VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

POROVNÁNÍ RELAČNÍCH A DOKUMENTOVÝCH DATABÁZOVÝCH SYSTÉMŮ PRO GENEALOGICKÉ ÚČELY

COMPARISON OF RELATION AND DOCUMENT DATABASE SYSTEMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MAREK PAKES**
AUTHOR

VEDOUCÍ PRÁCE **Ing. RADEK KOČÍ, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Pakes Marek**
Program: Informační technologie
Název: **Porovnání relačních a dokumentových databázových systémů pro genealogické účely**
Comparison of Relation and Document Database Systems
Kategorie: Databáze

Zadání:

1. Seznamte se s procesem digitalizace matričních údajů a prostudujte strukturu informací o svatbách a úmrtích.
2. Navrhněte strukturu dat pro ukládání informací z matrik.
3. Prostudujte problematiku relačních a dokumentových databázových systémů. Seznamte se s vybranými volně dostupnými databázovými systémy a po dohodě s vedoucím práce vyberte zástupce z každé skupiny.
4. Navrhněte model dat pro relační a dokumentové databázové systémy. Vytvořte příslušné databáze ve vybraných databázových systémech.
5. Vytvořte testovací datovou sadu a porovnejte výkonnost různých databázových operací.
6. Vyhodnoťte vhodnost jednotlivých typů databází pro zpracovávaná data.

Literatura:

- The MongoDB Documentation, dostupné on-line https://docs.mongodb.com/, říjen 2018.

Pro udělení zápočtu za první semestr je požadováno:

- První tři body zadání.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Kočí Radek, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Účelom tejto práce je porovnanie relačných a dokumentovo orientovaných databázových systémov pre problémovú doménu genealógie. Pre porovnanie boli použité databázové systémy PostgreSQL a MongoDB. Systémy sú porovnávané na základe analyzovaných matričných záznamov o svadbách a úmrtiach. Vygenerovanie testovacej sady dát a naplnenie databáz je realizované vlastným Node.js skriptom.

Abstract

The purpose of this thesis is to compare relational and document-oriented database systems for use in genealogy field. PostgreSQL and MongoDB database systems were used in pursuance of the comparison. The systems are compared according to vital records of marriage and death certificates. Testing data set is generated and filled in the databases by an own implementation of a Node.js script.

Kľúčové slová

genealógia, dokumentové databázy, MongoDB, PostgreSQL, relačné databázy

Keywords

Genealogy, Document-orietned database, MongoDB, PostgreSQL, Relational database

Citácia

PAKES, Marek. *Porovnání relačních a dokumentových databázových systémů pro genealogické účely.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Kočí, Ph.D.

Porovnání relačních a dokumentových databázových systémů pro genealogické účely

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Radka Kočího, Ph.D. Uvidedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Marek Pakes
14. mája 2019

Poďakovanie

Chcel by som sa poďakovať môjmu vedúcemu, pánovi Ing. Radkovi Kočímu, Ph.D. za metodickú a pedagogickú pomoc pri spracovaní tejto práce.

Obsah

Kapitola 1

Úvod

Základným kameňom genealógie sú dáta. Dáta, ktoré sa po stovky rokov uchovávali v knihách, ktoré sa nazývajú matriky. Táto práca sa bude zaoberať záznamami, ktoré matriky uchovávajú - konkrétne oddacími a úmrtnými, ich spracovaním a dôležitými krokmi pre ich zachovanie a jednoduchšie sprístupnenie verejnosti.

Málokto by veril tomu, že v roku 2019, keď máme elektromobily, kryptomeny, či povolanie „youtuber“, stále nie je možné jednoduchým spôsobom vyhľadať informácie o našich predkoch. Z prípadového hľadiska by takáto možnosť veľmi pomohla nielen školákovi, ktorý potrebuje vypracovať rodokmeň na domácu úlohu, ale aj napríklad sociológovi, ktorý skúma vplyv poklesu tradičných katolíckych svadieb na zvyšovanie volebných preferencií liberálnych strán v Hradci Králové.

V Českej republike sa dáta z archívov začali skenovať (digitalizovať) asi pred desiatimi rokmi. Problémom fotografií je, že sa v nich dajú vyhľadávať informácie len ich prezeraním. Taktiež je väčšina záznamov ťažko čitateľná, prípadne písaná v nemčine, či poľštine. Prezeranie naskenovaných matrík na internete je síce pohodlnejšie, ako si pol roka dopredu rezervovať návštevu bádateľne. Avšak, v dnešnej dobe máme na viac. Ďalším logickým krokom je preto ich postupné prepisovanie do digitálnej textovej formy.

Aj po takejto zdĺhavej „mravčej“ práci ešte nie sú dáta tam, kde ich chceme mať. Všetci by sme radi v týchto dátach vyhľadávali konkrétne kľúčové slová naprieč rôznymi archívmi v rôznych regiónoch. Z tohoto dôvodu je potrebný ďalší dôležitý krok, ktorým je vytvorenie databázového systému bežiaceho na serveri, kam sa aj spomínaný školák dokáže pripojiť pomocou webového klienta z pohodlia domova a vyhľadať, kedy sa narodila jeho prastará mama, pretože si na to dedko v pokročilom veku už nepamätá.

Konečne sme sa dopracovali k zmyslu tejto práce, a tým je vybrať vhodný databázový systém pre ukladanie spomínaných údajov. Pri státisícoch, až miliónoch záznamov s rôznymi vzťahmi je dôležité zvoliť, čo najefektívnejší spôsob ukladania. Relačné a dokumentové databázy sú jednými z najpoužívanejších typov databázových systémov v súčastnosti. Pre účely tejto práce boli zvolené systémy PostgreSQL a MongoDB, ktoré budú bližšie popísané v ďalšej časti tejto práce.

V nasledujúcich kapitolách budú vysvetlené základné koncepty vyššie spomenutých databázových systémov, ktoré ponúkajú rôzne spôsoby ukladania dát. V ďalšej časti bude popísaná analýza vstupných dát a databázových modelov, ktoré budú nakoniec testované z hľadiska rýchlosti vyhľadávania, vkladania veľkého množstva záznamov, pamäťovej náročnosti a ďalších kritérií. V závere práce budú zhrnuté výsledky testov a odporučenie jedného z vybraných adeptov pre účel uchovávania údajov z matričných archívov.

Kapitola 2

Databázové systémy

Táto kapitola sa bude venovať základným konceptom a z nich vyplývajúcim dôležitým vlastnostiam relačných a dokumentových databáz, kvôli ktorým boli zvolené ako vhodní adepti pre ukladanie matričných údajov a nasledovnému porovnaniu týchto vlastností na teoretickej úrovni.

2.1 Trendy v databázových systémoch

Medzi najpoužívanejšími relačnými databázami a databázami vôbec sa roky na prvých priečkach držia databázy Oracle a MySQL (viď. Obr. 2.1). Avšak pre túto prácu som sa rozhodol siahnuť po databázovom systéme **PostgreSQL**, ktorý sa po dlhých rokoch vývoja stáva najpopulárnejšou relačnou databázou pre nové projekty a taktiež ponúka mnoho zaujímavých funkcií, ktoré budú popísané v podkapitole 2.2.2. Popularita[1] je na portáli DB-Engines rátaná podľa vyhľadávaní na Google, Bing a Yandex, všeobecného záujmu o systém pomocou Google Trends, či otázkami na Stack Overflow a DBA Stack Exchange. Ďalšími kritériami sú ponuky práce a relevancia na profesionálnych (LinkedIn, Upwork), či sociálnych sieťach (Twitter) [5].

Rank			DBMS	Database Model	Score		
Apr 2019	Mar 2019	Apr 2018			Apr 2019	Mar 2019	Apr 2018
1.	1.	1.	Oracle	Relational, Multi-model	1279.94	+0.80	-9.85
2.	2.	2.	MySQL	Relational, Multi-model	1215.14	+16.89	-11.26
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1059.96	+12.11	-35.55
4.	4.	4.	PostgreSQL	Relational, Multi-model	478.72	+8.91	+83.25
5.	5.	5.	MongoDB	Document	401.98	+0.64	+60.57
6.	6.	6.	IBM Db2	Relational, Multi-model	176.05	-1.15	-12.89
7.	↑ 8.	↑ 9.	Redis	Key-value, Multi-model	146.38	+0.25	+16.27
8.	↑ 9.	8.	Elasticsearch	Search engine, Multi-model	146.00	+3.21	+14.64
9.	↓ 7.	↓ 7.	Microsoft Access	Relational	144.65	-1.55	+12.43
10.	10.	↑ 11.	SQLite	Relational	124.21	-0.66	+8.23

Obr. 2.1: **Top 10** najpopulárnejších databázových systémov k 04.2019 podľa portálu DB-engines. Z obrázku je vidieť, že za posledný rok (od 04.2018) sú práve PostgreSQL a MongoDB najviac rastúce databázové systémy v rámci popularity.

[1] https://db-engines.com/en/ranking

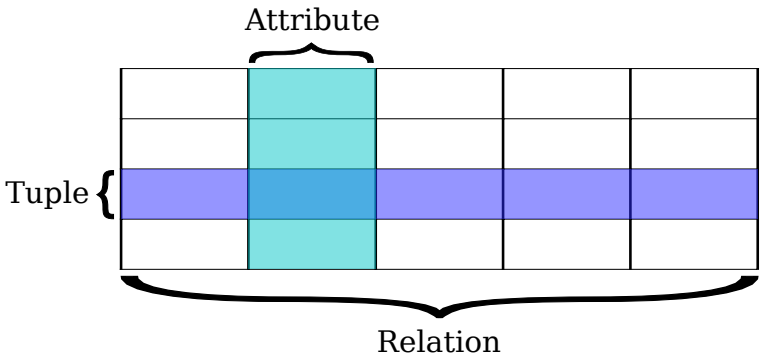
NoSQL

V poslednom desaťročí sa do popredia dostávajú aj tzv. **NoSQL** databázové systémy. Databázy tohoto typu sa zameriavajú na jednoduchosť dizajnu a lepšiu škálovateľnosť. Pôvodne boli definované ako databázové systémy, ktoré sú nerelačné (angl. *non-relational*), distribuované, open-source (voľne šíriteľné) a horizontálne škálovateľné [15]. Avšak dnes sa už prvá časť skratky namiesto doslovného významu z anglického *no*, resp. *not* častejšie označuje ako **NotOnly**SQL [19]. V súčasnosti využitie NoSQL databázových systémov rastie hlavne v oblasti webových aplikácií a softvérových riešeniach pre Big Data.

V tejto práci bol, ako druhý systém pre porovnanie vhodnosti pre uchovávanie genealogických dát, zvolený z radov NoSQL databáz systém **MongoDB**. Od prvej stabilnej verzie z roku 2009 jeho raketový vzostup nepostrehol len málokto. V kategórii NoSQL sa MongoDB radí medzi dokumentovo orientované databázy, ktoré budú bližšie popísané v sekcii 2.3.

2.2 Relačné databázy

Relačná databáza je taká databáza, ktorej položky sú organizované do kolekcie *vzťahov* a je založená na koncepte relačného modelu [4]. Tieto vzťahy sú reprezentované pomocou tabuliek, ktorých riadky obvykle chápeme ako záznamy. Stĺpce reprezentujú konkrétne atribúty jednotlivých záznamov. Niektoré stĺpce obsahujú takzvané cudzie kľúče, ktoré uchovávajú informáciu o relácii medzi jednotlivými záznamami v matematickom zmysle slova. Termín relačná databáza spolu s relačnou algebrou definoval Edgar F. Codd v jeho publikácii *A Relational Model of Data for Large Shared Data Banks* [2].



