. Tomuto procesu se říká denormalizace a obvykle se provádí v rámci optimalizace výkonu databáze na úkor splnění normálních forem. [10]

2.2 Charakteristika MySQL

MySQL je systém řízení relační databáze neboli RDBMS (z angl. Relational Database Management System). Jedná se o druhý nejpoužívanější RDBMS na světě. Tím prvním je systém Oracle. Na rozdíl od Oraclu je však MySQL open-source.[11]

2.2.1 Vznik a vývoj

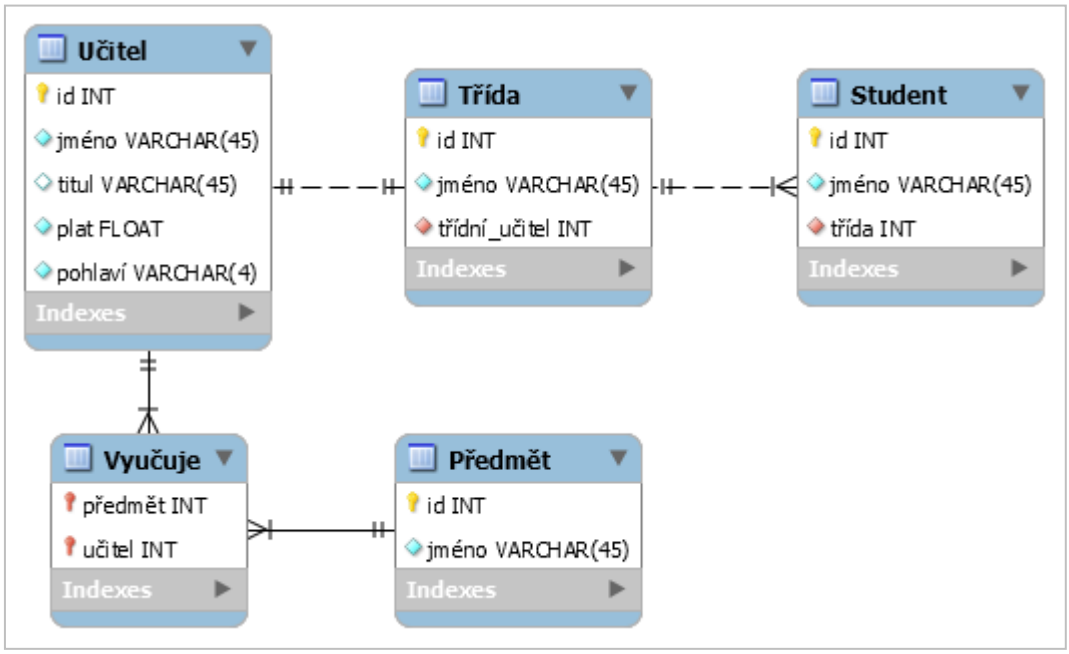
Projekt MySQL vznikl spolu s firmou MySQL AB v roce 1995. Firmu založili Švédové David Axmark, Allan Larsson a Fin Michael Widenius. Název MySQL je složen ze jména dcery Michaela Widenia, My, a zkratky pro Structured Query Language, což je jazyk vytvořený pro práci s daty v relačních databázích.[12]

V roce 2000 byl systém MySQL vydán jako open-source pod licencí GNU General Public License (všeobecná veřejná licence GNU, dále jen GPL). V roce 2008 byla firma MySQL AB odkoupena americkou firmou Sun Microsystems přibližně za miliardu dolarů. Roku 2009 pak byl celý Sun odkoupen firmou Oracle za 7,4 miliard dolarů.[13]

2.2.2 Práce se systémem

Při práci s MySQL databází lze pro definici struktury databáze i práci s daty použít přímo jazyk SQL. Další možností je použít nějaký nástroj pro tyto účely určený. Oracle vyvinul open-source nástroj MySQL Workbench, který lze použít k návrhu databáze, její správě a monitorování. V nástroji lze snadno vytvářet Entity Relationship Diagramy (dále jen ERD), tedy diagramy vazeb mezi entitami. Na obrázku 1 – ERD školy, je vidět struktura tabulek popsaných v předchozí podkapitole 2.1.1, včetně asociací mezi nimi. Tento diagram byl vytvořen právě pomocí nástroje MySQL Workbench verze 6.3.[5]

V diagramu jsou jednotlivé tabulky. U každé z tabulek je na jednom řádku znázorněn jeden atribut. Na prvním místě řádku je speciální znak. Znak klíče znázorňuje primární klíč,



Obrázek 1: *ER diagram prostředí školy (zdroj: autor)*

červený kosočtverec znamená cizí klíč a modrý kosočtverec zastává ostatní atributy (pokud je kosočtverec plný, je atribut povinný, jinak může nabývat hodnoty NULL). Za znakem následují jméno atributu a jeho datový typ. Přehled v práci použitých datových typů a jejich význam je součástí přílohy (Příloha A). Pro znázornění vazeb mezi tabulkami je

[5] MySQL Workbench je volně dostupný ke stažení z oficiálních stránek na webové adrese: http://www.mysql.com/products/workbench/

použita Barkerova notace (také známá jako notace Crow foot neboli „vraní noha“)[6]. Podobný diagram bude použit také v druhé části této práce.

2.2.3 Úložné formáty

Mezi nejsilnější stránky MySQL patří rychlost, škálovatelnost a jednoduchost implementace. MySQL také podporuje několik úložných formátů, které umožňují přizpůsobit databázi jejímu konkrétnímu použití. Například formát Memory umožňuje vytvářet dočasné tabulky, které existují pouze v operační paměti. [14]

Nejpoužívanější jsou formáty MyISAM a InnoDB. MyISAM je velmi rychlý a výhodný u tabulek, ze kterých se data převážně čtou, a málokdy upravují. Nejuniverzálnější je však InnoDB a právě ten má být v budoucnu nejvíce zlepšován a udržován. Jak je ze zápisu změn pro nejnovější verzi MySQL 5.7 vidět, byly do InnoDB implementovány některé vlastností, které dosud podporoval pouze systém MyISAM. [15]

Velkou výhodou InnoDB je podpora transakčního zpracování. To je nezbytné, mají-li být v databázi splněny tzv. ACID principy popsány v podkapitole 2.1.4. [14]

Jinou výhodou InnoDB oproti MyISAM je podpora integritních omezení ve formě cizích klíčů. Tím lze zaručit konzistenci dat a nemůže tedy dojít k situaci, kdy by cizí klíč odkazoval na neexistující záznam v jiné tabulce. [14]

V MySQL platí, že úložný formát lze definovat pro každou tabulku jiný, což je silný nástroj při optimalizaci výkonu databáze. Nad tabulkou, která je zřídka kdy upravována, ale často se z ní data čtou, může být výhodné nastavit formát MyISAM. Nad tabulkou, kde je kritická integrita dat, je naopak lepší nastavit formát InnoDB. Formát tabulky lze později změnit, pokud cílový formát podporuje všechny rysy současného formátu, které jsou zrovna na tabulce používané (jinak je potřeba strukturu tabulky nejprve patřičně upravit).[14]

---

[6] Barkerova notace je vysvětlena v publikaci: BARKER, Richard. *Case*Method: Entity Relationship Modelling*. 1990. ISBN 0201416964.

3 MongoDB

Podobným způsobem jako v první části se práce bude nejprve věnovat teoretickému zázemí NoSQL databází. Budou popsány jejich jednotlivé druhy. Dokumentová databáze MongoDB bude popsána na konci této části.

3.1 Principy NoSQL databází

3.1.1 Vznik a vývoj

Název NoSQL složený z částí No a SQL na první pohled vzbuzuje dojem, že se jedná o jakýsi protějšek k jazyku SQL. Jedná se však spíše o alternativu k relačním databázím popsaným v předchozí kapitole. Některé NoSQL databáze totiž mohou podporovat dotazovací jazyky podobné SQL. Z tohoto důvodu se přešlo k interpretaci názvu jako "Not only SQL" tedy v překladu „Nejen SQL“. [16]

Přesný okamžik vzniku NoSQL databází je těžké vytyčit. NoSQL databáze se začaly převážně objevovat v období, kdy se existující relační databáze začaly potýkat s velkým množstvím dat (občas se používá pojem Big Data[7]). Jak se ukázalo, relační model začíná mít výkonnostní problémy v systémech s velkým množstvím dat, s jakými pracuje třeba Google. Jiným aspektem je třeba to, že relační databáze silně spoléhají na dobrou strukturovanost dat. Dříve se předpokládalo, že data budou mít po celou dobu stejnou strukturu, jakou měla definovanou při návrhu modelu pro konkrétní databázi. Jakmile se však ukázalo, že data jsou stále různorodější, bylo potřeba upravovat model databáze (opuštění některých integritních omezení, uvolnění důrazu na ACID vlastnosti, denormalizace dat), čímž se model začal vzdalovat relačnímu pojetí. V tomto období v historii se začaly objevovat NoSQL databázové systémy s různými vlastnostmi. Dnes

[7] Dnes se za Big Data označuje množství dat v řádu petabytů (10^{15} bytů). Úložné kapacity a velikost ukládaných dat však s technologickým pokrokem rychle rostou. Jedna z možných definic tohoto pojmu je, že Big Data jsou takové množství dat, se kterým se nedá pracovat v rozumném čase.[17]

jich existuje početné množství a na rozdíl od relačních databází o nich mnohdy nelze říci, že by měly mnoho společného.[16]

3.1.2 Kategorizace

NoSQL databáze se dají kategorizovat různě. V pracích, zabývajícími se NoSQL databázemi, se lze nejčastěji setkat s rozdělením Bena Scofielda. Pro účely této práce je toto rozdělení rovněž dostačující:

- úložiště klíč – hodnota,
- sloupcová úložiště,
- dokumentové databáze,
- grafové databáze.

Některé NoSQL databáze nelze jednoznačně zařadit do jedné z uvedených kategorií. Třeba s databází OrientDB je možné pracovat jako s dokumentovou i jako grafovou databází. [18]

3.1.3 Úložiště klíč – hodnota

Jedná se o velmi jednoduchý datový model. Ten odpovídá asociativnímu poli[8], tedy jedná se o páry unikátních klíčů a hodnot. Tento typ úložiště se vzdává možnosti tvorby složitějších dotazů, jako jsou JOIN operace nebo agregační funkce. Soustřeďuje se zejména na rychlost a škálovatelnost. V úložištích klíč–hodnota neexistuje schéma, díky čemuž je potřeba menší úložné kapacity. Je využito méně operační paměti a operace jsou velmi rychlé. V hodnotě se může nacházet prakticky cokoliv – jednoduchý text, JSON, XML i obrázek. [19]

Tabulka 6: *Ukázka úložiště klíč–hodnota (zdroj: autor)*

klíč	hodnota
Učitel/1/jméno	Dan Myška
Učitel/1/plat	28000
Učitel/2/jméno	Jan Rychlý

[8] V některých programovacích jazycích se používá termín *slovník* nebo *mapa*.

Mezi zástupce, které je vhodné zmínit, patří třeba oblíbený Redis (ten spoléhá na ukládání dat v operační paměti). Jiným zajímavým zástupcem je Oracle NoSQL Database. Ten ukládá klíč složený z hlavní části a podklíčů (tento způsob připomíná adresářovou cestu) a tím podporuje širší možnosti dotazování. Jak by takové úložiště mohlo vypadat, je naznačeno v tabulce *Ukázka úložiště klíč–hodnota*. Data odpovídají prostředí školy popsanému v předchozí kapitole. [20]

3.1.4 Sloupcová úložiště

Sloupcové databáze jsou podobné těm relačním v tom, že ukládají data do tabulek. Pokud bude uvažována ukázka s tabulkou Učitel v první kapitole této části (*Tabulka 1*), byla by data v relační databázi na disk zapsána přibližně v této podobě (první číslice před dvojtečkou je číslo řádku v tabulce):

***Výpis 1**: Příklad uložení dat v relační databázi (zdroj: autor).*

```
001:1,Dan Myška,Ing.,28000,muž;
002:2,Jan Rychlý,,22000,muž;
003:3,Milan Pomalý,"RNDr.,Ph.D.",27500,muž;
```

Ve sloupcové databázi by byly jednotlivé záznamy reprezentovány takto:

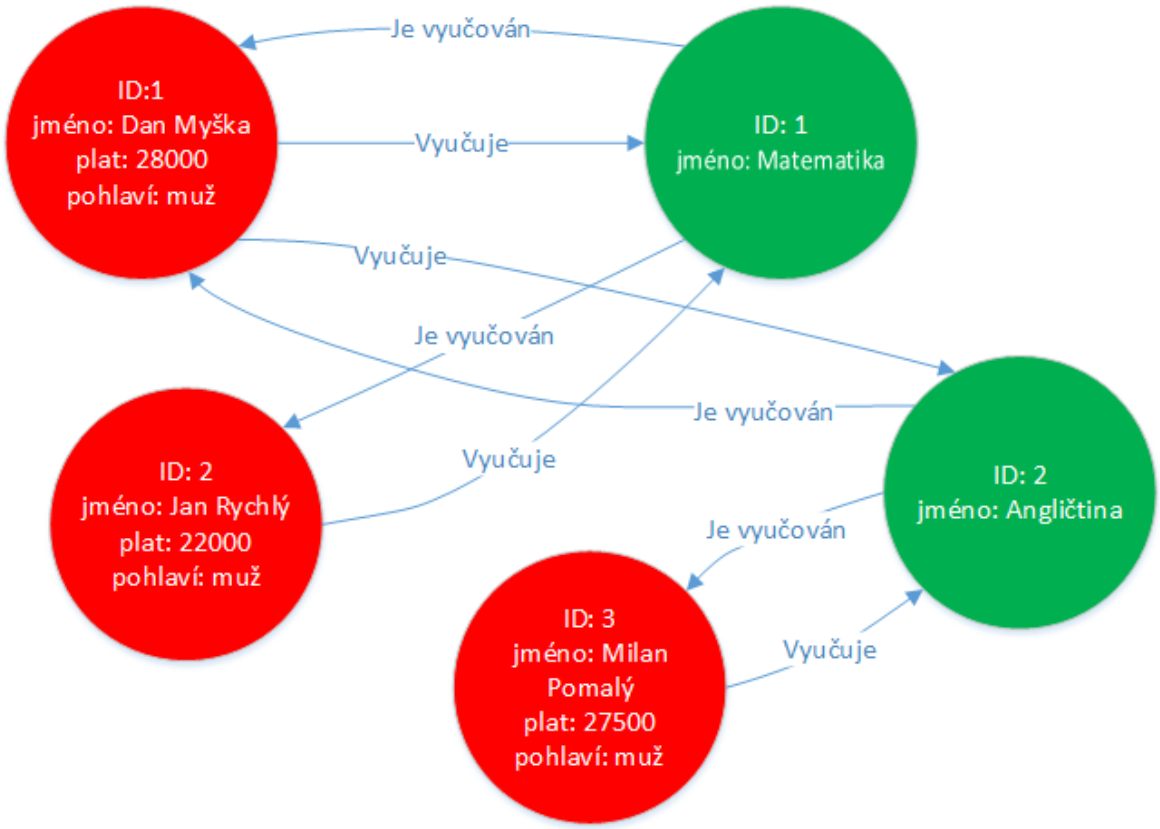
***Výpis 2**: Příklad uložení dat ve sloupcové databázi (zdroj: autor).*

```
1:001,2:002,3:003;
Dan Myška:001,Jan Rychlý:002,Milan Pomalý:003;
Ing.:001,"RNDr.,Ph.D.":003;
28000:001,22000:002,27500:003;
muž:001,002,003;
```

Tento přístup má za následek, že ve výsledku je využito méně úložného prostoru, protože záznamy s NULL hodnotou pro některý atribut jednoduše nebudou mít pro daný atribut ve sloupcové databázi záznam. Zároveň záznamy, které mají hodnotu nějakého atributu stejnou, mohou být konsolidovány (viz poslední řádek z *Výpisu 2*). A zatímco u relační databáze je potřeba číst celý řádek, u sloupcové stačí procházet pouze ty sloupce, podle kterých je vyhledáváno, a které mají být součástí vráceného výsledku.[21]

3.1.5 Grafové databáze

Existují situace, ve kterých jsou grafové databáze lepší než jakýkoli jiný typ databází. Obvykle se jedná o případy, kdy existuje mnoho vazeb mezi entitami (např. sociální síť). V grafové databázi jsou jednotlivé entity reprezentovány jako uzly grafu. Vztahy mezi entitami odpovídají hranám mezi uzly. Jak uzly, tak hrany mohou mít atributy, podobně jako v relačních databázích. Na obrázku je znázorněno několik entit, a vazeb mezi nimi, z příkladu prostředí školy z první kapitoly této části.



Obrázek 2: *Graf hran (modře) mezi uzly učitelů (červeně) a předmětů (zeleně). Graf byl vytvořen pomocí nástroje Microsoft Visio 2013 (zdroj: autor).*

Grafové databáze jsou vhodné pro některé specifické úkoly, jako je hledání nejkratší cesty v grafu nebo počítání Page Ranku[9]. Taková databáze by tedy byla ideální například pro implementaci navigačního systému v dopravní síti. [18]

[9] Page Rank je identifikátor důležitosti webových stránek počítaný speciálním algoritmem. Využívá jej například Google. [22]

3.1.6 Dokumentové databáze

Dokumentové databáze se dají považovat za komplexnější variantu databází typu klíč–hodnota, protože každý dokument je tvořen skupinou párů klíč–hodnota. Dokumenty mohou být různých formátů, nejčastější jsou JSON, YAML a XML.[19]

V následující ukázce jsou dokumenty obsahující údaje o dvou učitelích z kapitoly o relačních databázích. Za povšimnutí stojí, že u druhého dokumentu chybí pole "titul". V dokumentových databázích není pevně dané schéma a dokumenty se tak mohou svojí strukturou lišit. V tomto případě je použit formát JSON.

***Výpis 3**: Dokumentová databáze ve formátu JSON (zdroj: autor).*

```
{
  "typ":"Učitel",
  "id":1,
  "jméno":"Dan Myška",
  "titul":"Ing.",
  "plat":28000,
  "pohlaví":"muž"
}
```

```
{
  "typ":"Učitel",
  "id":2,
  "jméno":"Jan Rychlý",
  "plat":22000,
  "pohlaví":"muž"
}
```

Některé dokumentové databáze jako CouchDB nebo MongoDB umožňují v dokumentech použít odkaz na jiný dokument a tím vytvářet asociace. Tím se dokumentové databáze přibližují těm relačním. Jejich podstatná výhoda je však v absenci schématu, díku čemuž je vyhledávání v dokumentové databázi triviální záležitostí. Dokumentové databáze rovněž podporují indexování, takže není potřeba vždy číst celé dokumenty. [18]

3.2 MongoDB

3.2.1 Vznik a vývoj

MongoDB je poměrně mladý databázový systém. Uprostřed roku 2007 startupová společnost 10gen zahájila vývoj platform-as-a-service projektu, který měl sloužit pro hostování webových aplikací. Myšlenka byla taková, že vývojáři si navolí technologické požadavky své aplikace (jak hardwarové, tak softwarové) a soustředí se pouze na aplikační kód. Vývojáři se ale nechtěli v takové míře vzdát kontroly nad používanými technologiemi, což nakonec vedlo k opuštění projektu. Měli však zájem o nové technologie v oblasti správy dat, se kterým v 10gen přišli. Společnost tak opustila projekt a soustředila se čistě na jejich databázový systém, který pojmenovali MongoDB. Z MongoDB se stal open-source, i když byl nadále sponzorován ze strany 10gen. Jeho původní zamýšlené použití (napříč mnoha zařízeními) mu zanechalo důležitou výhodu a tou je jednoduchá škálovatelnost. Potřeby jeho uživatelů jej pak nasměrovaly tím způsobem, že kombinuje výhody jak relačních databázových systémů, tak úložišť typu klíč–hodnota. [23]

3.2.2 Dokumentový datový model

Základní jednotkou v MongoDB databázi je dokument. Ten svojí strukturou odpovídá skupině (minimálně jednomu) párů klíč–hodnota a na disku je uložen ve formátu BSON (binárně zakódovaný formát JSON). Dokumenty jsou v MongoDB sdružovány do kolekcí. Kolekci lze v dokumentovém datovém modelu chápat jako relaci v relačním modelu. Jednotlivé dokumenty v kolekci pak odpovídají jednotlivým záznamům v relaci, tedy řádkům v tabulce. Výhodou MongoDB je, že i bez použití schématu umí s dokumenty zacházet podobně, jako zacházejí relační databáze s entitami. Vytváření dotazů je pak velmi jednoduché a díky kolekcím zároveň není potřeba definovat typ dokumentu přímo uvnitř dokumentu, jak je to ve výpise 3 předchozí kapitoly. Ve výsledku se dají vytvářet podobné dotazy, jako umožňuje jazyk SQL v relačních databázích. [23] Pokud by bylo

potřeba například nalézt dokumenty učitelů, pro které platí, že jsou to muži a jejich plat je větší než 25 000, vypadal by dotaz takto[10]:

```
db.učitel.find( { pohlaví:'muž', plat: {$gt: 22000} } )
```

V dotazu „db“ označuje příkaz volaný nad právě vybranou databází, protože jich může být i více. Následuje jméno kolekce, tedy „učitel“. Klíčové slovo „find” identifikuje operaci, která je vykonávána. V SQL jazyce je mu nejblíže operace SELECT. Následují parametry operace, kde „$gt“ je volání MongoDB funkce „greater then” neboli větší než. [24]

MongoDB při vytváření dokumentů každému dokumentu přiřadí unikátní identifikátor v rámci kolekce. Fungují podobně jako primární klíč v relačním modelu a uživatel je může specifikovat. Identifikátor dokumentu je vždy hodnota pod klíčem „_id“ jako je to v následující ukázce dokumentu předmětu:[25]

Výpis 4: *Identifikátor v dokumentu (zdroj: autor).*

```
{
  "_id":1,
  "jméno": Matematika
}
```

```
{
  "_id": ObjectId("53d98f133bb604791249ca99"),
  "jméno": Matematika
}
```

Druhá ukázka ve výpisu je stejný dokument, kde však nebylo _id specifikováno, ale bylo přiděleno systémem automaticky. Toto automaticky vygenerované ID je 24 míst dlouhé hexadecimální číslo. To v sobě obsahuje Unixovou časovou značku, identifikátor zařízení, procesu a pořadové číslo. [26]

3.2.3 Asociace

Podobně jako existují cizí klíče u relačních databází, lze i v MongoDB vytvářet mezi dokumenty vztahy 1–1 a 1–n. Přitom existuje několik způsobů, jak v MongoDB takovou

[10] Ukázky volání příkazu nad MongoDB jsou v prostředí *Mongo shell*, což je interaktivní javascriptové rozhraní spustitelné z příkazové řádky. Jak s tímto rozhraním zacházet je popsáno v oficiální dokumentaci MongoDB a je mimo zaměření této práce. [27]

vazbu realizovat. Tím prvním je odkaz na jiný dokument podle *_id*, jak je to znázorněno v následující ukázce:

Výpis 5: *Vazba 1–n pomocí odkazů (zdroj: autor).*

```
{
  "_id": ObjectId("53d98f133bb604791249ca99"),
  "jméno": "1. A",
  "třídní_učitel": "1"
}
```

```
{
  "_id": "1",
  "jméno": "Pavel Zelený",
  "třída": ObjectId("53d98f133bb604791249ca99")
}
```

První dokument je z kolekce *Třída.* Tento dokument má automaticky vygenerované _id. Druhý dokument patří do kolekce *Student.* Nezáleží na tom, jestli je pro odkazování na jiný dokument použito vlastní *_id*, nebo *_id* vygenerované systémem. Následující ukázka znázorňuje dotaz, kterým lze nalézt všechny studenty, kteří patří do třídy 1. A. Je důležité poznamenat, že MongoDB nepodporuje JOIN operace jako SQL. Nejprve je tedy potřeba najít třídu podle jména, získat *_id* jejího dokumentu a podle něj následně najít studenty.[28]

Výpis 6: *Vyhledávání podle odkazovaného dokumentu (zdroj: autor).*

```
var trida = db.třída.findOne( { jméno:'1. A' } )
db.student.find( { "třída":trida["_id"] } )
```

Tímto způsobem je možné v MongoDB namodelovat asociace podobně jako v relačním modelu. Na rozdíl od relačního modelu však lze ze třídy odkazovat na jednotlivé studenty. Tento přístup je vhodný, pokud je v aplikaci potřeba často získat všechny studenty v některé třídě. Následuje ukázka dokumentu včetně příkazu, kterým lze nalézt všechny třídy, ve kterých se nachází student 1. Tento dotaz je od pohledu velmi jednoduchý.

[24] [28]

Výpis 7: *Odkazovaná kolekce (zdroj: autor).*

```
{
  "_id": ObjectId("53d98f133bb604791249ca99"),
  "jméno": "1. A",
  "třídní_učitel": "1",
  "studenti": [
      1,
      2,
      4
  ]
}
```

```
db.trida.find( { studenti: 1 })
```

Jiným způsobem, jak realizovat vazbu 1–n, je přímo vnořit dokumenty na straně *n* do dokumentu na straně *1*. Jak by mohl vypadat dokument třídy se všemi jejími studenty je v následující ukázce:

Výpis 8: *Vnořené dokumenty v MongoDB (zdroj: autor).*

```
{
  "_id": ObjectId("53d98f133bb604791249ca99"),
  "jméno": "1. A",
  "třídní_učitel": "1",
  "studenti": [
    {
      "jméno": "Pavel Zelený"
    },
    {
      "jméno": "Martin Modrý"
    },
    {
      "jméno": "Eva Hranatá"
    },
  ]
}
```

Tento způsob získání třídy se všemi studenty je výrazně rychlejší, protože všechny záznamy jsou vráceny jediným dotazem. Je však složitější nalézt konkrétního studenta. U jednotlivých vnořených studentů může a nemusí být identifikátor *_id* (MongoDB jej u vnořených dokumentů nevyžaduje a sám jej u nich ani nevytváří). Přítomnost *_id* však pravděpodobně vývojáři usnadní dotazování. [28]

4 Implementace databáze v modelové aplikaci

V této části práce budou implementovány obě představené databáze, MySQL a MongoDB, ve frameworku Symfony2. Nejprve budou popsány všechny kroky, které je potřeba uskutečnit před implementací obou databází. Bude představeno prostředí a všechny potřebné technologie. Po přípravě holé aplikace bude navržena modelová situace, pro kterou je potřeba realizovat databázové řešení. Nejprve bude nakonfigurována a implementována MySQL databáze, následně bude do stejné míry popsána implementace databáze MongoDB. V závěru této části budou obě řešení srovnána. Tato část práce předpokládá základní znalost jazyka PHP a objektového programování.