Obr. 2.2: **Tabuľka.** Tabuľka je základnou jednotkou konceptuálneho modelu relačnej databázy. Pojem *Tuple* predstavuje riadok tabuľky a konkrétny záznam. *Attribute* - stĺpec tabuľky predstavuje atomickú hodnotu jedného atribútu záznamu. *Relation*, teda vzťah predstavuje tabuľku ako množinu záznamov, ktoré majú rovnaké atribúty.

2.2.1 Modelovanie a výkon

Ako už bolo spomenuté, základným princípom relačných databáz sú vzťahy. Dôležitým faktorom pri navrhovaní relačných databáz je vytvorenie efektívneho modelu, ktorý bude spĺňať požiadavky aplikácie a zároveň striktné kritériá schémy relačnej databázy.

Vytvorenie schémy

Schému relačnej databázy môžeme získať dvomi spôsobmi:

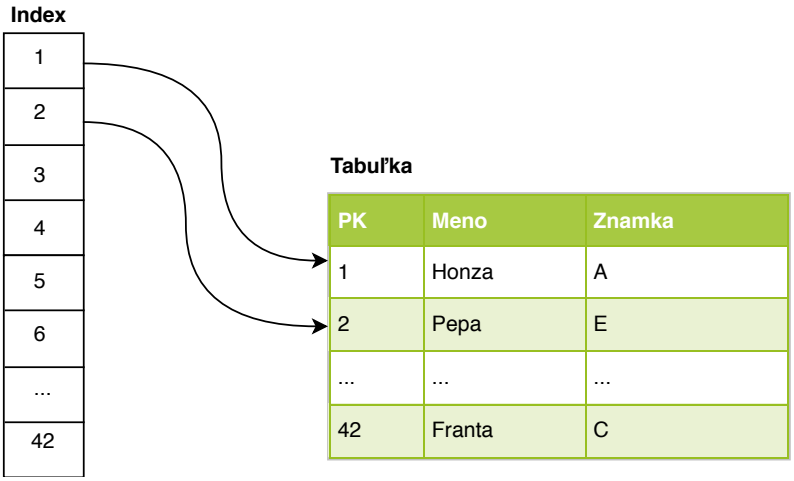
- vytvorením **konceptuálneho modelu** a jeho transformáciou
- použitím **normalizácie** s tým, že na počiatku predpokladáme, že všetky informácie budú uložené v jednej tabulke, ktorú postupnými krokmi prevádzame do vyšších normálnych foriem [21]

Výkon dotazovania

Výkon dotazovania ovplyvňuje primárne komplexita navrhnutej štruktúry dát. Jedným zo spôsobov ktorým môžeme výkonnosť operácií (nielen) v relačnej databáze zvýšiť je použitie indexov.

Indexy v mnohých prípadoch niekoľkonásobne zvyšujú výkon databázy tým, že povoľujú databázovému serveru vyhľadať a získať konkrétne riadky tabuliek oveľa rýchlejšie než bez vytvoreného indexu daného riadku. Problémom indexov je, že pridávajú určitú réžiu do databázového systému ako celku, takže by sa mali používať rozumne.

Bez indexu musí databázový server začať v prvom riadku a čítať cez celú tabuľku, aby našiel hľadaný záznam. To znamená, že čím je tabuľka väčšia, tým je operácia nákladnejšia.



Obr. 2.3: **Index.** Databázový index je kolekcia ukazovateľov. Každý ukazovateľ ukazuje na celý záznam v tabuľke uloženej v databáze. Indexy umožňujú veľmi efektívne vyhľadávanie založené na známej množine vyhľadávaných hodnôt.

2.2.2 PostgreSQL

PostgreSQL je výkonný, voľne dostupný objektovo-relačný databázový systém aktívne vyvíjaný vyše 30 rokov. Tento systém používa a rozširuje jazyk SQL a zároveň kombinuje mnoho funkcií, ktoré bezpečne ukladajú a škálujú aj najkomplikovanejšie dáta. Kladie dôraz na rozšíriteľnosť a dodržovanie štandardov. Zárukou je kompatibilita s ACID (angl. *Atomicity, Consistency, Isolation, Durability*), čo sú vlastnosti nutné pre transakčné spracovanie [8].

Rozdiel oproti klasickým relačným databázam (napr. MySQL) je minimálny. PostgreSQL je relačný databázový systém a používa tabuľky ako hlavné komponenty. Odlišnosťou oproti starším verziám MySQL je podpora niektorých NoSQL funkcií (z toho dôvodu objektovo-relačný databázový systém). Táto odlišnosť už súčastnosti neplatí, nakoľko sú NoSQL koncepty ako napríklad dokumentový sklad, či formát JSON pre ukladanie dát už aj súčasťou „klasického“ MySQL od verzie 5.7.

Tento databázový systém je vydaný pod MIT licenciou, takže sa jedná o open-source software. PostgreSQL nie je vlastnený jednou firmou. Na vývoji sa podieľa globálna komunita vývojárov a firiem.

História PostgreSQL

Predchodcom PostgreSQL bol Ingres, vyvíjaný na univerzite Berkeley v Kalifornii v rokoch 1977-1985, z ktorého sa neskôr stal jeden z prvých komerčne úspešných relačných databázových serverov. Jeho nástupcom sa stal objektovo-relačný server Postgres (1986-1994), ktorý bol časom obohatený o schopnosti SQL dvomi absolventami spomínanej univerzity - Jolly Chen a Andrewom Yu a bol premenovaný na Postgres95.

V roku 1996 bol prejavený veľký záujem o open-source SQL databázu, čo malo za následok nazdieľanie medzi vyše tisíc užívateľov, ktorí mali opravovať a vylepšovať zdrojový kód. Jolly Chen sa neskôr vyjadril, že tento projekt potrebuje pár ľudí s veľa voľným časom, nie veľa ľudí s málom času. Narážal tým na vtedajších 250,000 riadkov kódu v jazyku C. Ku koncu roka bol vytvorený svetový vývojársky tím a názov sa zmenil na dnešné PostgreSQL. Nové verzie začali vychádzať takmer každý štvrťrok. Ako spomína jeden z prvých členov vývojárskeho tímu a autor knihy, z ktorej čerpá veľká časť tejto práce, Bruce Momijan: „Nikdy sme sa nepokúšali sledovať viac agresívny spôsob vývoja s častejšími novými verziami. Databázový systém nie je ako textový procesor alebo hra, ktoré môžete jednoducho reštartovať, ak vznikne problém. Databázy sú viacužívateľské (angl. *multiuser*) a udržiavajú užívateľské dáta vo svojom vnútri, preto musia byť tak spoľahlivé, ako sa len dá.“ [11] Počas vývoja prišli vedúci vývojári prísť s mnohými nápadmi, ktoré pomohli posunúť gigantický projekt tam, kde je teraz - na 4. priečku vo svetovom rebríčku popularity (viď. Obr. 2.1).

Indexovanie v PostgreSQL

Interne PostgreSQL uchováva dáta v súboroch operačného systému [12]. Každá tabuľka má vlastný súbor obsahujúci aj dáta záznamov. Index je oddelený súbor, ktorý je zoradený podľa jedného alebo viacerých stĺpcov. Uchováva ukazovatele na súbor obsahujúci tabuľku a umožňuje rýchly prístup ku špecifickým hodnotám tabuľky.

PostgreSQL okrem indexov primárnych kľúčov nevytvára ďalšie indexy automaticky. Namiesto toho by ich mal používateľ vytvoriť na miestach, na ktorých očakáva časté filtrovanie pomocou klauzule `WHERE`.

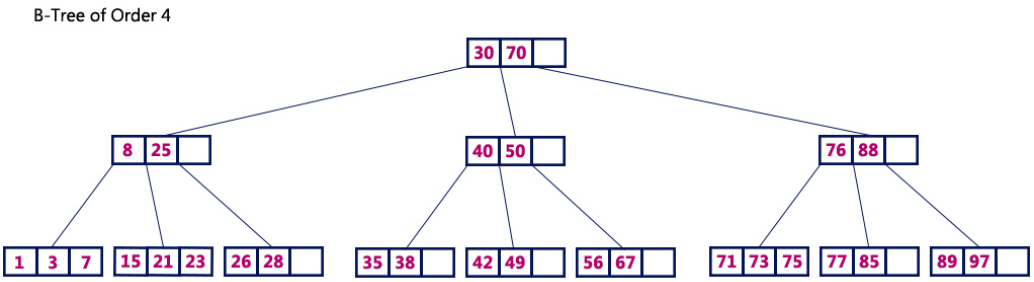
PostgreSQL prináša vstavanú podporu pre B-stromy a hashovacie indexy. Ďalej podporuje výrazové indexy (angl. expression index), ktoré môžu byť vytvorené indexom výsledku výrazu alebo funkcie namiesto hodnoty stĺpca. Čiastočný index (angl. partial index) môže byť vytvorený len pre časť tabuľky.

B-strom

B-strom (viď. obrázok[2] 2.4) je špeciálny prípad dobre známej vyhľadávacej štruktúry - strom, ktorý je formovaný z uzlov. Každý uzol, okrem špeciálneho koreňového uzla, má jeden rodičovský uzol a niekoľko listov (potomkov). B-strom je samostatne vyvážená stromová štruktúra, ktorá uchováva dáta zoradené a umožňuje vyhľadávania, sekvenčné prístupy, vkladanie a vymazávanie v logaritmickom čase. Je geralizáciou binárneho vyhľadávacieho stromu, v ktorom uzol môže mať viac ako dvoch potomkov. Narozdiel od iných samostatne vyvážených stromových štruktúr sú B-stromy vhodné pre skladovacie systémy, ktoré zapisujú a čítajú veľké bloky dát. Bežne sú využívané okrem databáz aj v súborových systémoch [7].

Základnou myšlienkou pri používaní B-stromov je redukovanie počtu prístupov na disk. Operácie ako vyhľadávanie, vkladanie, mazanie, min, max, atď. vyžadujú $O(h)$ prístupov na disk, pričom h je výška stromu. B-stromy sú namiesto do výšky rozložené čo najviac do šírky vkladaním maxima možných kľúčov do uzlu. Veľkosť uzlu sa pritom pohybuje väčšinou rovnomerne s veľkosťou bloku disku. Keďže výška stromu je nižšia, počet diskových prístupov pre väčšinu operácií je mnohonásobne znížený oproti vyváženým binárnym vyhľadávacím stromom ako napr. AVL strom. Vlastnosti B-stromu sú nasledovné:

1. Všetky listy sú na rovnakej úrovni.
2. Je definovaný pomocou minimálneho stupňa t, pričom veľkosť t závisí od veľkosti bloku disku.
3. Každý uzol okrem koreňového musí obsahovať aspoň $t - 1$ kľúčov (koreň môže obsahovať minimálne 1 kľúč).
4. Uzol môže obsahovať najviac $2t - 1$ kľúčov.
5. Počet potomkov uzlu je rovný počtu jeho kľúčov + 1.
6. Časová zložitosť vyhľadávania je $O(log_n)$.
7. Rastie a zmenšuje sa od koreňového uzlu.
8. Kľúče uzlov sú zoradené vzostupne.



Obr. 2.4: **B-strom** štvrtého rádu.

[2]Obrázok B-tree of Order 4 vypožičaný z http://btechsmartclass.com/data_structures/b-trees.html

2.3 Dokumentové databázy

Dokumentovo orientovaná databáza alebo dokumentový sklad je typ databázového systému navrhnutý pre ukladanie a spravovanie dokumentovo orientovaných informácií, tiež známych ako pološtruktúrované (angl. *semi-structured*) dáta. Tento typ databázového systému, narozdiel od relačných databáz, nie je založený na vzťahoch (angl. *relations*). Uložené dáta teda nie sú formované do tabuliek, ale do dokumentov, ktoré sú štruktúrou podobné objektom vo formáte JSON [6]. Koncept prináša programátorom jednoduchší spôsob ukladania a vyhľadávania dát v databáze použitím rovnakého formátu aký používajú v kóde aplikácie.

Flexibilná, pološtruktúrovaná a hierarchická povaha dokumentov a dokumentových databáz dovoľuje vývoj súbežne s potrebami aplikácie. Dokumentový model dobre pracuje s rôznymi prípadmi použitia, ako napríklad katalógy, profily užívateľov a CMS (Content Management System), kde je každý dokument unikátny a vyvíja sa postupom času. Dokumentové databázy tiež ponúkajú flexibilné indexovanie, rýchle ad hoc vyhľadávanie a rôzne analýzy naprieč kolekciami dokumentov.

Dokument

Základným konceptom každej dokumentovej databázy je **dokument**, ktorý môžu rôzne dokumentové databázy používať v rôznom formáte. Štandardné formáty pre ukladanie dokumentov sú JSON, BSON, YAML, či XML.

Narozdiel od relačných databáz, kde každý záznam obsahuje rovnaké polia, a kde nevyplnené polia sú prázdne, v dokumente (zázname) prázne polia neexistujú. Spôsob ukladania dát v dokumentovom sklade tým pádom dovoľuje pridávať nové údaje niektorým záznamom bez toho, aby bolo nutné pridávať pole pre tieto údaje každému ďalšiemu dokumentu v príslušnej kolekcii.

Momentálne najpoužívanejší databázový systém tohoto typu je MongoDB. Ďalšími sú napríklad CouchDB, OrientDB a iné. MongoDB je open source software vydaný pod GNU Affero General Public License[3] a Apache licenciami[4].

2.3.1 MongoDB

Názov je odvodený od anglického *„humongous“* preložiteľné ako **obrovský**, ktorý naznačuje, že je tento databázový systém schopný uchovávať veľké množstvo dát efektívnym spôsobom, s čím môžu mať jeho náprotivky relačného typu v niektorých prípadoch použitia problémy. Primárny dôvod pre odlúčenie sa od relačného modelu je jednoduchšie škálovanie, no tento databázový systém ponúka aj ďalšie výhody.

Ako to celé funguje?

Dokumenty, ktoré môžeme chápať ako konkrétne záznamy (riadky) v tabuľkách relačných databáz, sú v MongoDB združené v **kolekciách**, ktoré zase môžeme interpretovať ako tabuľky, či entitné množiny relačného modelu. Narozdiel od záznamov v tabuľkách relačných databáz, dokumenty nepodliehajú jednotnej schéme. Z toho vyplýva, že jedna kolekcia môže v sebe uchovávať dokumenty, ktoré majú odlišné atribúty. Táto na prvý pohľad „výhoda“ oproti relačným databázam, ale môže priniesť u návrhára databázy neistotu v tom, že všetky dáta dodržiavajú potrebný formát, preto je potrebné dobre si premyslieť aká kolekcia bude

[3] https://cs.wikipedia.org/wiki/Affero_General_Public_License
[4] https://cs.wikipedia.org/wiki/Apache_Licence

uchovávať konkrétne dokumenty. V konečnom dôsledku má voľba dokumentovej databázy vďaka tejto vlastnosti priniesť naozaj flexibilné riešenie pre dáta, ktoré sa postupom času vyvíjajú.

Vo všeobecnosti NoSQL naznačuje, že sa v tejto sfére nevyužívajú vzťahy. Pri používaní dokumentových databáz sa namiesto rozdeľovania informácií do mnohých tabuliek s atomickými stĺpcami naopak sústredíme na zhromažďovanie všetkých potrebných informácií na jednom mieste.

Základnou myšlienkou NoSQL je nestarať sa o vzťahy medzi rôznymi entitami, ktoré je nutné spájať ako v prípade relačných databáz „JOIN-ovaním“ tabuliek, ale mať na to extrémne rýchle a efektívne dotazy (queries), ktoré všetko potrebné získajú z jedného miesta. Samozrejmosťou pri takejto štruktúre sú duplicity dát, t.j. väčšia veľkosť a zložitejšia úprava. Informáciu je potrebné upraviť na všetkých miestach, kde sa vyskytuje. Všeobecne je toto riešenie vhodné pre aplikácie, ktoré využívajú oveľa viacej čítaní dát ako ich úpravy.

Indexovanie dokumentov

Ako v každej databáze, indexy sú dôležité pre rýchlejšie vyhľadávanie. Bez nich musí systém prehľadať celú kolekciu (angl. *collection scan*), čo znamená, že musí prejsť každým dokumentom v kolekcii, aby na základe požadovaných filtrov vybral tie dokumenty, ktorých atribúty vyhovujú zadaným parametrom vyhľadávania. Ak existuje príslušný index pre konkrétny atribút (aj vnorený), databázový systém ho využije pre zníženie počtu dokumentov, ktoré musí prehľadať.

Indexy v MongoDB [13] sú špeciálne dátové štruktúry - B-stromy (rovnako ako v PostgreSQL 2.4), ktoré ukladajú malú časť z dátovej množiny danej kolekcie v jednoducho prechádzateľnej forme. Ukladajú zoradené hodnoty polí (atribútov). Vo svojej podstate sú indexy v MongoDB podobné ako indexy v ktorejkoľvek inej databáze. MongoDB definuje indexy na úrovni kolekcií a podporuje sekundárne indexy na akomkoľvek poli (atribúte), resp. zanorenom poli dokumentov v kolekcii.

Základné indexy sú `_id` kolekcií. Tento index pomáha predhádzať vloženiu dvoch dokumentov s rovnakou hodnotou poľa `_id`. Je to v podstate primárny kľúč v relačných databázach a je vytváraný automaticky. Ďalšie typy indexov sú:

- *single field index*, čo je užívateľsky definovaný index pre jeden atribút (najčastejšie využitie),
- *compound index* zložený z viacerých atribútov,
- *multikey index* určený pre indexovanie hodnôt v poli,
- *geospatial index* využívaný pre súradnice,
- *text index* pre vyhľadávanie v textoch,
- *hashed index* na podporu tzv. *hashed sharding*, ktorý je využívaný pri rozdeľovanie dát pri horizontálnom škálovaní do viacerých clustrov. Tieto indexy sú viac náhodne distribuované, no sú obmedzené len na podporu porovnávania hodnôt a nemôžu byť využité na filtrovanie záznamov pomocou intervalu.

V tejto práci bude pre zvýšenie výkonu vyhľadávania v MongoDB použitý základný `_id` generovaný automaticky pre každý dokument v kolekcii. Ďalej, zo sekundárnych indexov, budú vytvorené `single field` indexy, pomocou ktorých budú indexované potenciálne

často využívané atribúty pre filtrovanie záznamov v kolekcii a `multikey` indexy, ktorými databáza indexuje položky vnútri polí. Atribút dokumentu, ktorý má hodnoty uložené v poli je databázou automaticky identifikovaný. MongoDB zaindexuje každú položku poľa (aj špecifikované atribútyv norených objektov v poli), čiže pri vytváraní indexov nie je potrebné starať sa o to, či pole bude jedno, dvoj alebo sto prvkové - index užívateľ vytvára rovnako ako pre atomické položky. Avšak je veľmi dôležité uvedomiť si, že každá kolekcia má limitovaný počet indexov. V momentálnej verzii MongoDB podporuje maximálne 64 indexov pre jednu kolekciu. Reálne by sa každá aplikácia do takéhoto počtu mala zmestiť, pretože ako už bolo spomenuté, indexovať je potrebné len atribúty, ktoré sa vyskytujú v dopytovacích filtroch frekventovane.

Databázové operácie

Metódy v MongoDB používajú JavaScript[5]. V tejto práci budú prevažne využité metódy dvoch typov:

- **metódy nad kolekciami - *Collection methods*** - CRUD operácie, vytváranie a vymazávanie indexov, atď.
- **metódy nad kurzormi - *Cursor methods*** - tieto metódy modifikujú spôsob, akým je dopytovací príkaz (angl. *query*) vykonaný. Dopĺňajú metódy nad kolekciami o napr. `.explain()`, `.limit()`, `.sort()`, `.map()` a iné možnosti modifikovania výstupu operácie.

Kurzor je ukazovateľ na množinu výsledkov príkazu na prehľadanie databázy. Konkrétny výsledok sa získa iterovaním kurzoru [3].

2.4 Porovnanie vlastností databáz

V tejto sekcii budú popísané odlišnosti základných vlastností relačných a dokumentových databáz. Zo špecifických vlastností PostgreSQL a MongoDB bude poukázané na základný užívateľský rozdiel, ktorým je používanie odlišných dotazovacích jazykov.

2.4.1 Vzťahy

V niektorých prípadoch použitia môžu byť vzťahy obrovskou výhodou. Ak je napríklad potrebné uchovávať dáta ako napr. produkt a užívateľ, ktoré často modifikujeme. V takom prípade môže byť podstatne obtiažnejšie aktualizovať takéto dáta v niekoľkých kolekciách v „NoSQL svete“, nakoľko sú mnohokrát duplicitné. Je jednoduchiše využiť viac štruktúrovaný SQL prístup, kde je potrebné aktualizovať dáta len v tabuľke `uzivatel` a následne každý dopyt na vytvorenie objednávky, ktorý vyberá informácie o užívateľovi bude automaticky vyberať dáta už aktualizovaného užívateľa pretože sú ukladané a spravované na jedinom mieste. Nevýhodou vzťahov môžu byť komplexné vyhľadávania, ktoré potrebujú dáta z viacerých tabuliek (veľké množstvo JOIN operácií). Takéto prípady majú NoSQL databázy vyriešené omnoho elegantnejšie, pretože skladujú všetky potrebné dáta už na jednom mieste spojené v jednej kolekcii a nie je potrebné ich spájať v konkrétnom vyhľadávacom príkaze.

[5] https://docs.mongodb.com/manual/reference/method/js-collection/

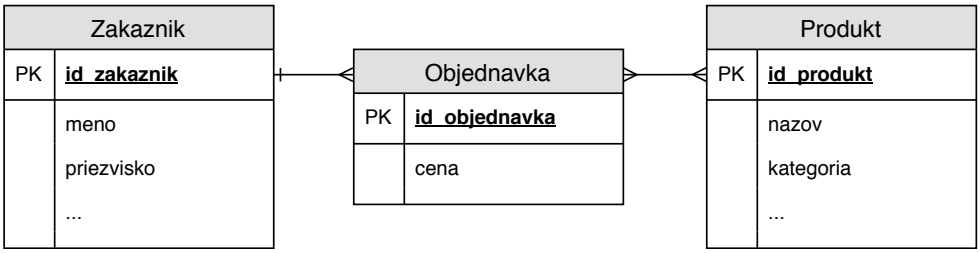
Žiadne, resp. takmer žiadne vzťahy sú výhodné pre veľké množstvo čítaní. Na druhej strane nevýhodou môžu byť zápisy, ktoré ovplyvňujú viaceré kolekcie kvôli duplikovaniu údajov namiesto vzťahov. Pre aplikáciu, v ktorej je potrebné často upravovať pevne viazané dáta ukladané vo viacerých kolekciách dokumentová databáza nie je najvhodnejším riešením. V relačných databázach je množstvo čitacích a zapisovacích dopytov za sekundu limitované, no je potrebné podotknúť, že sa tento údaj týka tisícov, až desaťtisícov vyhľadávaní v jednom momente. Vždy záleží na komplexite vyhľadávacích filtrov, agregácie alebo na množstve zjednocovaní tabuliek, tzv. „joinov“ v prípade relačných databáz. Kolekcií v dokumentovej databáze zvykne byť oveľa menej ako tabuliek v relačnej databáze, pretože princípom kolekcií je zoskupovať spolu dáta, ktoré spolu súvisia.