4.1 Popis prostředí

4.1.1 Platforma

Pro účely této práce byl k dispozici stolní počítač s operačním systémem Windows 7 Professional v 64-bitové verzi. Na zařízení byl k dispozici čtyřjádrový procesor Intel(R) Core(TM) i5-4440 s frekvencí 3.10 GHz a 8 GB operační paměti (RAM). Pro splnění předpokladů, definovaných v úvodu této práce bylo potřeba na tomto zařízení virtualizovat operační systém Linux. Pro tyto účely byl použit open-source nástroj firmy Oracle pro virtualizaci, VirtualBox.[11]

Pro vytvoření virtuálního stroje v nástroji VirtualBox byl využit open-source nástroj pro spravování virtuálních vývojových prostředí – Vagrant ve verzi 1.7.2.[12] Vagrant je produkt firmy HashiCorp a slouží k zjednodušení konfigurace vývojového prostředí a jeho jednoduchému sdílení s dalšími vývojáři. [29]

---

[11] Oracle VM VirtualBox je volně stažitelný z oficiálních stránek: https://www.virtualbox.org/

[12] Vagrant je k dispozici ke stažení ze svých oficiálních stránek: https://www.vagrantup.com/

Po instalaci VirtualBoxu a Vagrantu je potřeba ve Vagrantu nakonfigurovat, jaký systém si uživatel přeje virtualizovat. K tomu se používá soubor Vagrantfile. Opis souboru Vagrantfile, vytvořeného autorem této práce a použitého pro tuto ukázku, je součástí přílohy (příloha B). Z opisu je vidět, že pro tuto práci byl využit operační systém Linux – Debian 7 (řádek 6). V souboru je rovněž nastavena IP adresa (řádek 7), pod kterou bude aplikace lokálně přístupná z webového prohlížeče. Na řádku 9 je nastavení sdílených složek mezi hostujícím zařízením (Windows 7) a virtuálním strojem (Linux). Za zmínku ještě stojí řádky 16 a 17, na kterých je virtuálnímu stroji přidělena maximální operační paměť a počet jader procesoru. Nakonec na řádku 22 je cesta k souboru *init.sh*, v němž lze nadefinovat příkazy, jež se mají spustit ihned po vytvoření virtuálního zařízení. Soubor init.sh je vhodné místo pro definování technologií, které budou pro aplikaci potřeba, jako web server apache2, programovací jazyk PHP, databázové systémy a ostatní. Parametry souboru Vagrantfile lze kdykoliv změnit a stačí pak pouze restartovat virtuální stroj. [30]

4.1.2 Zprovoznění projektu

Repositář vytvořený autorem a použitý pro tuto práci je volně dostupný na úložišti Bitbucket od společnosti Atlassian. Pokud si čtenář přeje, může si celý projekt zprovoznit i na svém zařízení. Následující kroky jsou potřebné ke zprovoznění na zařízení s operačním systémem Windows:

1. Stáhnout a nainstalovat Oracle VM VirtualBox a Vagrant.
2. Stáhnout projekt z GIT repositáře na adrese:
   https://bitbucket.org/DominikFirla/reldoc/downloads
3. Z příkazové řádky se nanavigovat do kořenové složky staženého repositáře. Je to složka, která obsahuje soubory Vagrantfile, init.sh a složku /www.
4. V příkazové řádce zavolat příkaz: `vagrant up`

   Tímto se vybuduje a nainstaluje celý virtuální stroj, včetně všech závislostí v souboru init.sh. Po dokončení příkazu je virtuální stroj připraven k použití a lze na něm vytvořit nový projekt.
5. Do virtuálního stroje se lze nyní připojit přes SSH buďto příkazem `vagrant ssh` v příkazové řádce Windows (je potřeba volat ve složce se souborem Vagrantfile), anebo jakoukoliv aplikací, která umožňuje připojení přes SSH (například open-

source aplikace PuTTY). Výchozí přihlašovací jméno je *vagrant*, heslo je také *vagrant*.

6. Uvnitř virtuálního stroje, kam se uživatel dostal pomocí SSH, je nyní potřeba vytvořit nový projekt. Pokud uživatel neměnil nastavení sdílených složek ve *Vagrantfile*, je potřeba nanavigovat se do složky: /vagrant-www. Uvnitř této složky je nyní možné zavolat příkaz, který sám vytvoří základní Symfony projekt pomocí nástroje composer: [13]

```
composer create-project symfony/framework-standard-edition Projekt "2.8.0"
```

   Tento příkaz v aktuálním adresáři vytvoří složku *Projekt* (uživatel si může jméno projektu zvolit změnou příslušného řetězce v příkazu) obsahující základní aplikaci. Před koncem instalace bude uživatel vyzván k zadání hodnot některých parametrů, jako jsou údaje pro připojení do databáze nebo služba pro posílání e-mailů. Všechny parametry mají výchozí hodnotu a u každého z nich lze v tomto okamžiku výchozí hodnotu potvrdit klávesou *Enter*.

Tímto je aplikace připravena k použití na adrese 192.168.10.12/Projekt/web/.[14] Zbývající část této kapitoly bude předpokládat existenci nově vytvořeného projektu jménem Projekt.

4.1.3 Adresářová struktura Symfony projektu

Hned ze začátku je vhodné se v novém projektu zorientovat. Adresářová struktura Symfony v nově vytvořeném projektu odpovídá následujícímu schématu. Ve složce *app/* se nachází servisní záležitosti projektu, jako aplikační logy, konzole pro volání příkazů prostřednictvím SSH a hlavně konfigurace aplikace v podsložce *config/*. Do složky src/ patří všechny zdrojové kódy aplikace. Ve složce *vendor/* se nacházejí zdrojové kódy všech závislostí aplikace, které byly nainstalovány nástrojem *composer*. Obsah této složky by uživatel

Výpis 10: *Adresářová struktura v Symfony.[31]*

```
1 Projekt/
2 ├─ app/
3 │  ├─ console
4 │  ├─ cache/
5 │  ├─ config/
6 │  ├─ logs/
7 │  └─ Resources/
8 ├─ src/
9 │  └─ AppBundle/
10 ├─ vendor/
11 └─ web/
12    ├─ app.php
13    └─ app_dev.php
```

[13] Composer je nástroj pro jednoduchou správu závislostí v PHP projektu. Byl nainstalován automaticky jako jedna ze závislostí v soboru init.sh při spuštění příkazu vagrant up ve 4. kroku postupu.

[14] IP adresa odpovídá té, kterou má uživatel v souboru Vagrantfile.

neměl měnit a neměl by ani být součástí repositáře. Ve složce *web/* jsou umístěny kaskádové styly a javascripty. Také se zde nachází soubor *app.php*, který se spouští jako první při přístupu k aplikaci přes webové rozhraní.

MVC

Symfony používá architektonický vzor Model-View-Controller (česky Model-Pohled-Řadič). To znamená, že zdrojový kód je členěný na tři části. Modelová část obsahuje business logiku aplikace a práci s daty (databází). Vrstva View zařizuje zobrazení dat z vrstvy Modelu uživateli (to, co uživatel skutečně vidí v prohlížeči). Controller vše řídí – rozhoduje, jaká data z Modelu se mají ve View zobrazovat a naopak, jak se mají na datech projevit akce, které uživatel ve View provádí. [32]

Bude-li se vývojář řídit strukturou aplikace tak, jak je doporučená v dokumentaci Symfony, bude řídící logika umístěna ve složce Projekt/src/AppBundle/Controller/, všechny HTML šablony budou umístěny ve složce Projekt/src/AppBundle/Resources/views/ a v ostatních složkách bude business logika aplikace. S ohledem na tyto principy bude implementována aplikace v následujících kapitolách. [32]

4.1.4 Doctrine 2

Doctrine je projekt sestávající z několika PHP knihoven. Primárně cílí na řešení správy dat v aplikaci.

MySQL

Doctrine využívá knihovny **PDO** (PHP Data Objects), což je rozhraní umožňující základní přístup PHP aplikace k relační databázi. PDO samotné podporuje řadu RDBMS, kde z těch významnějších stojí za zmínku například Microsoft SQL Server, Oracle, PostgreSQL, SQLite a právě MySQL. [33]

Jedna z použitých Doctrine knihoven je Doctrine **DBAL** (Database abstraction and access layer – databázová abstrakční vrstva). Ta je prostředníkem mezi PHP aplikací (v tomto případě Symfony) a PDO vrstvou. Umožňuje zejména jednodušší dotazování nad relační databází a v kombinaci s další knihovnou, Doctrine **ORM** (Object-Relational Mapper – Objektově-Relační Mapování), zacházet s databázovými entitami jako s objekty v aplikaci.

Výsledkem je z pohledu programátora jednodušší manipulace s databází zevnitř aplikace. Implementovat do aplikace MySQL databázi je tak rychlejší a méně náchylné na chyby, neboť většina běžných úkonů s tím spojených je obstarána Doctrinou. [34] [35]

Součástí Doctrine DBAL je také jazyk DQL (Doctrine Query Language – dotazovací jazyk Doctrine). Ten je velmi podobný jazyku SQL a při použití Doctrine ORM může být výsledek rovnou mapován na objekty (jedná se především o dotazy typu SELECT, které vracejí nějaké záznamy). V některých velmi specifických případech se může stát, že dotaz pomocí jazyka DQL zkonstruovat nelze, i přestože v jazyce SQL je validní. V takovém případě lze v Doctrine vytvořit tzv. Native Query (v dokumentaci se lze setkat také s výrazem Native SQL), který má stejný syntax jako SQL. Výsledky takových dotazů však musí případně programátor mapovat na objekty ručně. [34]

Ve výsledku funguje práce s objekty tak, že programátor nadefinuje jednoduchým voláním funkcí na objektech, co si přeje s těmito objekty provést, Doctrine z těchto operací vybuduje dotaz v jazyce DQL, ten následně převede do Native SQL a předá ho vrstvě PDO, která již pracuje přímo s databází prostřednictvím nativních PHP funkcí.[15] [34]

MongoDB

Podobně jako Doctrine ORM mapuje relační databázové schéma na objekty, jsou prostřednictvím Doctrine **ODM** (Object-Document Mapper – Objektově-Dokumentové Mapování) mapovány dokumenty v MongoDB databázi opět na objekty. Přestože u MongoDB databáze neexistuje schéma, lze s objekty představujícími dokumenty pracovat, jako kdyby součástí schématu byly. Implementace tohoto mapování je součástí dalších kapitol. [36]

4.1.5 Modelová situace

Pro účely porovnání obou databázových systémů v praxi byla autorem navržena modelová situace.

---

[15] Ty byly nainstalovány jako součást oficiálního rozšíření *php5-mysql,* jehož instalace je definována v souboru *init.sh* na řádku 25. Podobným způsobem bylo nainstalováno také rozšíření *php5-mongo* (řádek 32 v *init.sh*).

Zdrojový kód aplikace, včetně dokumentace, bude psán výhradně v anglickém jazyce a to především z toho důvodu, že všechny používané technologie (Symfony, Doctrine,…) jsou rovněž v angličtině. Mísení českého a anglického jazyka ve zdrojovém kódu by dle autorova názoru vedlo k jeho špatné čitelnosti.

V následujících kapitolách bude uvažován vývoj webové aplikace, která by měla sloužit pro samostatně činné umělce k prezentování a prodeji svých děl. Umělci budou mít své uživatelské účty, u kterých bude evidováno jméno a příjmení. Umělci budou mít možnost nahrát do aplikace svoji tvorbu, která se bude dělit na dva typy: zvukovou a grafickou. U každého kusu umění budou uchovávány tyto informace:

- cena,
- informace, zda je použitelný komerčně,
- komerční cena,
- průměrné hodnocení,
- datová velikost v plné kvalitě,
- množina klíčových slov, které danou položku umění charakterizují (elektronická hudba, klasická, jazz,…, vektorová grafika, umělecké fotografie,…).

U zvukových záznamů bude navíc uchována informace o délce záznamu ve vteřinách. U grafických záznamů bude uložen rozměr v pixelech.

Aplikace bude podporovat také další samostatný typ uživatelů – recenzisty (jméno a příjmení). Ti mohou psát recenze k jednotlivým položkám umění.

Pro potřeby demonstrace nejsou potřeba a nebudou proto implementovány následující funkcionality:

- autentizace a autorizace uživatelů (v databázi tedy ani nebudou uchovávány přihlašovací jména, hesla),
- administrace webové aplikace, uživatelská podpora a správa obsahu,
- frontend webové aplikace.