2.4.2 Schéma

S použitím vzťahov úzko súvisí definovanie schémy databázy. V prípade relačných databáz je potrebná striktná schéma. Dokumentové databázy sa vyznačujú tým, že sú tzv. *„schemaless“*.

Relačný model

V relačnom databázovom systéme je pred naplnením záznamov do databázy potrebné definovať schému, ktorá popisuje štruktúru dát popísanú formálnym jazykom, ktorý databáza podporuje. Takáto schéma je návrhom tabuliek v databáze a vzťahov medzi nimi. V tabuľkách je potrebné definovať obmedzenia v riadkoch a pomenovaných stĺpcoch, a tiež dátové typy, ktoré budú v každom stĺpci uložené. Ukážku relačného modelu môžeme vidieť na obrázku 2.5 nižšie.

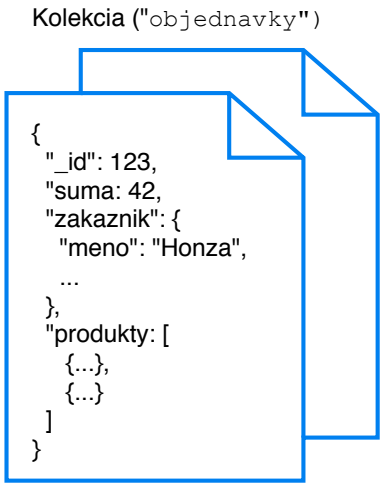


Obr. 2.5: **Relačný model.** V relačnom modeli je nutné oddeliť všetky entitné množiny do tabuliek s atomickými atribútmi (žiadne štrukturované dátové typy). Kvôli vlastnostiam relačného modelu, musíme objednávku prepojiť s tabuľkami zákazníka a produktu. V prípade vzťahu produktu a objednávke vidíme, že ide o M-ku-N vzťah, kvôli ktorému by sme potrebovali vytvoriť ďalšiu (štvrtú), tzv. väzobnú tabuľku.

Dokumentový model

V dokumentovo orientovaných databázach neexistuje preddefinovaná schéma. Kľúče a hodnoty dokumentov nie su fixného typu alebo veľkosti. Nevzniká tu teda potreba modelovať diagramy, či riešiť normalizáciu. Jediné, čo je pri návrhu dokumentovej databázy vhodné je definovanie kolekcií. Kolekcie budú totiž v sebe ukladať množinu dokumentov, ktoré spolu súvisia. Je vhodné poznamenať, že kolekcie majú dynamickú schému. [1] Znamená to, že dokumenty v kolekcii môžu mať variabilný tvar. Dokumenty sú záznamy popisujúce dáta

kolekcie a zároveň sú to už konkrétne dáta. Môžu byť tak komplexné, ako si návrhár databázy zvolí, keďže v sebe dovoľujú ukladať aj vnorené komplexné dátové štruktúry. Pri vnorených dátach je ale dôležité myslieť na skutočnosť, že čím viac sú dáta zanorené, tým zložitejšie je ich vyhľadávanie. Či už v zmysle zložitosti vytvárania vyhľadávacieho príkazu, ale aj rýchlosti vyhľadania dopytovaných dát. Rozdiel oproti relačnému modelu môžeme vidieť na obrázku 2.6.



Obr. 2.6: **Dokumentový model.** Kolekcia dokumentov obsahuje sémanticky podobné položky, ktoré môžu, ale nemusia mať všetky atribúty rovnaké. V takomto modeli všetky potrebné údaje k jednej objednávke nájdeme v jednom dokumente. Atribútmi môžu byť jednoduché, ale aj štrukturované dátové typy vrátane vnorených dokumentov, ktoré môžu byť uložené v poli.

Výhody a nevýhody pevnej schémy

Pevná schéma môže byť výhodou aj nevýhodou. Výhodou je predikovateľnosť dát, resp. schopnosť vždy vedieť, čo budú dáta obsahovať (atribúty záznamov). Nevýhodou pevnej schémy je nemožnosť flexibility dát. Ak má systém pevne stanovenú schému, každý nový záznam ju musí spĺňať, čo znamená, že nie je možné vkladať záznamy, ktoré schému porušujú. Pridanie záznamu s atribútom, ktorý entitná množina nemá definovaný nie je možné, kým daný atribút nepridáme všetkým entitám, ktoré obsahuje. To znamená, že kvôli pridaniu jedného nového záznamu s novým atribútom by sme museli každému starému záznamu v tabuľke vyplniť tento atribút prázdnou hodnotou. Dokumentové databázy nemajú pevnú schému. Pridaním dokumentu s novým atribútom v žiadnom prípade neovplyvní ostatné dokumenty kolekcie. Na druhej strane, nevýhodou takéhoto prístupu je nemožnosť sa spoľahnúť, že konkrétny záznam (dokument) bude mať konkrétny atribút - neexistuje schéma, ktorá by to vynucovala.

2.4.3 Škálovanie

Možnosť škálovať nie je nutnou výhodou, ale vždy je dobré ju mať po ruke. Vždy záleží veľkosti databáz a schopnosti predpovedať, do akej miery budú dáta pribúdať. Rozdiel medzi *horizontálnym* a *vertikálnym* škálovaním je triviálny. Horizontálnym škálovaním pridávame

viac serverov, teda je teoreticky neobmedzené. Je to prakticky zväčšovanie výpočtového výkonu a priestoru pre dáta pridávaním ďalších staníc. Táto možnosť je pre relačné databázy veľmi obtiažna, resp. nemožná. Dôvodom je, že sa dáta nedajú jednoducho rozdeliť medzi viac serverov kvôli striktnej schéme a transakčnosti (ACID). Vertikálne škálovanie je alternatíva. Znamená zväčšovanie výkonu už existujúceho serveru (horné ohraničenie maximálnym vylepšením jedného počítača, resp. serveru). Vertikálne škálovanie je teda obmedzené a mnohokrát drahšie, pretože je jednoduchšie nakúpiť viac lacných serverov ako vylepšovať jeden masívny a drahý stroj. NoSQL databázy v tomto bode vyhrávajú, pretože je jednoduché ich škálovať horizontálne vďaka spôsobu, akým sú dáta ukladané (žiadne vzťahy, iba samostatné (angl. „standalone“) kolekcie).

V tabuľke 2.1 môžeme vidieť zhrnuté porovnanie základných vlastností relačných a dokumentových databáz vo všeobecnosti.

Tabuľka 2.1: Porovnanie vlastností relačných a dokumentových databáz

Relačná	**Dokumentová**
prísna schéma	žiadna predpísaná schéma, resp. dynamická schéma kolekcií
využíva vzťahy	žiadne (príp. minimum) vzťahov
dáta distribuované naprieč mnohými tabuľkami	dáta zjednotené (vnorené) v malom počte kolekcií
žiadna duplicita dát	pomerne častá duplicita dát (závisí od požiadavkov aplikácie)
len vertikálne škálovanie	horizontálne aj vertikálne škálovanie
limitované množstvo dopytov za sekundu	veľký výkon pre masové čítania a zápisy

2.4.4 Syntax operácií

Viditeľným rozdielom pri práci s týmito dvomi databázami je použitie jazyka pre operácie s nimi. Dobrému programátorovi by nemalo prekážať naučiť sa používať akýkoľvek nástroj, či jazyk, no aj tak je vhodné poukázať aj na tieto rozdiely.

PostgreSQL

CRUD (*Create*, *Read*, *Update*, *Delete*) operácie v PostgreSQL sú vykonávané v rovnakom jazyku ako v bežnej relačnej databáze - používaním SQL (Structured Query Language):

Výpis 2.1: Vybrať všetky záznamy z tabuľky `Agent`

```
SELECT * FROM "Agent";
```

Výpis 2.2: Vloženie záznamu do tabuľky `Agent`

```
INSERT INTO Agent(agent_id, name, surname) VALUES (7, 'James', 'Bond');
```

Výpis 2.3: Úprava záznamov v tabuľke `Agent`

```
UPDATE Agent SET retired="yes" WHERE agent_id < 7;
```

MongoDB

Ako bolo už popísané v predošlej sekcii, operácie nad kolekciami v MongoDB sú písané notáciou jazyka JavaScript:

Výpis 2.4: Vyhľadanie všetkých dokumentov v kolekcii `agents`

```
db.agents.find()
```

Výpis 2.5: Vloženie dokumentu do kolekcie `agents`

```
db.agents.insert({
  agent_id: 7,
  name: "James",
  surname: "Bond"
})
```

Výpis 2.6: Úprava dokumentov kolekcie `agents`

```
db.agents.update(
  {agent_id: {$lt: 7}},
  {$set: {retired: "yes"}},
  {multi: true}
)
```

PostgreSQL a MongoDB v kocke

V nasledujúcej tabuľke sú pre prehľadnosť spísané ekvivalencie dát, ich vlastnosti, použitie jazykov pre vytváranie dotazov (angl. *queries*) a krátky zoznam niektorých technologických gigantov používajúcich PostgreSQL[6] a MongoDB[7].

Tabuľka 2.2: Spôsob, operácie a

	PostgreSQL	**MongoDB**
Entitná množina	tabuľka	kolekcia
Entita	riadok	dokument
Atribút	stĺpec	pole (field)
Vlastnosti atribútov	atomické	atomické aj komplexné
Dotazovací jazyk	SQL	JavaScript
Kto ich používa	Geni.com, Reddit, Skype, BASF	Facebook, Google, SAP, eBay

Kto je teda víťaz? Striktné relačné alebo voľnejšie dokumentové (prípadne NoSQL) databázy? Neexistuje jasný víťaz. Vždy záleží na potrebách aplikácie. Z tohoto dôvodu je potrebné analyzovať dáta, ktoré budú ukladané a čo všetko bude výsledná aplikácia vyžadovať. Bežne sa v riešeniach veľkých aplikácií využívajú tieto databázy kombinovane.

[6] https://en.wikipedia.org/wiki/PostgreSQL#Notable_users môžeme si všimnúť komerčnú genealogickú a sociálnu platformu Geni.com, ktorá PostgreSQL používa ako ich hlavnú genealogickú databázu

[7] https://www.mongodb.com/who-uses-mongodb

Kapitola 3

Analýza a návrh

V prvej časti tejto kapitoly budú vysvetlené základné pojmy týkajúce sa genealógie a matrík a popísaná momentálna situácia digitalizácie a spôsobu uchovávania matričných údajov v Českej republike.

Druhá časť sa bude venovať analýze dát a požiadavkov aplikácie, ktorá ich bude využívať. Na základe toho bude navrhnutá schéma relačnej databázy a vytvorené kolekcie dokumentov, ktoré budú slúžiť pre testovanie dokumentovej databázy.

3.1 Čo je genealógia?