4.1.6 Kompletní projekt

Součástí repositáře staženého v podkapitole 4.1.2 je i hotový, funkční projekt. Nachází se ve složce /www/RelDoc a k jeho zprovoznění stačí pouze nainstalovat jeho závislosti pomocí nástroje *composer* a vytvořit jeho MySQL databázi. To lze provést připojením do virtuálního stroje přes SSH a postupným spuštěním následujících příkazů:

Výpis 9: *Sada příkazů pro inicializaci hotového projektu (zdroj: autor).*

```
cd /vagrant-www/RelDoc
composer install
php app/console doctrine:database:create
php app/console doctrine:schema:create
```

Projekt je nyní přístupný z webového prohlížeče na adrese 192.168.10.12/RelDoc/web. V projektu byla použita databáze MySQL verze 5.5.47 a MongoDB verze 2.6.

4.2 Implementace MySQL

4.2.1 Konfigurace

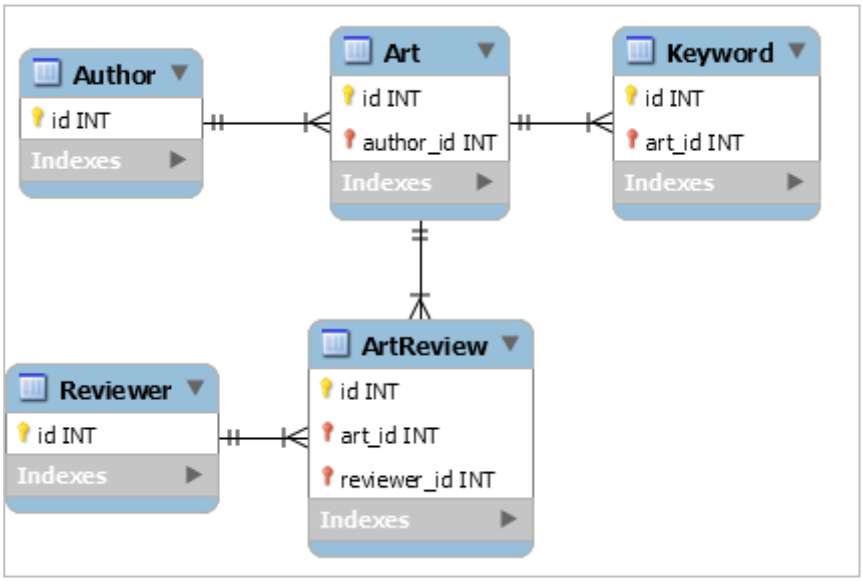
Nastavení knihoven Doctrine DBAL a Doctrine ORM se provádí v konfiguračním souboru /app/config/config.yml. Ve stejné složce se nacházejí také soubory config_dev.yml, config_prod.yml a config_test.yml. Tyto soubory slouží k různému nastavení v závislosti na prostředí. Lze tedy například nastavit používání jedné databáze (nebo jakékoliv jiné nastavení, které patří do config.yml) ve vývojovém prostředí (dev), jiné v testovacím (test) a jiné v produkčním (prod). V této práci však tato možnost využití nenalezne a veškeré nastavení tedy bude prováděno v souboru config.yml.

Výpis 10: *Konfigurace MySQL (zdroj: autor).*

```
# /app/config/config.yml
40 doctrine:
41    dbal:
42        driver:   pdo_mysql
43        host:     "%database_host%"
44        port:     "%database_port%"
45        dbname:   "%database_name%"
46        user:     "%database_user%"
47        password: "%database_password%"
48        charset:  UTF8
...
56    orm:
57        auto_generate_proxy_classes: "%kernel.debug%"
58        naming_strategy: doctrine.orm.naming_strategy.underscore
59        auto_mapping: true
```

Ve výpisu je config.yml tak, jak byl vygenerován pro nový projekt. Na řádcích 41 až 48 je nastaveno samotné připojení k databázi. Zápis ve tvaru "%database_host%", znamená, že hodnota není zadána přímo, ale je převzata ze souboru s aplikačními parametry /app/config/parameters.yml. Hodnoty v souboru parameters.yml byly zadávány při počáteční instalaci projektu a lze je kdykoliv měnit. V sekci s nastavením DBAL lze specifikovat i mnoho dalších možností, například logování všech databázových dotazů do dedikovaného souboru. V části s nastavením ORM lze nastavit způsoby mapování, cestu k mapovacím souborům, způsob automatického generování mapovacích souborů z existujících databází a jiné. Výchozí nastavení DBAL i ORM je vyhovující a nebude tedy zatím měněno.

4.2.2 Implementace schématu



***Obrázek 3**: ER diagram modelové aplikace (zdroj: autor).*

Pro modelovou situaci, popsanou v podkapitole 4.1.5, byl vytvořen následující ER diagram. V této verzi diagramu jsou naznačeny pouze vazby mezi entitami. Plná verze diagramu, obsahující také ostatní atributy, je součástí přílohy (Příloha C). Následující text se bude na tyto atributy často odvolávat.

V Symfony je potřeba začít vytvořením složky /src/AppBundle/Entity. V této složce budou umístěny PHP třídy představující jednotlivé relace relačního modelu. Začít lze třeba vytvořením souboru Art.php, který bude představovat entitu umění.

Výpis 11 znázorňuje vybrané části souboru Art.php. Přestože kód projektu splňuje (až na výjimky) standardy psaní kódu PSR-1 a PSR-2, byl kód v ukázce upraven z důvodu

úspory prostoru. Čísla řádků odpovídají původní, plné verzi souboru Art.php, která je dostupná v rámci hotového projektu (viz podkapitola 4.1.6).

Výpis 11: *Soubor /src/AppBundle/Entity/Art.php v zestručněné podobě (zdroj: autor).*

```
1  <?php
3  namespace AppBundle\Entity;
5  use Doctrine\ORM\Mapping as ORM;
8  /**
9   * Art
10  * @author Dominik Firla
11  * @ORM\Table(name="Art")
13  * @ORM\Entity()
14  * @ORM\HasLifecycleCallbacks()
15  * @ORM\InheritanceType("SINGLE_TABLE")
16  * @ORM\DiscriminatorColumn(name="type", type="string")
17  * @ORM\DiscriminatorMap({"music" = "Music", "picture" = "Picture"})
18  */
19 abstract class Art {
20     /**
22      * @ORM\Column(name="id", type="integer")
23      * @ORM\Id
24      * @ORM\GeneratedValue(strategy="AUTO")
25      */
26     private $id;
...
299    /**
300     * @ORM\PrePersist()
302     */
303    public function prePersist() {$this->setDatePublished(new \DateTime());}
```

Základní mapování

V Doctrine ORM existuje několik způsobů, jak mapovat PHP třídy na databázové entity. Mapování lze definovat v samostatných XML nebo YAML souborech, pomocí PHP přímo ve třídách, nebo pomocí anotací (tedy kódu, který je součástí komentářů metod a objektů). V této práci byla dána přednost variantě anotací.[16] [35]

Před definicí třídy je na řádcích 11 a 13 řečeno, že se jedná o mapovanou entitu, která patří do relace „Art“. Řádek 14 lze použít k implementaci tzv. callback funkcí, kde jedna taková

[16] Při volbě vhodné varianty záleží z velké části na subjektivních preferencích programátora. V této práci jsem zvolil anotace, protože dle mého názoru je vhodné mít mapování přímo u objektů entit (což vylučuje varianty XML a YAML souborů), a mapování pomocí PHP považuji za neúhledné a špatně přehledné (pozn. autora).

je definována na konci výpisu, na řádcích 299 až 303, a říká, že při ukládání nového záznamu má být nastaveno datum publikace (atribut date_published) na dnešní datum.

Řádky 15 až 17 definují dědičnost. Databáze má uchovávat záznamy hudby a obrázků, které mají většinu atributů společných. Lze je tedy ukládat ve společné tabulce (řádek 15) Art, ve které bude existovat atribut *type* (řádek 16), a pokud hodnota tohoto atributu bude „music“, bude entita mapována na instanci třídy Music a totéž platí analogicky pro obrázky (řádek 17). Samotná třída Art je deklarovaná jako abstraktní, protože není žádoucí mít záznamy umění, které nejsou ani hudbou ani obrázkem. Třída potomka Music tedy bude vypadat, jak je naznačeno ve výpise 12.

Výpis 12: *Soubor src/AppBundle/Entity/Music.php ve stručné podobě (zdroj: autor).*

```
1 <?php
3 namespace AppBundle\Entity;
5 use Doctrine\ORM\Mapping as ORM;
7 /**
8  * Music
10 * @ORM\Entity()
11 */
12 class Music extends Art {
13     /**
14      * @ORM\Column(name="length", type="integer", nullable=false)
15      */
16     private $length;
...
21     public function getLength() {return $this->length;}
...
28     public function setLength($length) {$this->length = $length;}
```

Anotace na řádku 14 definuje, že atribut délky audiozáznamu bude v databázi pojmenován *length* a bude datového typu *integer* (v MySQL datový typ INT()). Další datové typy použité v entitách jsou string, boolean, text, date a float, které jsou v MySQL mapované na varchar(), tinyint(1), longtext, date a float (jejich vysvětlení je součástí přílohy A). V tomto výpise, je také ukázka metod *get* a *set*, které slouží pro nastavování a čtení hodnot atributů. Tyto metody si může programátor a) napsat ručně, b) nechat je negenerovat svým vývojovým prostředím (pokud je v něm taková funkce podporována), c) použít Doctrine příkaz. U poslední možnosti Doctrine pomocí obslužného příkazu vygeneruje pouze chybějící metody a již existující nepřepíše. Zároveň tento příkaz umí vygenerovat také metody potřebné pro práci s entitami, které jsou součástí asociace, což bude demonstrováno níže. [32]

```
php app/console doctrine:generate:entities AppBundle/Entity/Art
```

Za zmínku také stojí, že v entitě Music již není potřeba definovat mapování primárního klíče, to je provedeno v entitě Art na řádcích 22 až 24, kde je řečeno, že atribut „id“ je identifikátor záznamu a jeho hodnota je generována automaticky.

Mapování asociací

Součástí schématu je také množství vazeb *1–n*. V následujících dvou výpisech je zaznamenána implementace jedné takové vazby mezi entitou umělce (Author) na straně *1* a entitou umění (Art) na straně *n*.

Výpis 13: *Vazba 1–n mezi entitami Author a Art, strana n (zdroj: autor).*

```
#       /src/AppBundle/Entity/Art.php
28      /**
29       * @var Author
31       * @ORM\ManyToOne(targetEntity="Author", inversedBy="arts")
32       * @ORM\JoinColumn(name="author_id", referencedColumnName="id", onDelete="CASCADE")
33       */
34      private $author;
...
115     /** @return Author */
118     public function getAuthor() {return $this->author;}
121
122     /** @param Author $author */
125     public function setAuthor(Author $author) {$this->author = $author;}
```

V prvním výpise je entita Art, která má podle schématu obsahovat cizí klíč *author_id* odkazující na primární klíč *id* na entitě Author. Na řádku 31 výpisu je definováno, o kterou stranu vazby se jedná pomocí anotace *ManyToOne*, na kterou entitu je vazba realizována pomocí parametru *targetEntity*, a název atributu na opačné straně vazby pomocí parametru *inversedBy*. Řádek 32 pak definuje atribut cizího klíče, kde parametr *name* je jméno cizího klíče v relaci Art a parametr *referencedColumnName* je jméno primárního klíče v relaci Author. Parametr *onDelete* s hodnotou *CASCADE* znamená, že při smazání entity Author budou smazány také všechny záznamy v tabulce Art, které na něj odkazovaly. [35]

Výpis 14: *Vazba 1–n mezi entitami Author a Art, strana 1 (zdroj: autor).*

```
#       /src/AppBundle/Entity/Author.php
6  use Doctrine\Common\Collections\ArrayCollection;
37      /**
38       * @var ArrayCollection|Art[]
40       * @ORM\OneToMany(targetEntity="Art", mappedBy="author", orphanRemoval=true,
cascade={"persist"})
41       */
42      private $arts;
```

```
43
44      public function __construct() {$this->arts = new ArrayCollection();}
...
83      /** @param Art $art */
86      public function addArt(Art $art) {
87              $this->arts[] = $art;
88              $art->setAuthor($this);
89      }
90
91      /** @param Art $art */
94      public function removeArt(Art $art) {$this->arts->removeElement($art);}
97
98      /** @return ArrayCollection|Art[] */
101     public function getArts() {return $this->arts;}
```

Podobně jako ve třídě Art je přítomna anotace ManyToOne, je ve třídě Author anotace OneToMany. Parametr *targetEntity* bude analogicky obsahovat jméno entity na opačné straně vazby a parametr *mappedBy* funguje jako protějšek parametru *inversedBy*.

Všechny entity Art odkazující na konkrétně tuto entitu Author, jsou v objektově relačním mapování dostupné v Doctrine objektu Collection (v této definici ArrayCollection, která je potomek třídy Collection), což je obalová třída pro PHP pole *array*. V třídě Author se budou pro tuto kolekci vyskytovat metody *add, remove* a *get*, které slouží k přidání, odebrání a čtení položek kolekce tříd Art. Parametry anotace OneToMany *orphanRemoval* a *cascade={"persist"}* Doctrine umožní automaticky vkládat nebo mazat entity Art, pokud jsou volány metody add/remove. Správa vztahů mezi entitami je tak v Doctrine velmi jednoduchá.[35]

Vygenerování databáze

Byly popsány všechny postupy potřebné k definici celého schématu. Jak by měl přibližně vypadat výsledek lze vidět na hotovém projektu RelDoc. Po definici všech tříd a jejich mapování lze celou MySQL databázi vygenerovat a udržovat pomocí následujících příkazů (příkazy předpokládají, že uživatel se nachází ve složce /vagrant-www/Projekt/):

Výpis 15: *Sada příkazů pro údržbu relační databáze (zdroj: autor).*

```
1       php app/console doctrine:database:create
2       php app/console doctrine:database:drop
3       php app/console doctrine:schema:create
4       php app/console doctrine:schema:update --force
5       php app/console doctrine:schema:drop
```

Příkazy 1 a 2 slouží k vytvoření/smazání databáze. Příkaz 3 slouží k vytvoření schématu, příkaz 4 k jeho aktualizaci v případě pozdějších změn v mapování a nakonec příkaz 5 k jeho smazání. Pokud se v mapování vyskytují chyby, bude uživatel informován ve výstupu těchto příkazů. [32]

4.2.3 Práce s daty

V hotovém projektu (podkapitola 4.1.6) jsou nad kompletním schématem prováděny CRUD operace (create, read, update, delete). V SQL jazyce jim odpovídají příkazy INSERT, SELECT, UPDATE a DELETE. Při přístupu z webového prohlížeče na adresu 192.168.10.12/RelDoc/web se zobrazí v nabídce možnosti odpovídající těmto operacím. Každá možnost vede na jednu akci v aplikaci, které jsou definované v souboru /src/AppBundle/Controller/MysqlController.php. Obsah jednotlivých metod může uživatel libovolně měnit a experimentovat s ním. Pro zobrazení databáze (a ověření výsledků svých akcí) je pak vhodné využít nějakého MySQL klienta, jako například open-source MySQL Workbench, jež byl využit již dříve k tvorbě ER diagramů. V jednotlivých metodách akcí jsou již implementovány některé operace a ty budou nyní blíže vysvětleny.

INSERT

Jak lze na výpisu vidět, je možné jednoduše vytvořit objekty Music a Author, a následně jim pomocí jejich metod nastavit povinné hodnoty. Na řádku 28 je pomocí techniky Service Location (načítání služeb) načtena třída EntityManager, která slouží ke správě entity v ORM. Metoda EntityManager::persist(), volaná na dalším řádku, naplánuje INSERT operaci objektu předaného parametrem. Metoda EntityManager::flush() provede všechny naplánované operace (v transakci) a po jejím zavolání budou v databázi existovat nové záznamy entit Author a Music. [32]

Výpis 16: *Vytvoření nového záznamu pomocí ORM (zdroj: autor).*

```
16     public function insertAction(Request $request) {
17             // creates object of Author and sets its mandatory properties
18             $author = new Author();
19             $author->setFirstName("Jan");
20             $author->setLastName("Novák");
21
22             // creates object of Music and sets its mandatory properties
```

```
23            $music = new Music();
24            $music->setName("Hudba 1");
25            $music->setSize(150);
26            $author->addArt($music); // adds Music to collection of Art on Author object
27
28            $entityManager = $this->get("doctrine")->getManager(); // retrieves
Doctrine\ORM\EntityManager
29            $entityManager->persist($author); // schedules Author for insert
30            $entityManager->flush(); // peforms all scheduled operations
```

SELECT

Doctrine umožňuje mnoho způsobů, jak načítat data. Na následujícím výpise jsou nejprve demonstrovány nenáročné dotazy, které lze realizovat voláním různých variací metod *find()* na objektu repositáře konkrétní relace. Výsledek dotazů lze vypsat pomocí ladící funkce Symfony \dump().[17][32]

Výpis 17: *Načtení záznamů v ORM metodami repositáře (zdroj: autor).*

```
38 public function selectAction(Request $request) {
39     $entityManager = $this->get("doctrine")->getManager();
40     $result1 = $entityManager->getRepository('AppBundle:Author')->find(1);
41     $result2 = $entityManager->getRepository('AppBundle:Author')-
>findOneByFirstName("Jan");
42     $result3 = $entityManager->getRepository('AppBundle:Author')-
>findBy(array('lastName' => 'Novák'));
43     \dump($result1);
44     exit;
```

Pro implementaci složitějších dotazů však tyto metody nestačí a je potřeba využít DoctrineQueryBuilder. Tato třída umožňuje voláním metod kousek po kousku poskládat celý dotaz v jazyce DQL. Ukázka takového dotazu je na dalším výpise. Cílem dotazu je načíst všechny umělce, kteří jsou autory alespoň jednoho umění, které je možné používat komerčně a má přiřazené klíčové slovo „dolorem“. Na konci výpisu je pro srovnání výsledná podoba dotazu v jazyce DQL a SQL. [32]

---

[17] Při použití nativních PHP funkcí jako var_dump nebo echo pro vypisování objektů v Symfony, dojde při výpisu k zacyklení vykonávaného skriptu, je tedy vhodné použít funkci \dump(), která je dostupná v *dev* prostředí (pozn. autora).

Výpis 18: *Doctrine Query Builder a jazyk DQL (zdroj: autor).*

```
46 $queryBuilder = $entityManager->createQueryBuilder();
47 $queryBuilder   ->select('author')
48                  ->from('AppBundle:Author', 'author')
49                 ->leftJoin('AppBundle:Art', 'art', 'WITH', 'art.author = author')
50                 ->leftJoin('AppBundle:Keyword', 'keyword', 'WITH', 'keyword.art = art')
51                 ->where('art.commercial = true')
52                 ->andWhere('keyword.keyword = :word')
53                 ->setParameter('word', 'dolorem');
54 $result4 = $queryBuilder->getQuery()->getResult();

DQL:
SELECT author
FROM AppBundle:Author author
LEFT JOIN AppBundle:Art art WITH art.author = author
LEFT JOIN AppBundle:Keyword keyword WITH keyword.art = art
WHERE art.commercial = true AND keyword.keyword = :word

SQL:
SELECT a0_.id AS id_0, a0_.first_name AS first_name_1, a0_.last_name AS last_name_2
FROM Author a0_
LEFT JOIN Art a1_ ON (a1_.author_id = a0_.id) AND a1_.type IN ('music', 'picture')
LEFT JOIN Keyword k2_ ON (k2_.art_id = a1_.id)
WHERE a1_.commercial = 1 AND k2_.keyword = 'epic';
```

UPDATE a DELETE

Na implementaci těchto dvou operací už není nic složitého. Pro provedení operace UPDATE stačí načtenou entitu upravit podobně jako při operaci INSERT (není potřeba volat metodu *persist()).* Pro smazání záznamu je potřeba načtenou entitu označit pro operaci DELETE pomocí metody EntityManager::remove(). Obě operace se provedou po zavolání metody flush(). [32]

4.3 Implementace MongoDB

4.3.1 Konfigurace

Zatímco Doctrine ORM je součástí standardní instalace Symfony, u Doctrine ODM tomu tak není. Nejprve je potřeba přidat závislost na Doctrine ODM a MongoDB ODM Bundle (balíček, jež slouží jako most mezi knihovnou Doctrine ODM a frameworkem Symfony)

do souboru ./composer.json. Nové závislosti patří do části "require", takže výsledek by měl vypadat přibližně takto:

Výpis 19: *Závislost na Doctrine ODM v composer.json (zdroj: autor).*

```
19  "require": {
...
29        "doctrine/mongodb-odm": "~1.0",
30        "doctrine/mongodb-odm-bundle": "~3.0"
31  }
```

Přidané závislosti se nyní automaticky nainstalují zavoláním příkazu *composer update* přes SSH ve složce se souborem composer.json. Stažený balíček je nyní nutné zaregistrovat v Symfony. K tomu je potřeba upravit soubor /app/AppKernel.php, kde má Symfony nadefinováno, jaké balíčky má načítat ze složky /vendor. Zároveň je potřeba zaregistrovat knihovnu s anotacemi v souboru /app/autoload.php. Obě úpravy jsou zaznamenány v následujícím výpisu. [37]

Výpis 20: *Registrace balíčku ODM a anotací v Symfony (zdroj: autor).*

```
# /app/AppKernel.php
$bundles = [
...
new Doctrine\Bundle\MongoDBBundle\DoctrineMongoDBBundle()
];

# /app/autoload.php
use Doctrine\ODM\MongoDB\Mapping\Driver\AnnotationDriver;
AnnotationDriver::registerAnnotationClasses();
```

Podobně jako je ORM konfigurován v souboru /app/config/config.yml, je na stejném místě konfigurováno připojení k databázi MongoDB. V následujícím výpise je velmi jednoduché, ale postačující nastavení. Na řádcích 64 až 67 je nastavena adresa databáze a přístupový port (27017 je standardní číslo portu pro MongoDB). Následuje jméno databáze a nakonec nastavení mapování. [37]

Výpis 21: *Konfigurace MongoDB (zdroj: autor).*

```
# /app/config/config.yml
63 doctrine_mongodb:
64    connections:
65        default:
66            server: mongodb://localhost:27017
67            options: {}
68    default_database: ArtStore
69    document_managers:
70        default:
71            auto_mapping: true
```

4.3.2 Definice dokumentů

Mapování dokumentů v ODM je velmi podobné mapování entit v ORM. Následující výpis popisuje mapování objektu Art na dokument. Tento výpis je analogický k výpisu 11, kde byla stejná část objektu mapována pomocí ORM.

Výpis 22: *Soubor src/AppBundle/Document/Art.php ve stručné podobě (zdroj: autor).*

```
1  <?php
3  namespace AppBundle\Document;
5  use Doctrine\ODM\MongoDB\Mapping\Annotations as MongoDB;
7  /**
8   * @author Dominik Firla
9   * @MongoDB\Document
10  * @MongoDB\MappedSuperclass
11  * @MongoDB\HasLifecycleCallbacks
12  * @MongoDB\InheritanceType("SINGLE_COLLECTION")
13  * @MongoDB\DiscriminatorField("type")
14  * @MongoDB\DiscriminatorMap({"music"="Music", "picture"="Picture"})
15  */
16 abstract class Art {
17     /**
18      * @MongoDB\Id
19      */
20     private $id;
...
232    /** @MongoDB\PrePersist */
235    public function prePersist() {$this->setDatePublished(new \DateTime());}
```

Mapování dokumentů je na první pohled velmi podobné. Jak si lze všimnout z definice jmenného prostoru na řádku 3, dokumenty je potřeba umístit do složky /src/AppBundle/Document. Následuje mapování pomocí anotací, podobně jako v souboru /src/AppBundle/Entity/Art.php. Na řádcích 9 až 14 je postupně deklarováno, že se jedná o MongoDB dokument, který bude využívat callback funkce v rámci jeho životnosti, a jenž je rodičem potomků Music a Picture. Ve jmenném prostoru *Doctrine\ODM\MongoDB\Mapping\Annotations* se nacházejí anotace pro mapování datových typů. S jejich pomocí ODM automaticky převádí datové typy MongoDB na datové typy v PHP. V dokumentech jsou pro základní atributy použity například datové typy @Id (řádek 18 výpisu), @String, @Float nebo @Date. [36]

Mapování asociací

Mezi MongoDB dokumenty lze utvářet asociace podobně jako v relačním modelu. Na následujícím výpisu je znázorněná vazba 1–n mezi dokumenty Author a Art.

Výpis 23: *Vazba 1–n mezi entitami Author a Art (zdroj: autor).*

```
#       /src/AppBundle/Document/Art.php
22      /**
23       * @MongoDB\ReferenceOne(targetDocument="Author")
24       */
25      private $author;


#       /src/AppBundle/Document/Author.php
6 use Doctrine\Common\Collections\ArrayCollection;
...
27      /**
28        * @MongoDB\ReferenceMany(targetDocument="Art", cascade={"persist"})
29        */
30      private $arts;
31
32      public function __construct() {$this->arts = new ArrayCollection();}
...
71      /** @return ArrayCollection */
74      public function getArts() {return $this->arts;}
```

Pro odkazování na jeden dokument je použita anotace @ReferenceOne (řádek 23). Pro atribut autora budou dále ve třídě Art deklarovány metody getAuthor a setAuthor. V PHP budou přidružené dokumenty Art dostupné prostřednictvím objektu ArrayCollection a v ODM budou mapovány pomocí anotace @ReferenceMany. Metody *set* a *get* lze i v ODM vygenerovat příkazem. [36]

```
php app/console doctrine:mongodb:generate:documents AppBundle
```

Konsolidace dokumentů

V MongoDB databázi je pro snazší a rychlejší dotazování vhodné udržovat co nejvíce souvisejících informací pohromadě. Vhodným návrhem je zahození dokumentu Keyword, který by obsahoval pouze ID dokumentu a samotné klíčové slovo. Na místo toho bude v dokumentu Art atribut $keywords, který bude mapován pomocí anotace @MongoDB\Collection. [36]

```
74      /** @MongoDB\Collection */
76     private $keywords = array();
```

Vnořené dokumenty

V MongoDB lze pro modelovou situaci využít možnosti vnoření dokumentů do sebe a tímto celkové množství dokumentů dále snížit. Zároveň se tím zjednoduší některé typy dotazů, v MongoDB přece jen nelze provádět operace JOIN jako v MySQL. Z toho důvodu

nebude vytvářen samostatný dokument pro recenze ArtReview, ale bude vnořen v dokumentu Art. Vnořit dokument lze anotací @MongoDB\EmbedMany, jak je tomu v následujícím příkladu. [36]

Výpis 24: *Vnořené dokumenty (zdroj: autor).*

```
# /src/AppBundle/Document/Art.php
79      /** @MongoDB\EmbedMany(targetDocument="ArtReview") */
81     private $reviews;
86      public function __construct() {$this->reviews = new ArrayCollection();}

# /src/AppBundle/Document/ArtReview.php
        /** @MongoDB\EmbeddedDocument */
        class ArtReview {...}
```

Vygenerování kolekcí

V MongoDB není potřeba vytvářet kolekce předem. Při vytvoření prvního dokumentu bude zároveň vytvořena databáze, pokud dosud neexistuje. Stejně tak s vytvořením prvního dokumentu je eventuálně založena i kolekce. Protože MongoDB nemá schéma, je ihned po namapování objektů databáze připravena k použití. Databázi, všechny namapované kolekce a případné indexy lze vytvořit/odstranit pomocí následujících dvou příkazů:

```
1      php app/console doctrine:mongodb:schema:create
2      php app/console doctrine:mongodb:schema:drop
```

4.3.3 Práce s daty

V hotovém projektu jsou na adrese 192.168.10.12/RelDoc/web nadefinované CRUD operace nad MongoDB databází. Jednotlivé možnosti nabídky v sekci MongoDB vedou na příslušné akce v řadiči RelDoc/src/AppBundle/Controller/MongoController.php.