Genealógia (z Gréckeho genea, „generácia“ a logos, „poznanie“) alebo rodopis je pomocná veda historická skúmajúca vývoj rodov, ich línií a histórie, a vzťahy medzi rodovo príbuznými jedincami. Taktiež prezentuje súvisiace biologické, historické sociologické a právne dôsledky [20]. Genealógovia pre svoj výskum využívajú širokú škálu záznamov. Sú nimi napríklad:

- Biografie
- Census, resp. sčítania ľudu
- Majetkové listiny
- **Matriky**
- Súdne záznamy

3.2 Matriky a digitalizácia v ČR

V tejto sekcii budú vysvetlené základné pojmy týkajúce sa matriky a popísaný momentálny stav digitalizácie údajov v Českej republike. Definície odborných pojmov *matrika* a *matričný úrad* budú preberané zo zákona o matrikách [10] a oficiálnej stránky Českej genealogickej a heraldickej společnosti v Prahe [17].

Matrika je štátna evidencia narodení, uzatvorení manželstiev, vzniku registrovaných partnerstiev a úmrtí fyzických osôb na území Českej republiky, prípadne v cudzine, ak sa jedná o štátnych občanov Českej republiky alebo uzatvorení manželstiev, ku ktorým došlo v cudzine, ak bol život snúbenca priamo ohrozený a nejde o občanov Českej republiky.

Matriky pôvodne vznikali ako základná forma evidencie obyvateľov pre potreby cirkvi. Od počiatku zaznamenávali podstatné skutočnosti a udalosti týkajúce sa človeka. Postupom času vďaka zásahom štátu získavali súčasnú podobu kníh, ktoré evidujú záznamy o všetkých okolnostiach dôležitých pre poznanie osobného stavu občanov nielen v cirkvi, ale aj štáte ako takom.

Matričným úradom môže byť obecný úrad, úrad mestskej časti či mestského obvodu v územne členených štatutárnych mestách alebo obvodný úrad vo vojenských obvodoch, vždy ale len pre svoj vymedzený územný obvod.

Digitalizácia matričných údajov je proces, ktorého účelom je previesť fyzické dáta z archívnych kníh do digitálnej podoby. V Českej republike sa digitalizácia matrík začala v roku 2007. Po desiatich rokoch, v roku 2017 bolo podľa genealogičky Heleny Voldánovej dostupných približne 95% matričných údajov vo forme digitálnych fotografií [16]. Od začiatku digitalizácie sú všetky digitalizované historické archívy v Českej republike spravované ôsmymi inštitúciami [17]. Napríklad pre Jihomoravský kraj, Zlínsky kraj, časť Vysočiny a Olomouckého kraja sú digitalizované archívy ukladané a spravované Moravským zemským archívom v Brne (MZA). K roku 2019 MZA uchováva už viac ako 11000 *signatúr*. Signatúry predstavujú unikátne číslo, ktorým môžeme odlíšiť jednotlivé digitalizované knižné zväzky.

Jedna matrika môže pokrývať niekoľko obcí a zároveň uchovávať niekoľko signatúr k jednej, či viacerým obciam. Je to z historického dôvodu, keď bolo bežné, že niekoľko dedín spadalo pod jeden farský úrad, prípadne „cirkevnú oblasť“. Väčšina matričných úradov za desaťročia až storočia ich pôsobenia vyprodukujú niekoľko signatúr.

Ďalšou skutočnosťou je, že niektoré matriky vedú záznamy o svadbách, narodeniach a úmrtiach v spoločne v jednom zväzku, iné oddelene. Existujú aj zvláštne matričné úrady, ako napríklad Úrad mestskej časti Brno-střed, ktorý vedie zvláštnu matriku pre narodenia, uzavretia manželstiev či vznik partnerstva alebo úmrtí českých občanov, ku ktorým došlo mimo územia Českej republiky [22].

Tabuľka 3.1: Zjednodušená ukážka vyhľadávacieho systému Acta Publica

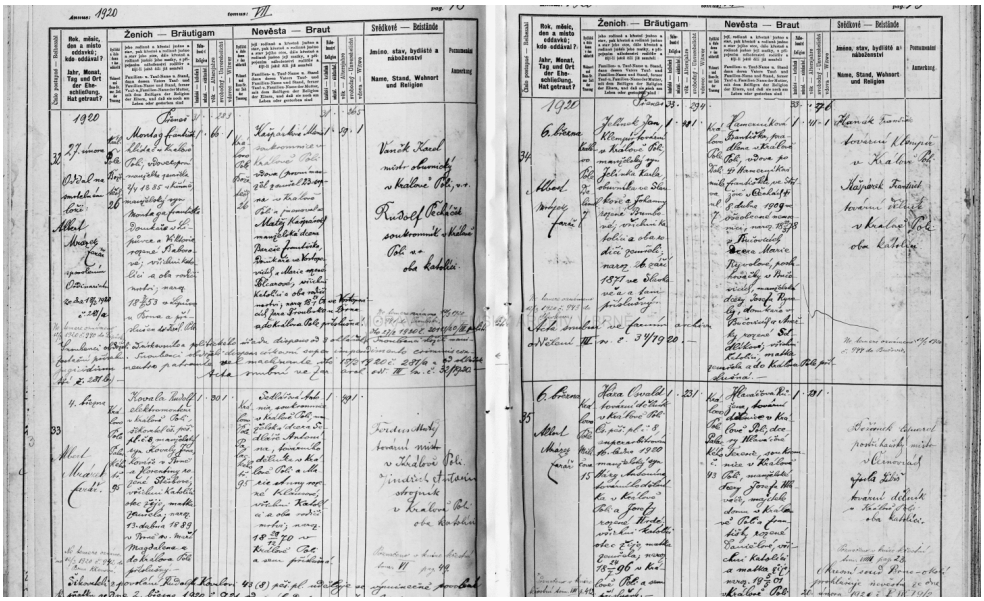
Signatúra	Okres	Obce	Pôvodca	Narodenia	Svadby	Úmrtia
1	Blansko	1	Adamov	1857 - 1884	-	-
...						
11	Blansko	5	Bedřichov	1785 - 1859	-	-
12	Blansko	5	Bedřichov	1860 - 1889	-	-
15	Blansko	5	Bedřichov	-	1785 - 1859	-
...						
2648	Břeclav	6	Hustopeče	1731 - 1757	1731 - 1732	1731 - 1750
...						
13260	Vyškov	34	Vyškov	1624 - 1670	1602 - 1670	1660 - 1671
...						

V tabuľke 3.1 je zjednodušene popísaná štruktúra vyhľadávacieho systému pre MZA na stránke projektu Acta Publica [14]. Každá signatúra má okrem vyššie uvedených záznamov uvedený počet strán a uchováva všetky naskenované fotografie danej matričnej knihy. Práve tieto záznamy slúžia ako primárny zdroj pre analýzu dát o svadbách a úmrtiach, ktorými sa zaoberá táto bakalárska práca.

Na stránkach je momentálne možné vyhľadávať naskenované signatúry bez potreby návštevy bádateľne. Jednou z možností je zadať do vyhľadávača názov obce, v pre ktorú chce

užívateľ záznamy prezerať. Ďalšou možnosťou je zadať konkrétne číslo signatúry, a prípadne konkrétnu stranu. Vyhľadávanie prázdneho reťazca zase poslúži pre prezeranie všetkých zverejnených záznamov.

Ako už bolo vyššie spomenuté, každá digitalizovaná signatúra je niekoľko stranový zväzok, ktorý je voľne dostupný na internete. Príklad naskenovanej matričnej knihy môžeme vidieť na obrázku 3.1. Analýza konkrétnych matričných záznamov o svadbách a úmrtiach a spôsobe ich ukladania bude popísaná v nasledujúcej časti.



Obr. 3.1: **Ukážka matričných záznamov o svadbách v digitalizovanej podobe - zatiaľ len ako fotografia.** Na tomto obrázku môžeme vidieť digitalizovaný scan matriky *Brno - Královo Pole, Nejsvětější Trojice* so signatúrou 17578, ktorá uchováva oddacie záznamy z rokov 1920 - 1928. Obrázok je prevzatý zo záznamov platformy Acta Publica pre Moravský zemský archív v Brne.

3.3 Vstupné dáta

Úlohou tejto práce je analyzovať záznamy z matrík, ktoré uchovávajú informácie o svadbách a úmrtiach a následne navrhnúť štruktúru dát pre ich ukladanie.

Ako primárny zdroj informácií o týchto záznamoch boli použité digitalizované matriky projektu Acta Publica pre Moravský zemský archív v Brne [14]. Väčšina naskenovaných záznamov je ťažko čitateľná. Pre zjednodušenie analýzy záznamov bolo pre túto prácu poskytnutých niekoľko už prepísaných záznamov signatúr svadieb[1] a úmrtí[2]. Súbory s prepísanými záznamami boli poskytnuté spracovateľmi MZA vo forme `xls` tabuľkových dokumentov.

Na rozdiel od matričných záznamov rodokmeňov, či obcí sú svadby a úmrtia záznamy udalostí. Znamená to, že pre každý záznam bude v návrhu potrebné uvažovať okrem entity samotného záznamu a osoby (resp. dvoch osôb pre manželský zväzok) s ním prepojenej aj ďalšie entity, ktoré sú pre danú udalosť zaznamenané ako jej súčasť, čo v konečnom dôsledku znamená viac vzťahov, nad ktorými sa bude potrebné pri modelovaní schémy relačnej

[1] http://actapublica.eu/matriky/brno/prohlizec/6861/?strana=89
[2] http://actapublica.eu/matriky/brno/prohlizec/3819/?strana=10

databázy (prípadne kolekcií dokumentov) zamyslieť. Všetky entity a dôležité atribúty sú popísané v podsekciách 3.3.1 a 3.3.2.

Keďže sa jedná o dáta staré až niekoľko storočí, je potrebné brať do úvahy, že nie každý záznam má vyplnené všetky polia. Napríklad sa môže v archívoch vyskytnúť záznam o pohrebe, kde je uvedený dátum úmrtia, no dátum pohrebu chýba. Iným „extrémom“ sú napríklad záznamy o svadbe bez ženícha. Rátať je potrebné naozaj s každou variantou a preto všetky atribúty okrem primárnych kľúčov (identifikátor záznamu) budú musieť byť nepovinné.

3.3.1 Oddací záznam

V prípade, že si predstavíme, aké základné údaje môže záznam o svadbe uchovávať, pravdepodobne každému napadnú údaje o ženíchovi, neveste, dátume a mieste svadby. Všetky tieto informácie, samozrejme, záznamy o svadbách v sebe nesú. Okrem nich sa v matrikách uchovávajú aj údaje o oddávajúcom, ktorým môže byť farár, či administratívny pracovník matriky, ktorý má právomoc občanov zosobášiť. Je nutné zapisovať aj údaje o svedkoch, pretože svadba bez nich či už v minulosti, ale aj v súčasnosti nie je právoplatná [18]. V analyzovaných matrikách sa vyskytujú údaje prevažne až o štyroch svedkoch. Ďalej svadobný záznam dopĺňajú informácie o rodičoch ženícha a nevesty. Ďalšími údajmi, ktoré by mohli vo výslednom modeli vytvoriť entitnú množinu sú rečník, stará svadby a družba. Po konzultácii s vedúcim práce boli tieto entity v návrhoch modelov vynechané, nakoľko informácie o nich neboli v žiadnom z poskytnutých dokumentov vyplnené a ich absenciu pri testovaní dotazovania bude kompenzovať entita Oddávajúci (v ER diagramoch `Officiant`), ktorá v sebe uchováva podobné atribúty v podobnom rozsahu a hlavne má rovnakú kardinalitu vzťahu ku svadbe (1-ku-N). Rozdelenie entitných množín súvisiacich so záznamami o svadbách a ich základné atribúty bez normalizácie sú popísané v tabuľke 3.2.