INSERT, UPDATE a DELETE

Při tomto typu operací aplikace pracuje s objekty stejným způsobem, jako tomu bylo v podkapitole 4.2.3. Namísto třídy Doctrine\ORM\EntityManager se však pro správu dokumentů používá Doctrine\ODM\MongoDB\DocumentManager. DocumentManager také implementuje rozhraní Doctrine\Common\Persistence\ObjectManager a má tedy naimplementované metody jako *persist(), remove(), flush(), find()* a další. DocumentManager je v kontrolerech přístupný pomocí Service Location takto:

```
use Doctrine\ODM\MongoDB\DocumentManager;
...
$documentManager = $this->get('doctrine_mongodb')->getManager();
```

Vzorové operace pro vytváření, modifikaci a mazání dokumentů jsou implementovány v příslušných metodách insertAction(), updateAction() a deleteAction() ve třídě MongoController.

SELECT

Stejně jako EntityManager, DocumentManager rovněž implementuje metody find(), findOneBy(), findBy() a podobné. Chovají se a fungují stejně, není tedy potřeba věnovat jim bližší pozornost. Rozdíl je však v tvorbě složitějších dotazů. Následující výpis obsahuje implementaci kódu, jehož cílem je načtení všech umělců, kteří jsou autory alespoň jednoho umění, které je možné používat komerčně a má přiřazené klíčové slovo „dolorem“, tak jako v předchozí kapitole.

Výpis 25: *Doctrine Query Builder pro dotazování nad MongoDB dokumenty (zdroj: autor).*

```
# /src/AppBundle/Controller/MongoController.php
48        $artQueryBuilder = $documentManager->createQueryBuilder('AppBundle:Art');
49        $artQueryBuilder   ->field('commercial')->equals(true)
50                           ->field('keywords')->in(array('dolorem'));
51        $result4 = $artQueryBuilder->getQuery()->execute();
52        foreach ($result4 as $art) {
53                  $art->getAuthor();
54        }
```

Při volání metody execute() na řádku 51 bude vykonán tento dotaz:

```
db.Art.find({ "commercial": true, "keywords": { "$in": [ "epic" ] } });
```

Tato metoda nevrátí instanci třídy Art, nýbrž objekt Doctrine\ODM\MongoDB\Cursor. Objekt třídy Cursor v sobě nese odkazy na všechny dokumenty, které vyhověly volanému dotazu. Tento objekt lze iterovat a procházet tak všechny dokumenty, jež jsou výsledkem dotazu. [32]

4.4 Srovnání implementace obou řešení

Konfigurace

Na začátku byly oba systémy nakonfigurovány. Konfigurace je velmi rychlá a základní, fungující konfigurace i jednoduchá. U MongoDB bylo oproti MySQL nejprve potřeba doinstalovat potřebné knihovny, protože nejsou součástí standardní instalace. Ve výsledku je konfigurace obou systému na srovnatelné úrovni.

Mapování

Už při mapování databáze na PHP objekty se projevila absence schématu v MongoDB. Mapování v ODM je stručnější, protože v něm není potřeba definovat například jména sloupců nebo povinnost hodnot. Zatímco u MySQL bylo potřeba držet se ER diagramu, u MongoDB byly dokumenty denormalizovány ve snaze udržovat co nejvíce souvisejících informací pohromadě. Oproti pěti tabulkám v relační databázi si tedy v MongoDB řešení aplikace vystačí se třemi kolekcemi dokumentů.

Díky tomu, že ORM i ODM jsou produkty Doctrine, je i o něco složitější deklarace dědičnosti mezi entitami/dokumenty prakticky totožná.

Mapováním pomocí ORM ani ODM nelze validovat hodnoty nastavené jednotlivým atributům objektů. Pro validaci je potřeba využít buďto Symfony formulářů, nebo omezení z jmenného prostoru Symfony\Component\Validator\Constraints (případně kombinaci obou zmíněných). Validací objektů se tato práce nezabývá. [32]

Práce s daty

Pro jednoduché dotazy, kde je možné použít metodu třídy EntityManager nebo DocumentManager *find()*, případně její variace, není v práci s daty v obou případech rozdíl. Pro složitější dotazy je potřeba použít QueryBuilder (\Doctrine\ORM\QueryBuilder případně \Doctrine\ODM\MongoDB\Query\Builder). Tvorba dotazů nad ORM QueryBuilderem je pro programátora znalého jazyka SQL velmi intuitivní. Tvorba dotazů nad ODM je však (už z podstaty fungování databáze a absence schématu) implicitně jednodušší.

Postupy konfigurace, mapování a tvorby dotazů jsou v obou případech velmi dobře zdokumentované. Při volbě řešení z hlediska obtížnosti implementace jsou si oba porovnávané systémy rovny a je potřeba se tedy rozhodovat podle jiných hledisek.

4.5 Srovnání výkonu obou řešení

V této kapitole budou oba systémy porovnány s hlediska rychlosti zpracovávání dotazů. Nejprve bude popsán způsob přípravy testovacích dat. Následně budou osvětleny možnosti měření rychlosti dotazů pro oba systémy. Po přípravě prostředí bude definováno několik dotazů, na kterých bude následně provedeno měření. Výsledky měření budou v závěru kapitoly vyhodnoceny.

Všechny operace, jako generování dat nebo spouštění dotazů, byly prováděny na zařízení, jehož specifikace jsou uvedeny v podkapitole 4.4.1. Virtuální stroj měl pro účely této práce přidělena 2 jádra procesoru a 5 120 MB operační paměti (viz Příloha B).

4.5.1 Generování testovacích dat

Pro generování testovacích dat byl použit Symfony balíček AliceBundle, jenž sdružuje dvě knihovny – Alice a Faker. Tyto knihovny slouží ke generování velkého množství náhodných dat, která splňují požadavky na formát (jména, telefonní čísla, adresy, časové údaje,…). [38]

Konfigurace

Pro nainstalování balíčku je nejprve potřeba jej přidat jako závislost do Composeru. To lze provést tímto příkazem, volaným ve složce se souborem *composer.json* v kořenovém adresáři projektu:

```
composer require --dev hautelook/alice-bundle doctrine/data-fixtures
```

Následně je potřeba balíček zaregistrovat v souboru /app/AppKernel.php, přidáním mezi načítané balíčky:

```
# /app/AppKernel.php
$bundles = [
...
new Hautelook\AliceBundle\HautelookAliceBundle ()
];
```

A nakonec balíček nakonfigurovat v souboru /app/config.yml.

Výpis 26: *Konfigurace balíčku Alice (zdroj: autor).*

```
# /app/config.yml
hautelook_alice:
    db_drivers:
        orm: ~
        mongodb: ~
    locale: cs_CZ
    seed: 1
    persist_once: false
    loading_limit: 5
```

Tato konfigurace vesměs říká, jaké databázové systémy jsou použity, v jaké lokalizaci mají být generovaná data, a že data se mají při opětovném generování vygenerovat stejně. [38]

Generování dat

Nyní stačí ve složce /src/AppBundle/DataFixtures/ORM nadefinovat soubory typu YAML s pravidly pro generování testovacích dat. Deklarace pravidel generování dat může vypadat přibližně takto:

Výpis 27: *Definice podoby testovacích dat v balíčku Alice (zdroj: autor).*

```
# /src/AppBundle/DataFixtures/ORM/LoadArts.yml
AppBundle\Entity\Music:
    music{1..25000}:
        name: <words(3, true)>
        price: <randomElement([0, 0, 0, 620, 8900, 739, 299, 59, 20])>
        commercialPrice: 30%? <randomElement([620, 8900])>
        commercial: <boolean(50)>
        author: @author<numberBetween(1, 5000)>
        ...
```

Pro účely testování bylo pro MySQL připraveno generování 50 000 záznamů v tabulce Art (polovina typu Music a polovina typu Picture), 5 000 záznamů v tabulce Author, 5 000 recenzistů v tabulce Reviewer a 150 000 záznamů v tabulce Keyword. Každá entita Art byla přidělena náhodnému autorovi, každá entita Keyword náhodné entitě Art. Rovněž bylo připraveno 50 000 recenzí (ArtReview), jakožto vazby mezi entitami Art a Reviewer.

Nad prázdnou databází s vytvořeným schématem lze testovací data pro MySQL vygenerovat tímto příkazem:

```
php app/console hautelook_alice:doctrine:fixtures:load --no-interaction
```

Stejná data, ale odpovídající struktuře dokumentů, byla připravena pro naplnění kolekcí MongoDB databáze. Vygenerování dokumentů lze provést podobným příkazem: [18]

```
php app/console hautelook_alice:doctrine:mongodb:fixtures:load --no-interaction
```

Vykonávání každého ze dvou příkazů může trvat až několik desítek minut. Celková doba závisí především na výkonu procesoru a množství přidělené operační paměti.

4.5.2 Monitorování výkonu

MySQL

Pro monitorování výkonu MySQL databáze byl použit výchozí Symfony Web Profiler. Tento ladící nástroj je dostupný v *dev* prostředí aplikace přes webový prohlížeč. V dolní části obrazovky se zobrazí ladící lišta a po rozkliknutí prostředí Profileru. Zde lze nalézt posledních 10 HTTP požadavků a u každého z nich si lze zobrazit všechny dotazy, které Doctrine vykonala včetně potřebného času v milisekundách. Tento nástroj bude použit pro měření rychlosti dotazů v MySQL. [19]

MongoDB

Přestože Symfony Web Profiler dokáže zobrazit vykonané dotazy nad MongoDB databází, nezobrazuje dobu zpracování těchto dotazů. Pro tyto účely je však možné využít profilování, které nabízí přímo MongoDB. Následujícím skupinou příkazů lze zapnout

---

[18] Oba příkazy před svým vykonáním nejprve smažou všechna data v tabulkách/kolekcích. Pro přidání dalších dat bez smazání původního obsahu databáze je potřeba na konec příkazu připsat parametr *--append*.

[19] *Dev* (development – vývoj) prostředí je ve webovém prohlížečí dostupné, pokud uživatel v URL adrese specifikuje prostředí pomocí řetězece /app_dev.php/. Výsledek by vypadal třeba takto: http://192.168.10.12/Projekt/web/app_dev.php/. V hotovém projektu RelDoc bylo toto prostředí nastaveno jako výchozí.

profilování v testované databázi. Poslední z příkazů vypíše několik posledních vykonávaných dotazů včetně doby zpracování:

Výpis 28: *Monitorování MongoDB (zdroj: autor).*

```
$ mongo
> use ArtStore
> db.setProfilingLevel(2)
> db.system.profile.find( { op: { $ne : 'command' } } ).pretty()
```

4.5.3 Vybrané dotazy

Pro porovnání výkonu obou databází bylo potřeba navrhnout vhodné databázové dotazy.

1. Vybrat všechny autory, kteří mají přiřazenou alespoň jednu položku umění, pro kterou zároveň platí, že je možné používat ji komerčně a má přiřazené klíčové slovo „dolorem“.
2. Vybrat záznamy Art typu Music, které jsou komerční, seřazené abecedně podle jména a s limitem 50 záznamů.
3. U všech záznamů typu Music změnit atribut „date_published“ na dnešní datum.

První z dotazů byl použit jako vzor při implementaci obou systémů v předchozí kapitole. Tento dotaz byl zvolen, protože lze sledovat vliv rozdílné implementace množiny Keywords v MySQL (samostatná tabulka s cizím klíčem na Art) a v MongoDB (jednoduchý výčet přímo v dokumentu Art).