Tabuľka 3.2: Entity spojené so svadobným záznamom a k nim príslušné atribúty

Entitná množina	**Atribúty**
Svadobný záznam	záznam hotový, poradie záznamu, poradie scanu, rozloženie scanu, dátum svadby, ohlášky (1-3), vek ženícha a nevesty, vzťah
Osoba (ženích, nevesta, rodičia, svedok)	meno, priezvisko, obec, ulica, č.p., dátum narodenia, vierovyznanie, povolanie
Oddávajúci	meno, priezvisko, titul
Spracovateľ	meno
Register	archív, fond, signatúra

Konkrétne vzťahy medzi entitnými množinami vrátane riešení M-ku-N vzťahov sú zobrazené v relačných diagramoch návrhu relačnej databázy v sekcii 3.4.

Atribúty, ktoré nie sú na prvý pohľad jasné:

- **ohlášky** - svadobné ohlášky sú oficiálnou informáciou o zamýšľanom manželstkom zväzku pre miestne spoločenstvo veriacich. V databáze budú ukladané dátumy ohlášok. Veľká časť svadobných záznamov nemá dátum ohlášok vyplnený, no niektoré uchovávajú až tri dátumy ohlášok.

- **vzťah** - v minulosti bolo bežné, že sa sobášili jedinci v rámci rodín. Či už to boli bratranci a sesternice z prvého alebo druhého kolena, strýkovia a netere, dokonca v niektorých prípadoch nevlastní súrodenci. Informácia o tzv. *inbreedingu* [9] je užitočná napríklad pre zisťovanie výskytu chorých potomkov, resp. genetických porúch.

3.3.2 Úmrtný záznam

Úmrtný záznam sa v celkovom návrhu od svadobného záznamu líši počtom previazaných entitných množín a niekoľkými atribútmi vlastnej entity záznamu. Oproti svadobnému záznamu, ktorý bol prepojený s dvomi osobami (ženích a nevesta) je úmrtný záznam potrebné prepojiť s jednou osobou. V svadobnom zázname sú ženích a nevesta prepojení s rodičmi a svedkami. Pre osobu v úmrtnom zázname je tiež potrebné prepojenie s rodičmi, no namiesto svedkov je potrebné osobu prepojiť s jej deťmi. Pre úmrtné záznamy nebolo nutné riešiť neúplnosť získaných materiálov, tým pádom návrh databázového modelu z hľadiska úmrtných záznamov obsahuje všetky uchovávané údaje z analyzovaných matrík. Rozdelenie entitných množín súvisiacich so záznamami o svadbách a ich základné atribúty bez normalizácie sú popísané v tabuľke 3.3.

Tabuľka 3.3: Entity spojené so úmrtným záznamom a k nim príslušné atribúty

Entitná množina	**Atribúty**
Úmrtný záznam	záznam hotový, poradie záznamu a scanu, poznámky, rozloženie, dátum úmrtia, zaopatrenia a pohrebu, miesto úmrtia a pohrebu, vek zosnulého, príčina smrti, meno,
Osoba (zosnulý, rodičia, deti)	priezvisko, adresa, prehliadka (kým), poznámky dátum narodenia, vierovyznanie, povolanie
Zaopatrovateľ	meno, priezvisko, titul
Pochovávajúci	meno, priezvisko, titul/povolanie
Spracovateľ	meno
Register	archív, fond, signatúra

Atribúty, ktoré nie sú na prvý pohľad jasné:

- **prehliadka (kým)** - v analyzovaných úmrtných záznamoch sa vždy vyskytujú kolónky *„ohledání A/N“* a *„ohledání kým“*. V prvej z nich je označené, či prebehla prehliadka zosnulého (áno / nie). Druhá kolónka zaznamenáva meno - najčastejšie lekára.

- **titul/povolanie** - v niektorých matričných záznamoch sa pochovávajúci od zaopatrovateľa líšili, v iných to bola tá istá osoba. Z tohoto dôvodu som sa rozhodol pochovávajúceho modelovať ako vlastnú entitnú množinu, ktorá nemusela byť vždy duchovný správca, či farár - tzn. mala nejaký titul. Pochovávajúci mohol byť aj napríklad dedinský hrobár, čo už titul nie je. Stále je však tento atribút modelovaný ako atomický, nakoľko vo všetkých analyzovaných záznamoch bola táto položka vyplnená maximálne jednou hodnotou.

Požiadavky aplikácie ovplyvňujúce návrh (schémy a kolekcií)

Je dôležité spomenúť, že oba typy záznamov nesú okrem informácií, ktoré môžeme vyčítať z naskenovanej matričnej knihy, aj ďalšie informácie dôležité pre správu celého systému, ktorý zastrešuje záznamy analyzované v tejto bakalárskej práci. Sú to údaje o spracovateľovi daného záznamu (entita `User`). Atribúty archív, fond a signatúra, ktoré spája entita `Register`. Slúžia k identifikácii miesta v matričnom systéme, na ktorom môžeme konkrétny záznam nájsť. Poradie záznamu a scanu, informácia o tom, či je záznam hotový (prepísaný) a rozloženie scanu sú ďalšie informácie, ktoré je v systéme potrebné uchovávať pre potreby zapisovateľov.

3.4 Relačný model

Pre popis návrhu relačnej databázy bol zvolený ERD (*Entity Relationship Diagram*) model.

3.4.1 Normalizácia schémy relačnej databázy

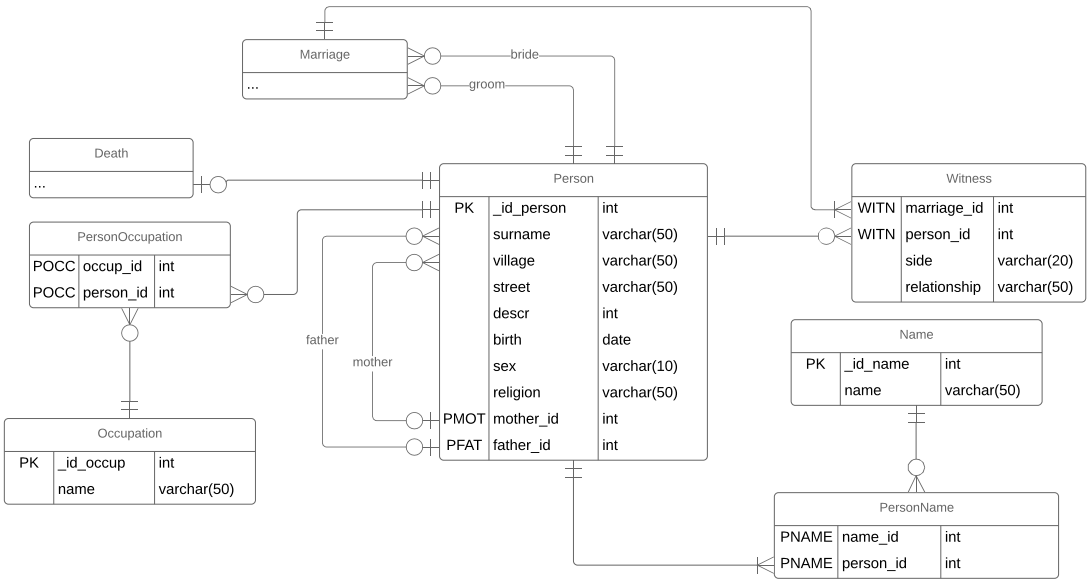
Zo základov relačnej algebry a znalosti relačného modelu dát, ktorý využíva normálne formy pre vytvorenie efektívnej schémy relačnej databázy vieme, že hodnoty v každom poli tabuľky musia byť atomické. Aby každá tabuľka pozostávala výlučne z atomických hodnôt a aby sa zároveň nestratili žiadne dôležité informácie, je nutné vysporiadať sa s niekoľkými zloženými atribútmi, či M-ku-N vzťahmi medzi entitnými množinami:

- **deti** - v oboch záznamoch je nutné ku osobám, ktorých sa záznam týka, previazať entitu otca a matky. V databázovej schéme to znamená jednoduché prepojenie tabuľky samej so sebou pomocou pomenovaného vzťahu. V prípade úmrtných záznamov je osoba previazaná aj s deťmi, čo schému neovplyvní (viď. 3.2), no je potrebné to zohľadniť pri generovaní testovacej sady, ktoré bude popísané v kapitole 4.3.
- **svedkovia** - k jednej svadbe prislúcha niekoľko svedkov a rovnako osoba, ktorá je svedok, môže byť svedkom na viacerých svadbách. U svedkov sa v záznamoch tiež rozlišuje, či je vo vzťahu k neveste alebo ženíchovi a aký je tento vzťah. Tento M-ku-N vzťah je riešený pomocou vzťahovej tabuľky, ktorá v sebe zahŕňa ID svedka, ID svadby, stranu (ženích / nevesta) a vzťah, ktorý má svedok k oddávanému.
- **mená** - v poskytnutých záznamoch sa vyskytujú osoby s krstnými menami, ktoré sú zapisované rôzne. Buď ide o zápis v rôznych jazykoch (Jan, Janusz), či zápis v rôznych skloňovacích pádoch (Janem, Janovi). Pre prehľadnosť a jednoduchosť vyhľadávania je žiadúce, aby boli tieto mená ukladané aj v normalizovanej podobe. Niektoré osoby majú taktiež dve krstné mená, resp. stredné meno (napr. oddávajúci Johannes Georgius Skalka). Kvôli tomu je meno vhodné modelovať ako vlastnú entitnú množinu a vytvoriť preň novú tabuľku. Tá bude s osobami prepojená cez tabuľku vzťahovú, ktorá bude v sebe niesť odkaz na ID mena a ID osoby, čo vyrieši ďalší M-ku-N problém osôb s viacerými menami a mien, ktoré môžu odkazovať na viac osôb.
- **povolania** - niektoré osoby v matričných záznamoch majú zapísané viac ako jedno povolanie, čoho riešením je obdoba problému viacerých mien - vytvorením vlastnej tabuľky povolaní.

Riešenia vyššie spomenutých problémov môžeme vidieť na obrázkoch nižšie. Kvôli príliš veľkému diagramu a lepšej prehľadnosti jeho popisu bol návrh rozdelený do „pohľadov“ troch základných entitných množín navrhovanej databázy. Spojením týchto pohľadov vzniká návrh celého databázového systému svadieb a úmrtí viditeľného v prílohe A.

3.4.2 Model z pohľadu osoby

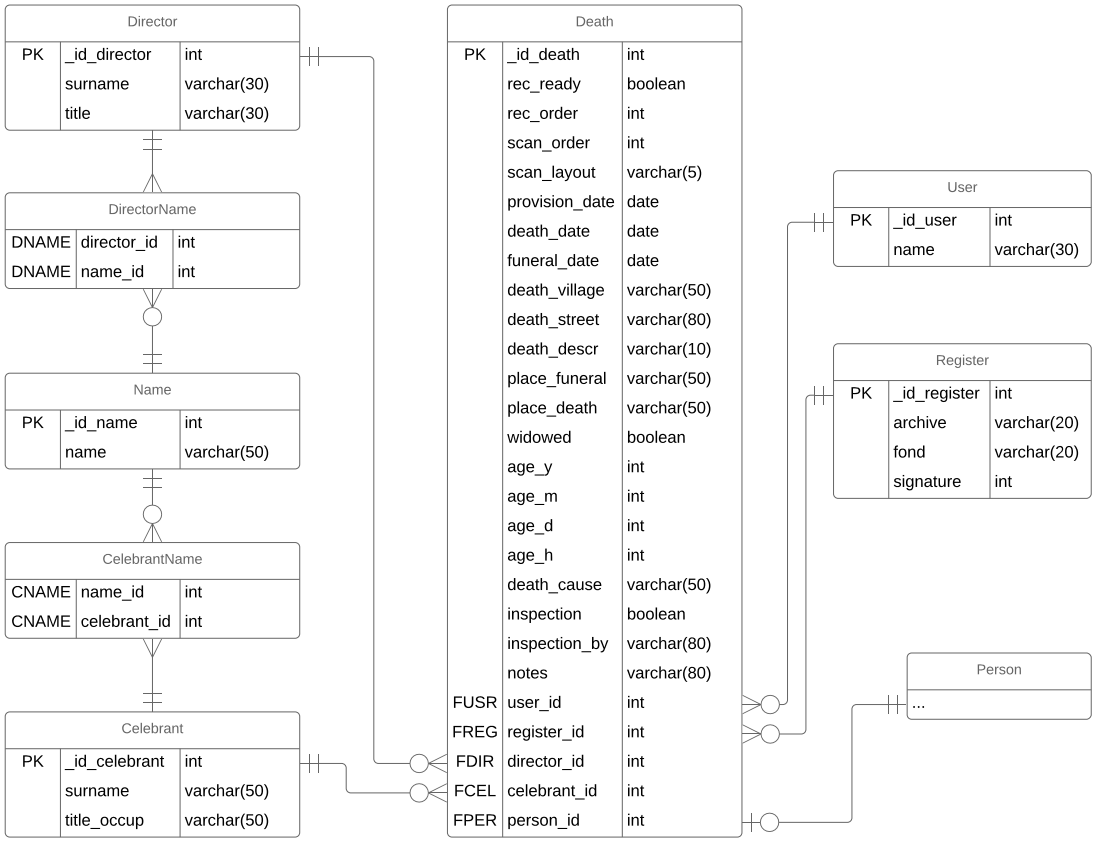
Medzi tabuľkou `Person` a tabuľkami `Name` (Meno) a `Occupation` (Povolanie) sú riešené pomocou väzobných tabuliek `PersonName` a `PersonOccupation`, ktoré obsahujú len primárne kľúče prepájaných entitných množín. Viacero mien a povolaní môže mať každá osoba v entitnej množine `Person`. Oddávajúceho povolanie sa v databázovom systéme nemodeluje, nakoľko je u neho záznam vždy maximálne jedného titulu. Zaujímavejšie je riešenie vzťahu medzi osobou, ktorá môže byť svedkom na 0 až N svadbách a svadbami, ktoré majú až štyroch svedkov. Daný problém rieši väzobná tabuľka `Witness` (Svedok), ktorá v sebe okrem primárnych kľúčov spájaných entitných množín drží aj informácie o tom, ku ktorej osobe z oddávaných má prípadne vzťah (ak medzi svedkom a jednotlivcom vzťah je). Tieto informácie sú uložené v atribútoch `side` (strana) a `relationship` (vzťah). Zložený atribút pre adresu je rozdelený do atomických polí `village` pre obec, `street` pre ulicu a `descr` pre číslo popisné. Ďalej je pre každú osobu uchovávaný dátum narodenia `birth`, pohlavie `sex` a vierovyznanie `religion`. Každá osoba, ktorá bude prepojená so záznamom `Death` alebo `Marriage` má v atribútoch cudzie kľúče rodičov `mother_id` a `father_id`, ktoré na obrázku prepájajú entitu osoby s rovnakou entitou pomenovaným vzťahom.



Obr. 3.2: **Návrh databázy z pohľadu entity `Person` (Osoba).** Na obrázku sú zobrazené všetky vzťahy osoby k ostatným entitným množinám v návrhu databázového systému. Môžeme si všimnúť, že sú tu riešené až tri M-ku-N vzťahy medzi osobou a ďalšími entitnými množinami.

3.4.3 Model z pohľadu úmrtného záznamu

Entitná množina pre úmrtný záznam - tabuľka `Death` - je prepojená so „systémovými“ tabuľkami `User` (Zapisovateľ) nesúca meno a ID zapisovateľa, a `Register` (Matričná kniha), ktorý je určený archívom, fondom a signatúrou. Tabuľky `Directorp` re entitu zaopatrovateľa pohrebu a `Celebrant` pre entitu pochovávajúceho sú ďalej prepojené s tabuľkou `Name` obdobne ako entita `Person` na obrázku vyššie - pomocou väzobných tabuliek. Záznam o úmrtí nesie systémové údaje pre zapisovateľov (poradie záznamu a scanu, rozloženie scanu, atď.) a matričné údaje o mieste a dátume smrti a pohrebu, resp. zaopatrenia (`provision`). Vek osoby by mohol byť len odvodený atribút (vypočítaný z dátumu úmrtia a dátumu narodenia), no pre jednoduchosť a rýchlosť vyhľadávania je lepšie tieto atribúty ukladať priamo v tabuľke.

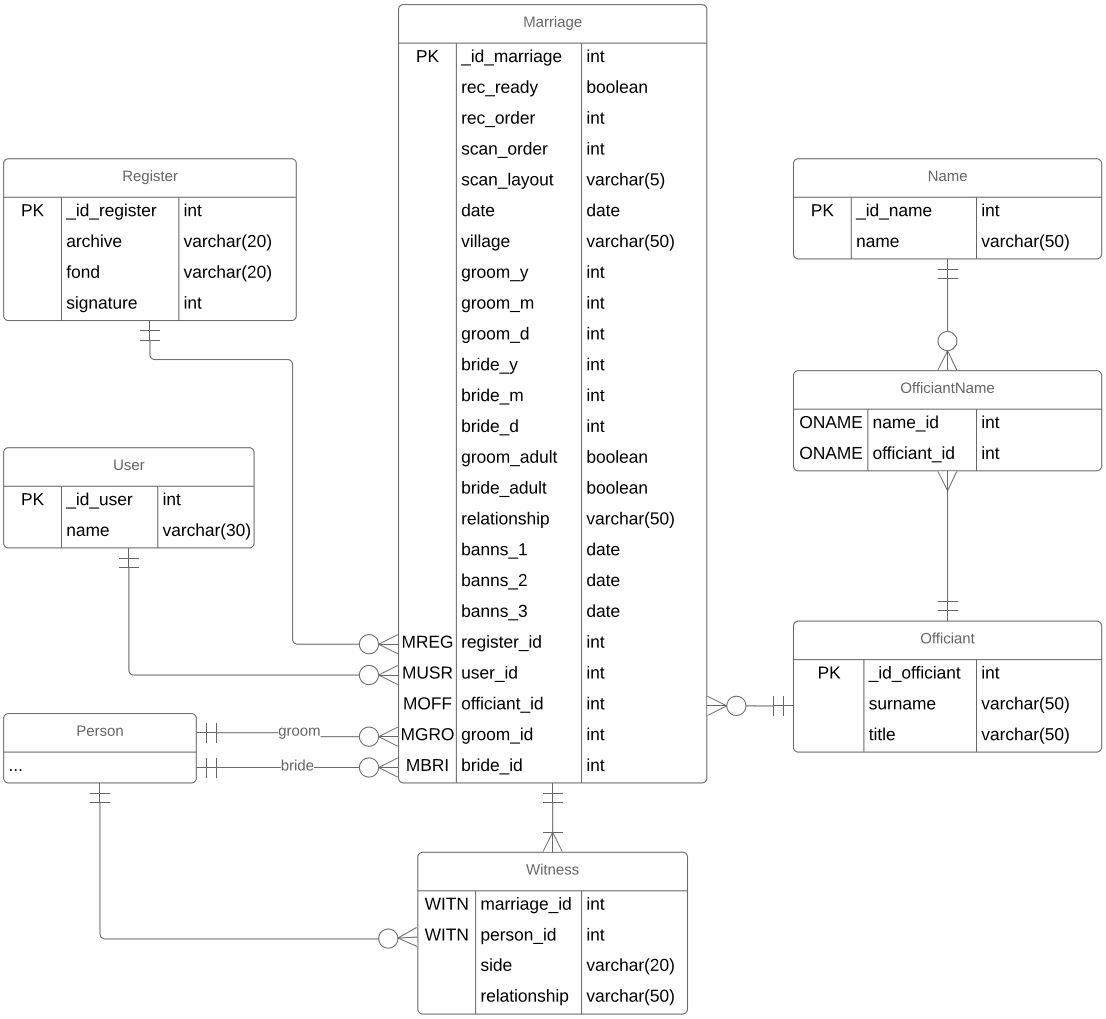


Obr. 3.3: **Návrh databázy z pohľadu entity `Death` (Úmrtný záznam).** Na obrázku sú zobrazené všetky vzťahy úmrtného záznamu k ostatným entitným množinám v návrhu.

3.4.4 Model z pohľadu svadobného záznamu

Obdobne, ako pri úmrtnom zázname je táto entitná množina pre svadobný záznam - tabuľka `Marriage` - je prepojená s tabuľkami `User` (Zapisovateľ) a `Register` pre potreby zapisovateľov a možnosť zaradenia matričného záznamu do správneho archívu a fondu. Tabuľka `Officiant` (Oddávajúci) je kvôli možnosti viacerých mien a možnej normalizovanej

podobe mena prepojená s tabuľkou `Name` pomocou väzobnej tabuľky `OfficiantName`. S entitou `Person` má svadobný záznam tri prepojenia pre svadobný pár a svedkov, ktoré boli popísané v prvom obrázku 3.2. Rovnako ako v prípade úmrtného záznamu som sa kvôli efektivite vyhľadávania a vytvárania dopytovacích príkazov rozhodol vek oddávaných ponechať ako atomické atribúty v tejto entitnej množine, a tiež rovnako, ako je zaznamenávaný v matričných knihách.



Obr. 3.4: **Návrh databázy z pohľadu entity `Marriage` (Svadobný záznam).**

3.4.5 Finálny model relačnej databázy

Finálny model, ktorý je spojením obrázkov 3.2, 3.3 a 3.4 pozostáva zo 16 tabuliek, z ktorých 10 popisuje entitné množiny a zvyšných 6 pomáhajú modelovať M-ku-N vzťahy mien, povolaní a svedkov na svadbách. Vo všetkých tabuľkách si môžeme všimnúť okrem primárnych kľúčov aj pridelenie cudzích kľúčov, ktoré zabezpečujú vzťahy medzi tabuľkami.

3.5 Dokumentový model

Hneď v úvode je vhodné pripomenúť, čo je na dokumentovo orientovaných databázach odlišné oproti bežným relačným databázam. Dokumentové databázy, aj MongoDB sú tzv. **schema-less**, čo znamená že s databázou môžeme začať pracovať bez toho, aby sme mali definovanú jednotnú schému. Každý záznam (dokument) môže mať inú štruktúru. Nový zápis do matriky svadieb, či úmrtí je uložený ako nová položka v prislúchajúcej kolekcii a každá takáto položka môže mať viac či menej atribútov ako predošlá.

3.5.1 Kolekcie dokumentov

Logické oddelenie dokumentov svadieb a úmrtí zabezpečia samostatné kolekcie. Kolekcia `marriages` bude uchovávať všetky dokumenty, resp. záznamy o svadbách a v kolekcii `deaths` nájdeme všetky potrebné dokumenty úmrtí. Pre operácie nad týmito kolekciami budeme potrebovať v JavaScriptovom shelli zadať príkaz `db.deaths.<method>()` pre operácie nad kolekciou úmrtných záznamov a obdobne `db.marriages.<method>()` pre rôzne operácie nad kolekciou svadobných záznamov.