Druhý dotaz je prováděn pouze nad jednou tabulkou/kolekcí a není proto potřeba žádných JOIN operací nebo donačítání referencovaných dokumentů. Kvůli požadavku na řazení je však stále potřeba projít celou tabulku/kolekci.

Třetí dotaz je podmíněná operace UPDATE.

Provedení dotazu

V souboru /src/AppBundle/Controller/PerformanceController.php byla uskutečněna realizace těchto dotazů prostřednictvím akce volané z webového prohlížeče (akce "Compare performance" dostupná na adrese 192.168.10.12/RelDoc/web). Po zavolání této akce je v metodě PerformanceController::performanceAction() provedena vybraná

operace. Pro co nejobjektivnější měření byla vždy pro každý běh aplikace provedena maximálně jedna operace.

Vyčištění cache (mezipaměti)

MySQL si ukládá do mezipaměti prováděné dotazy a jejich výsledky. Pro získání co nejobjektivnějších výsledků bylo měření několikrát provedeno i s vyprázdněním mezipaměti MySQL pomocí příkazu RESET QUERY CACHE;. Pro vyčištění operační paměti od MongoDB dokumentů byl opakovaně restartován server.

4.5.4 Výsledky testu

***Tabulka 7**: Výsledky měření výkonu (zdroj: autor)*

Dotaz	Databáze	Počet vykonaných dotazů	Čas v milisekundách s vyprázdněnou mezipamětí	Čas (v ms) při opakovaném dotazování
1	**MySQL**	1	170	2
1	**MongoDB**	3	137 (+ 21 +5)	34
2	**MySQL**	1	32	2
2	**MongoDB**	1	60	52
3	**MySQL**	1	145 až 190	92
3	**MongoDB**	1	136	60

Předcházející tabulka zaznamenává výsledky měření. Měření bylo prováděno opakovaně až do okamžiku, kdy bylo možné vyloučit z měření ojedinělé výkyvy a byla získána objektivní data.

Z výsledků u prvního dotazu lze vidět, že v případě složitějšího dotazu (přes více tabulek) je MongoDB trochu rychlejší při prvním dotazování, když ještě nejsou data v mezipaměti. U prvního dotazu stačilo MySQL díky JOIN operaci k získání potřebných dat jediného dotazu. MongoDB provedlo nejprve dotaz pro načtení položek Art a následně pro ně donačetlo autory (ve dvou dávkách). Při opakovaném dotazování bylo MySQL díky mezipaměti rychlejší.

V případě druhého dotazu (jednoduchý dotaz nad jednou tabulkou/kolekcí) MongoDB nedokázalo vytěžit z výhod mezipaměti tolik, co MySQL databáze, která opakovaný dotaz ,stejně jako v prvním případě, vykonala za pouhé 2 milisekundy.

U třetího dotazu byly poměrně velké výkyvy v rychlosti u MySQL při prvním volání dotazu s vyčištěnou mezipamětí. MongoDB u UPDATE operace dokázalo více využít načítání dokumentů z mezipaměti a při opakovaném volání bylo konzistentně o 30 milisekund rychlejší.

4.5.5 Odvozené závěry

Z výsledku testů lze odvodit, že při některých typech dotazů je lepší MySQL databáze a při jiných zase MongoDB. Velkou roli při rychlosti zpracování dotazů hraje právě mezipaměť.

V případě MySQL jsou do mezipaměti ukládány jak dotazy, tak i výsledky těchto dotazů. To znamená, že při opakovaném volání stejného dotazu je možné výsledek vrátit během pár milisekund. Pokud se však dotaz jen trochu změní, MySQL už dat v mezipaměti nevyužije a jeho vykonávání trvá delší dobu. Časté, stejné dotazy tedy MySQL dokáže zpracovávat velmi rychle, ale rozmanitost v dotazech jej činí méně efektivním.

U MongoDB nejsou ukládány dotazy a jejich výsledky, ale do paměti jsou načteny dokumenty, ke kterým bylo přistupováno. To znamená, že i rozmanitější dotazy nad stejnou kolekcí je možné na první volání provádět s maximální rychlostí, a opakované volání rychlost neovlivní. Všechny dokumenty testovací databáze, vytvořené na začátku této kapitoly, nezabíraly na disku ani 100 MB prostoru. Pokud by databáze měla rozsah třeba několika GB, byl by efekt mezipaměti na rychlost výkonu pravděpodobně nižší.

Výstupem tohoto testu budiž poznatek, že žádná z obou databází není jednoznačně rychlejší, než ta druhá. Při volbě, kterou z nich vybrat pro konkrétní aplikaci, je vhodné nejprve analyzovat, jaké typy dotazů se budou převážně provádět, nebo budou kritické na rychlost zpracování.

5 Závěr

Dosažené poznatky

V teoretické části této práce byly nejprve představeny principy relačních databází a speciálně byla představena databáze MySQL. Následně byly popsány NoSQL databáze a jejich jednotlivé typy. Bližší pozornost byla věnována databázím dokumentovým a především pak jejich představiteli MongoDB. Tato část práce umožnila pochopit hlavní rozdíly ve struktuře a fungování obou systémů. Vymezila jejich možnosti, omezení a poskytla teoretické znalosti potřebné pro jejich nasazení.

V praktické části byla popsána implementace databázových systémů ve frameworku Symfony 2. Nejprve byly podány kroky potřebné k vytvoření projektu v tomto frameworku. Následně byla navržena modelová situace a převážná část praktické části se věnovala realizaci databázového řešení pro tento model. Výstupem implementace obou systémů byl poznatek, že ani jeden není pro implementaci výrazně jednodušší než jeho protějšek. Náročnost implementace by tedy neměla být brána v potaz při rozhodování, který systém použít pro konkrétní aplikaci.

V závěru práce byla nejprve vygenerována testovací data, na kterých byly vzápětí provedeny výkonové testy MySQL a MongoDB. Při vyhodnocení testu nebylo prokázáno, že by jeden nebo druhý systém byl jednoznačně rychlejší za všech okolností. Při některých typech dotazů byla výkonnější relační databáze MySQL, při jiných zase MongoDB. Při výběru systému je tedy potřeba nejdříve odhalit, jaké druhy dotazů bude potřeba v aplikaci vykonávat.

Zjednodušeně lze říci, že pokud je systém citlivý na transakční zpracování dotazů, je jednoznačně lepší zvolit MySQL, protože MongoDB transakce nepodporuje. Pokud není pevně dána struktura záznamů, které má databáze uchovávat, je vhodnější použít MongoDB databázi, která není svázána pevným schématem.

Vždy existuje také varianta obě databáze zkombinovat a čerpat tak silné stránky obou z nich. Doctrine dokáže oba systémy flexibilně kombinovat, takže není problém vytvořit co nejlepší řešení.

Přínos práce

Výsledky této práce nesou přínos především pro autora. Poznatky z této práce poskytují znalosti užitečné v oboru softwarové architektury při volbě technologií vhodných pro konkrétní aplikaci. Práce rovněž demonstruje základní implementaci obou systémů a tím umožňuje lepší pochopení, jaké problémy musí vývojář při realizaci řešení správy dat řešit.

Dalším přínosem práce je funkční projekt, na němž je nasazení obou systémů v praxi předvedeno.

Možnosti rozšíření

Díky funkčnímu projektu případnému zájemci nic nebrání experimentovat s různými dalšími druhy databázových dotazů a tím odhalit další silné a slabé stránky obou databázových systémů.

Jinou vhodnou možností rozšíření je přidat do porovnání další databázový systém nebo systémy, jako například v poslední době stále používanější klíč–hodnota databázový systém Redis.

Terminologický slovník

Termín	Zkratka	Vysvětlení [zdroj]
Apache		Web server pro Linux. [autor]
Atomicity-Consistency-Isolation-Durability	ACID	Principy, na které cílí relační databáze především prostřednictvím transakcí. Bližší vysvětlení je v podkapitole 2.1.4. [10]
Atribut		Vlastnost entity. [10]
Create-Read-Update-Delete	CRUD	Vytváření, čtení, úprava a mazání. Jedná se o základní operace při správě persistentních dat. [autor]
Database abstraction layer	DBAL	Databázová abstrakční vrstva slouží jako most mezi aplikační a databázovou vrstvou. [autor]
Symfony 2		PHP framework s architekturou Model-view-controller. [32]
Doctrine		Knihovna implementující ORM, ODM a DBAL. [autor]
Doména		Výčet všech možných hodnot, kterých může atribut entity nabývat. [10]
GNU General Public License	GNU GPL	Všeobecná veřejná licence. [13]
Hypertext preprocesor	PHP	Programovací jazyk určený především pro webové aplikace. [autor]
Index		Nástroj podobný rejstříku, který slouží ke zrychlení hledání dat v databázích. [autor]
InnoDB		Jeden z úložných formátů v MySQL. [12]
International Business Machines Corporation	IBM	Americká společnost, ve které vznikly relační databáze.

Linux		Operační systém, podobně jako Microsoft Windows nebo Mac OS X. [autor]
Model-view-controller	MVC	Architektura aplikací, která odděluje řídící vrstvu, prezentační vrstvu a business logiku aplikace. [32]
MongoDB		Jedna z dokumentových NoSQL databází. [autor]
MySQL		Jedna z mnoha relačních databází. [autor]
NoSQL		Druh databází, který se rozšířil jako alternativa k relačním databázím. [autor]
Object-Document-Mapping	ODM	Mapování dokumentů dokumentové databáze na objekty v aplikační vrstvě (PHP). [autor]
Object-Relational-Mapping	ORM	Mapování entit relační databáze na objekty v aplikační vrstvě (PHP). [autor]
Relational Database Management System	RDBMS	Systém řízení dat relační databáze. [11]
Structured query language	SQL	Dotazovací jazyk implementovaný většinou v relačních databázích. [autor]
Vagrant		Nástroj pro virtualizaci a snadné sdílení nastavení linuxového serveru. [autor]

Seznam literatury

[1] *Usage of operating systems for websites.* [Online] [cit. 5. březen 2016]. Dostupné z: http://w3techs.com/technologies/overview/operating_system/all

[2] *Usage statistics and market share of Unix for websites.* [Online] [cit. 5. březen 2016]. Dostupné z: http://w3techs.com/technologies/details/os-unix/all/all

[3] *Usage statistics and market share of Apache for websites.* [Online] [cit. 5. březen 2016] Dostupné z: http://w3techs.com/technologies/details/ws-apache/all/all

[4] *Usage statistics and market share of PHP for websites.* [Online] [cit. 5. březen 2016] Dostupné z: http://w3techs.com/technologies/details/pl-php/all/all

[5] *The Best PHP Framework for 2015: SitePoint Survey Results.* [Online] [cit. 5. březen 2016] Dostupné z: http://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/

[6] ČERMÁK, Miroslav. *Důvody k nasazení NoSQL, a speciálně MongoDB* [Bakalářská práce] Vysoká škola ekonomická v Praze, 2015.

[7] DOČKÁL, Jan. *Tvorba webových aplikací pomocí frameworku Symfony* [Bakalářská práce] Vysoká škola ekonomická v Praze, 2015.

[8] MROZEK, Jakub. *NoSQL databáze* [Bakalářská práce] Vysoká škola ekonomická v Praze, 2014.

[9] PETERA, Martin. *MongoDB* [Bakalářská práce] Vysoká škola ekonomická v Praze, 2013.

[10] SILBERSCHATZ, Abraham, Henry F KORTH a S SUDARSHAN. *Database system concepts*. 6th ed. Boston: McGraw-Hill, c2011. ISBN 978-0-07-352332-3.

[11] ASSAY, Matt. *Open source threates to eat the database market* [Online] [cit. 6. březen 2016] Dostupné z: http://www.infoworld.com/article/2916057/open-source-software/open-source-threatens-to-eat-the-database-market.html

[12] *MySQL 5.7 Reference Manual: History of MySQL.* [Online] [cit. 11. březen 2016] Dostupné z: http://dev.mysql.com/doc/refman/5.7/en/history.html

[13] BUYTAERT, Dries. *The history of MySQL AB.* [Online] [cit. 12. březen 2016] Dostupné z: http://buytaert.net/the-history-of-mysql-ab

[14] DYER, Russell J. *MySQL in a nutshell*. 2nd ed. Sebastopol, CA: O'Reilly, c2008. ISBN 978-0-596-51433-4.

[15] *MySQL 5.7 Reference Manual: Introcution to InnoDB.* [Online] [cit. 13. březen 2016] Dostupné z: http://dev.mysql.com/doc/refman/5.7/en/innodb-introduction.html

[16] TIWARI, Shashank. *Professional NoSQL*. Indianapolis, Ind.: John Wiley & Sons, Inc., 2011. ISBN 978-0-470-94224-6.

[17] DOLÁK, Ondřej. *Big data.* [Online] [cit. 18. březen 2016] Dostupné z: http://www.systemonline.cz/clanky/big-data.htm

[18] VAISH, Gaurav. *Getting started with NoSQL: your guide to the world and technology of NoSQL*. Birmingham: Packt Publishing, 2013. ISBN: 978-1-84969-4-988.

[19] STRAUCH, Christoff. *NoSQL Databases.*[Online] [cit. 19. březen 2016] Dostupné z: http://www.christof-strauch.de/nosqldbs.pdf

[20] *Database Systems Journal vol. IV, no. 2/2013.* [Online] [cit. 18. březen 2016] Dostupné z: http://www.dbjournal.ro/archive/12/12.pdf

[21] *Column-Oriented Database Systems* [Prezentace] [Online] [cit. 18. březen 2016] Dostupné z: http://www.cs.yale.edu/homes/dna/talks/Column_Store_Tutorial_VLDB09.pdf

[22] *The Google Pagerank Algorithm and How It Works.* [Online] [cit. 18. březen 2016] Dostupné z: http://www.cs.princeton.edu/~chazelle/courses/BIB/pagerank.htm

[23] BANKER, Kyle. *MongoDB in action*. Shelter Island, NY: Manning, c2012. ISBN 1935182870.

[24] *Query Documents.* [Online] [cit. 19. březen 2016] Dostupné z: https://docs.mongodb.org/manual/tutorial/query-documents/

[25] *Insert Documents.* [Online] [cit. 19. březen 2016] Dostupné z: https://docs.mongodb.org/manual/tutorial/insert-documents/

[26] *ObjectId.* [Online] [cit. 19. březen 2016] Dostupné z: https://docs.mongodb.org/manual/reference/method/ObjectId/

[27] *MongoDB Shell (mongo).* [Online] [cit. 19. březen 2016] Dostupné z: https://docs.mongodb.org/getting-started/shell/client/#start-mongo

[28] *Model One-To-Many Relationship with Embedded Documents* [Online] [cit. 19. březen 2016] Dostupné z: https://docs.mongodb.org/v3.0/tutorial/model-embedded-one-to-many-relationships-between-documents/#pattern

[29] *Why Vagrant?* [Online] [cit. 20. březen 2016] Dostupné z: https://www.vagrantup.com/docs/why-vagrant/

[30] *Vagrantfile* [Online] [cit. 25. březen 2016] Dostupné z: https://www.vagrantup.com/docs/vagrantfile/

[31] *Creating the Project* [Online] [cit. 26. březen 2016] Dostupné z: *http://symfony.com/doc/2.8/best_practices/creating-the-project.html*

[32] *Symfony: The book, version: 2.8.* [PDF] [Online] Dostupné z: http://symfony.com/pdf/Symfony_book_2.8.pdf

[33] *PHP Data Objects* [Online] [cit. 27. březen 2016] Dostupné z: http://php.net/manual/en/book.pdo.php

[34] *Welcome to Doctrine DBAL's Documentation!* [Online] [cit. 27. březen 2016] Dostupné z: http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/index.html

[35] *Welcome to Doctrine 2 ORM's documentation!* [Online] [cit. 27. březen 2016] Dostupné z: http://doctrine-orm.readthedocs.org/en/latest/

[36] *Doctrine MongoDB ODM's documentation!* [Online] [cit. 28. březen 2016] Dostupné z: http://doctrine-orm.readthedocs.org/projects/doctrine-mongodb-odm/en/latest/

[37] *DoctrineMongoDBBundle* [Online] [cit. 3. duben 2016] Dostupné z: http://symfony.com/doc/current/bundles/DoctrineMongoDBBundle/index.html

[38] *AliceBundle* [Online] [cit.10. duben 2016] Dostupné z: https://github.com/hautelook/AliceBundle#installation

[39] *Analyze Performance of Database Operations* [Online] [cit.10. duben 2016] Dostupné z: https://docs.mongodb.org/manual/tutorial/manage-the-database-profiler/

[40] *String Type Overview* [Online] [cit. 1. duben 2016] Dostupné z: http://dev.mysql.com/doc/refman/5.7/en/string-type-overview.html

[41] *Numeric Type Overview* [Online] [cit. 1. duben 2016] Dostupné z: http://dev.mysql.com/doc/refman/5.7/en/numeric-type-overview.html

[42] *Date and Time Type Overview* [Online] [cit. 1. duben 2016] Dostupné z: http://dev.mysql.com/doc/refman/5.7/en/date-and-time-type-overview.html

Seznam obrázků, tabulek a výpisů

Seznam obrázků

Obrázek 1: ER diagram prostředí školy (zdroj: autor) 10
Obrázek 2: Graf hran (modře) mezi uzly učitelů (červeně) a předmětů (zeleně). Graf byl vytvořen pomocí nástroje Microsoft Visio 2013 (zdroj: autor)..... 15
Obrázek 3: ER diagram modelové aplikace (zdroj: autor). 28

Seznam tabulek

Tabulka 1: Učitel (zdroj: autor)..... 5
Tabulka 2: Třída (zdroj: autor)..... 5
Tabulka 3: Student (zdroj: autor)..... 6
Tabulka 4: Vyučuje (zdroj: autor) 6
Tabulka 5: Předmět (zdroj: autor)..... 6
Tabulka 6: Ukázka úložiště klíč–hodnota (zdroj: autor)..... 13
Tabulka 7: Výsledky měření výkonu (zdroj: autor) 46

Seznam výpisů

Výpis 1: Příklad uložení dat v relační databázi (zdroj: autor)..... 14
Výpis 2: Příklad uložení dat ve sloupcové databázi (zdroj: autor). 14
Výpis 3: Dokumentová databáze ve formátu JSON (zdroj: autor). 16
Výpis 4: Identifikátor v dokumentu (zdroj: autor). 18
Výpis 5: Vazba 1–n pomocí odkazů (zdroj: autor)..... 19
Výpis 6: Vyhledávání podle odkazovaného dokumentu (zdroj: autor)..... 19
Výpis 7: Odkazovaná kolekce (zdroj: autor)..... 20
Výpis 8: Vnořené dokumenty v MongoDB (zdroj: autor). 20
Výpis 9: Sada příkazů pro inicializaci hotového projektu (zdroj: autor). 27
Výpis 10: Konfigurace MySQL (zdroj: autor)..... 27
Výpis 11: Soubor /src/AppBundle/Entity/Art.php v zestručněné podobě (zdroj: autor)..... 29
Výpis 12: Soubor src/AppBundle/Entity/Music.php ve stručné podobě (zdroj: autor)..... 30
Výpis 13: Vazba 1–n mezi entitami Author a Art, strana n (zdroj: autor)..... 31
Výpis 14: Vazba 1–n mezi entitami Author a Art, strana 1 (zdroj: autor)..... 31
Výpis 15: Sada příkazů pro údržbu relační databáze (zdroj: autor). 32
Výpis 16: Vytvoření nového záznamu pomocí ORM (zdroj: autor)..... 33
Výpis 17: Načtení záznamů v ORM metodami repositáře (zdroj: autor)..... 34
Výpis 18: Doctrine Query Builder a jazyk DQL (zdroj: autor). 35
Výpis 19: Závislost na Doctrine ODM v composer.json (zdroj: autor). 36
Výpis 20: Registrace balíčku ODM a anotací v Symfony (zdroj: autor)..... 36
Výpis 21: Konfigurace MongoDB (zdroj: autor)..... 36
Výpis 22: Soubor src/AppBundle/Document/Art.php ve stručné podobě (zdroj: autor). 37
Výpis 23: Vazba 1–n mezi entitami Author a Art (zdroj: autor)..... 38
Výpis 24: Vnořené dokumenty (zdroj: autor)..... 39

Výpis 25: Doctrine Query Builder pro dotazování nad MongoDB dokumenty (zdroj: autor). ... 40
Výpis 26: Konfigurace balíčku Alice (zdroj: autor). ... 43
Výpis 27: Definice podoby testovacích dat v balíčku Alice (zdroj: autor). ... 43
Výpis 28: Monitorování MongoDB (zdroj: autor). ... 45

Přílohy

Příloha A: Použité datové typy z MySQL

Název datového typu	Význam
INT(M[20])	Celé číslo v rozsahu -2147483648 až 2147483647 nebo 0 až 4294967295[21]. [41]
TINYINT(M)	Malá celá čísla v rozsahu −128 až 127 nebo 0 až 255. [41]
FLOAT	Číslo s pohyblivým množstvím desetinných míst. Povolený rozsah pro záporné hodnoty je -3.402823466E+38 až -1.175494351E-38. Pro kladné hodnoty 1.175494351E-38 až 3.402823466E+38. Skutečný rozsah může být o něco menší v závislosti na operačním systému. [41]
BOOL/BOOLEAN	V MySQL je reprezentován jako TINYINT(1), kde 0 odpovídá hodnotě FALSE a jakákoliv jiná hodnota odpovídá TRUE. Je-li atribut nepovinný, může TINYINT() nabývat také hodnoty NULL, která neodpovídá ani TRUE ani FALSE. [41]
VARCHAR(M)	Řetězec s povinnou maximální délkou. Povolený rozsah je 0 až 65 535 znaků. [40]
LONGTEXT	Velmi dlouhý řetězec pro 4 294 967 295 znaků (4 GB). [40]
DATE	Datum ve formátu 'YYYY-MM-DD'. [42]

[20] Parametr M určuje maximální počet míst. Například INT(5) tedy znamená, že nejvyšší možné číslo v dané doméně je 99999.

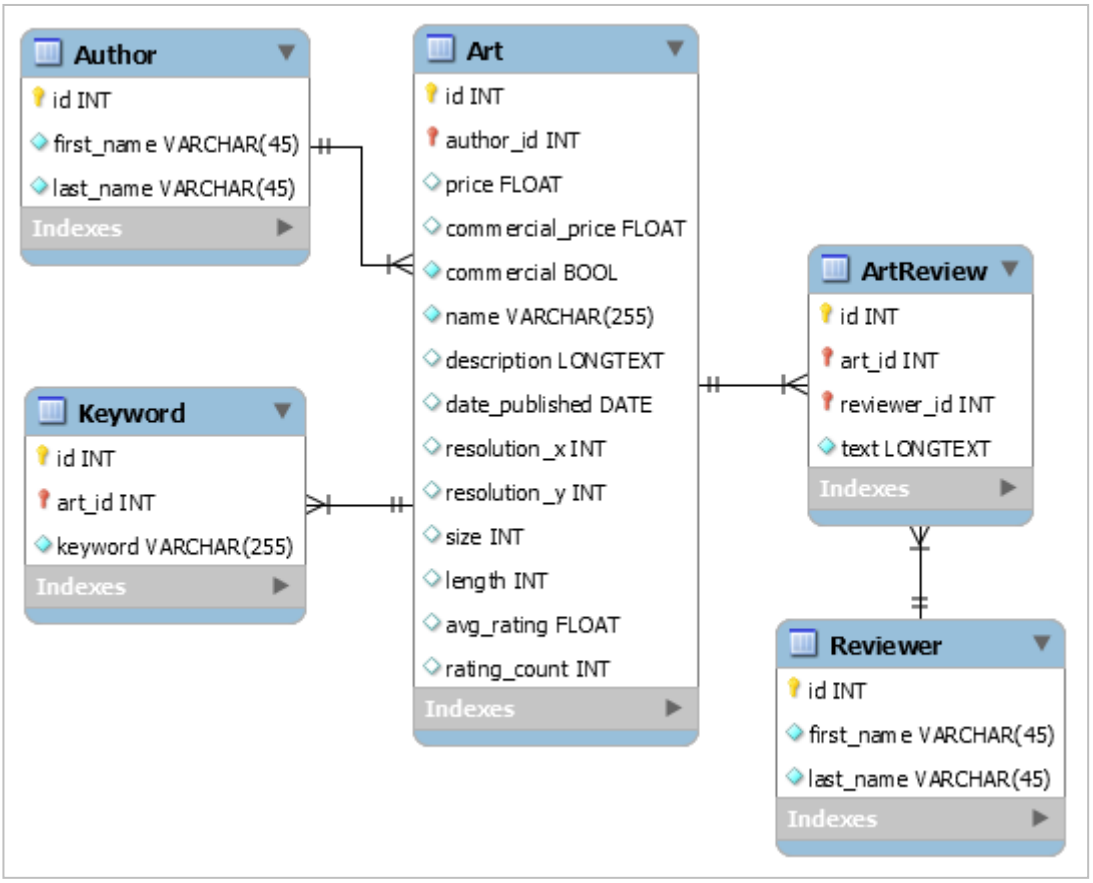
[21] U číselných datových typů lze v MySQL nastavit atribut SIGNED/UNSIGNED. UNSIGNED znamená, že jsou povoleny pouze nezáporné hodnoty. Výchozí hodnota je SIGNED.

Příloha B: Vagrantfile

```
1   # -*- mode: ruby -*-
2   # vi: set ft=ruby :
3
4   Vagrant.configure("2") do |config|
5
6   config.vm.box = "mokote/debian-7"
7   config.vm.network "private_network", ip: "192.168.10.12"
8   config.vm.hostname = "godsbox"
9   config.vm.synced_folder "www", "/vagrant-www", type: "nfs"
10
11  # Forwarded ports
12  config.vm.network "forwarded_port", guest: 80, host: 8000
13  config.vm.network "forwarded_port", guest: 3306, host: 33060
14
15  config.vm.provider "virtualbox" do |v|
16    v.memory = 5120
17    v.cpus = 2
18  end
19
20  # Provisioning Script
21  # --------------------
22  config.vm.provision "shell", path: "init.sh"
23
24  end
```

Zdroj: autor

Příloha C: Úplný ER diagram schématu



Zdroj: autor