3.5.2 Indexy pre kolekcie

Aby sme však docielili rýchlosti vyhľadávania aj pri, aké MongoDB sľubuje, je vhodné pre každú kolekciu vytvoriť indexy. Indexovať je v dokumentovej databáze potrebné každý atribút, pomocou ktorého hodnoty chceme záznamy filtrovať - pre potreby matričných záznamov je v konečnom riešení indexovaná zhruba polovica atribútov vrátane atribútov vnorených záznamov, tzv. *embedded documents*. Bez indexov je taktiež obtiažnejšie dokumenty upravovať, nakoľko je potrebné nájsť všetky výskyty položky, ktorú chceme upraviť.

Napríklad zistíme, že osoba, ktorá je ženíchom na svadbe a svedkom na inej svadbe má zle zapísaný atribút obce. V relačnej databáze stačí upraviť danú tabuľku osoby, na ktorú sa svadba, resp. svedok odkazuje. Dokumentová databáza má všetky tieto záznamy vytvorené znova - neexistuje „jediný zdroj pravdy“. Pomocou indexov sa takýto proces podstatne urýchli, pretože nie je nutné (či už pri vyhľadávaní, alebo upravovaní) prejsť celú kolekciu (vykonať tzv. *collection scan*) pre nájdenie záznamu, ktorý vyhovuje zadaným parametrom operácie.

3.5.3 Interná schéma kolekcií

Aby bolo možné vytvoriť požadované indexy, je vhodné mať pre prehľadnosť vytvorenú tzv. internú schému pre každú kolekciu. V prípade databázy pre túto prácu to budú dve kolekcie. Jedna pre svadobné záznamy s názvom `marriages` a druhá pre záznamy úmrtné s názvom `deaths`. Každý dokument v príslušnej kolekcii bude obsahovať všetky potrebné informácie v základných atribútoch záznamu a vnorených atribútoch rôznych osôb, ku ktorým je nutné údaje uchovávať.

Základné atribúty oboch kolekcií sa zhodujú s tabuľkami `Marriage` z Obr. 3.4 pre kolekciu `marriages` a `Death` z Obr. 3.3 pre kolekciu `deaths` bez primárneho kľúča `_id_death`, ktorý je nahradený automaticky indexovaným `_id` a všetkých cudzích kľúčov (aj vnorených dokumentov osôb), ktoré boli potrebné pre relačnú databázu. Atribúty `register`, `user`, `officiant`, `director`, `celebrant` odpovedajú rovnomenným entitám pre relačnú databázu s výnimkou mien, ktoré sú ukladané v poli vnoreného dokumentu.

Atribúty `witnesses`, `groom`, `bride`, `father`, `mother`, `bride_groom` a `kids` su vnorené dokumenty (*embedded documents*) základnou štruktúrou odpovedajúce tabuľke `Person` na Obr. 3.2 návrhu relačnej databázy. Povolania a krstné mená sú ukladané ako polia typu `String`. Položky `witnesses` a `kids` sú polia vnorených objektov, pričom každá hodnota poľa `witnesses` nesie okrem atribútov tabuľky `Person` ešte atribúty `side` a `relationship` uchovávané vzťahovou tabuľkou `Witness` v návrhu relačnej databázy (viď 3.2). Pre jednoduchšiu predstavu sú nižšie okomentované skrátené výpisy schém kolekcií.

Kolekcia svadobných záznamov

Výpis 3.1: Náhľad štruktúry kolekcie `marriages` - svadobné záznamy

```
marriages: {
    _id: ObjectId // identifikátor dokumentu a zároveň primárny index
    /*** základné atribúty svadobného záznamu ***/
    rec_ready: Boolean
    ...
    /*** vnorené dokumenty osob, spracovatela, atď. ***/
    register: Object // ekvivalent tabulky Register
    user: Object // ekvivalent tabulky User
    officiant: Object // ekvivalent tabulky Officiant
    witnesses: Array // vnorených dokumentov svedkov (Person + Witness)
    /*** ekvivalentné entity z tabulky Person, ***/
    /*** ktoré majú v sebe vnorené dokumenty rodičov ***/
    groom: Object
    bride: Object
}
```

Kolekcia úmrtných záznamov

Výpis 3.2: Náhľad štruktúry kolekcie `deaths` - úmrtné záznamy:

```
deaths: {
    _id: ObjectId // identifikátor dokumentu a a zároveň primárny index
    /*** základné atribúty úmrtného záznamu ***/
    rec_ready: Boolean
    ...
    /*** vnorené dokumenty osob, spracovatela, atď. ***/
    register: Object // ekvivalent tabulky Register
    user: Object // ekvivalent tabulky User
    director: Object // ekvivalent tabulky Director
    celebrant: Object // ekvivalent tabulky Celebrant
    /*** ekvivalenty tabulky Person ***/
    person: Object
    father: Object
    mother: Object
    bride_groom: Object
    kids: Array // pole vnorených dokumentov detí
}
```

3.5.4 Vytvorené sekundárne indexy

Pre obe kolekcie bolo vytvorených dokopy 75 indexov. Kolekcii `deaths` pre účely efektívneho vyhľadávania pri veľkom počte dokumentov bolo vytvorených 41 indexov. Kolekcii `marriages` 34 indexov. Ku každej kolekcii je ešte potreba prirátať po jednom primárnom indexe `_id` typu `ObjectId`, ktorý je vytvorený databázou automaticky.

V realite by malo pre požiadavky systému stačiť aj menej indexov, cca 20 pre každú kolekciu. Avšak pre účely testovania v prípade netradičných dotazov som sa rozhodol ich vytvoriť aj pre niektoré pravdepodobne menej často filtrované atribúty. Používanie indexov je vhodné hlavne pri veľkých objemoch dát. Aplikácie, ktoré využívajú MongoDB mnohokrát obsahujú terabajty dát a milióny záznamov (v big data aplikáciách možné až petabajty dát, a miliardy záznamov). K takýmto číslam sa matriky v Českej republike pravdepodobne veľmi skoro nedostanú, čo znamená, že testy na tisícoch, až desiatkach tisícov záznamov s použitím indexov a bez nich by sa nemali rýchlostne veľmi odchyľovať. Práve tieto možnosti budú testované v sekcii 4.4 nasledujúcej kapitoly.

Kapitola 4

Porovnanie MongoDB a PostgreSQL

Kapitola sa bude venovať porovnaniu zvolených databázových systémov pre využitie v genealógii na základe návrhov vytvorených v predošlej kapitole. V prvej časti budú popísané zvolené metriky porovnávania a ich odôvodnenie. Ďalšie podkapitoly priblížia postup od návrhu testovacej sady až po výsledky meraní podľa zvolených metrík.

4.1 Metriky porovnávania

Najdôležitejšou časťou tejto práce je výber správnych metrík pre porovnanie zvolených databázových systémov. Je potrebné zamyslieť sa, čo je potrebné testovať a aká váha bude jednotlivým metrikám pridelená. Aspekty porovnávania budú v nasledujúcom zozname zoradené od najdôležitejších po najmenej dôležité. Zároveň bude zoznam akousi osnovou pre záver tejto práce.

- **Návrh.** Návrh je neoddeliteľnou súčasťou každej databázy. Správny návrh je dôležitý, nie len pre dobrý výkon databázy, ale aj pre neskorší rast aplikácie a možné zmeny jej požiadavkov. Hlavne z druhého dôvodu si myslím, že je to najpodstatnejšia vlastnosť pre porovnanie prístupu relačného a dokumentového modelu.
- **Rýchlosť vyhľadávania.** V akejkoľvek genealogickej aplikácii je najčastejšou operáciou práve vyhľadávanie. Efektivita vyhľadávania úzko súvisí s dobrým návrhom. Samozrejme rôzne databázy môžu ponúknuť rôzny výkon, nie je to inak ani v prípade testovaných adeptov, ktorí sa radia práve medzi tie najrýchlejšie na svete. Rýchlosti rôznych typov dotazov vrátane agregácií budú preto dôkladne porovnané v nasledujúcej časti.
- **Rýchlosť hromadného vkladania.** Schopnosť databázového systému vysporiadať sa s rôznymi počtami vkladaných matričných záznamov bude testovaná z dôvodu možnej migrácie veľkého počtu dát do nového systému.
- **Zložitosť vytvárania príkazov.** Vytváranie CRUD operácií pomocou JavaScriptu, či jazyka SQL je hlavne vecou osobnej preferencie. Je ale vhodné zistiť, do akej miery zoskupenie dokumentov v kolekciách dokumentovej databázy ušetrí námahu programátorovi oproti tzv. „JOIN-ovaniu“ tabuliek relačnej databázy.

- **Veľkosť.** Dnes už veľkosť výslednej databázy nehrá až tak dôležitú rolu, ak sa pohybujeme v desiatkach, či stovkách MB. Je ale vhodné vedieť si predstaviť rozdiel medzi veľkosťou relačnej databázy bez redundancie a dokumentovej s pomerne častou redundanciou dát. Taktiež je dôležité zistiť do akej miery môžu indexy ovplyvniť veľkosť danej databázy.

4.2 Konfigurácia a prostredie pre testovanie

Testovanie databáz prebehlo na notebooku strednej užívateľskej triedy s nasledujúcimi parametrami a verziami databázových systémov:

- **CPU**: Intel Core i7-8550U 1.8GHz (4.00GHz max frekvencia)
- **RAM**: 16GB DDR4 2133Hz
- **PostgreSQL 10.7**
- **MongoDB 4.0.9**

Práca s databázami od inštalovania po testovanie prebiehala vo virtualizačnom nástroji **Docker**, ktorý zaručuje rovnakú kompatibilitu naprieč operačnými systémami a hlavne bezpečne „zaobaľuje“ všetko potrebné v kontajneri oddelenom od užívateľského prostredia a jeho nastavení, ako aj od konfigurácie a súborov operačného systému.

Docker

Docker je open-source projekt, ktorého cieľom je poskytnúť jednotné rozhranie pre izoláciu aplikácií do kontajnerov (angl. *Docker container*) v prostredí Linux a Windows (tzv. „odľahčená virtualizácia“). Kontajner obsahuje iba požadované aplikácie a pre ne špecifické súbory bez virtualizovaného operačného systému, čím je výrazne znížená réžia na rozdiel od klasických virtuálnych strojov.

Od jeho vzniku v roku 2012 sa Docker stal jednou z najrýchlejšie rastúcich technológií v DevOps a webovom vývoji. V prípade tejto práce bude slúžiť ako izolované prostredie pre testovanie zvolených databázových systémov.

Výhody, ktoré viedli k použitiu Dockeru pre túto prácu:

- **Izolácia** - pri testovaní (nielen) nových technológií je vhodné všetky nastavenia a závislosti izolovať od vlastného užívateľského prostredia. Kontajner zaručí, že prípadné modifikácia akýchkoľvek nastavení, či inštalácia závislostí neovplyvní chod užívateľského prostredia. Kontajner je možné kedykoľvek jednoducho odstrániť a prípadne vytvoriť nový.
- **Inštalácia a spustenie** - všetko, čo je potrebné k spusteniu databázy je stiahnutie obrazu (*Docker image*) zo stránok Docker Hub[1], kde stačí stiahnuť oficiálne obrazy pre MongoDB, resp. PostgreSQL.
- **Veľkosť** - je omnoho nižšia ako pri iných virtuálnych strojoch.

[1] https://hub.docker.com/

- **Reprodukovateľnosť** - tak ako Java aplikácia, ktorá beží rovnako na každom zariadení, ktoré je schopné spustiť JVM (Java Virtual Machine), kontajner zaručuje identickosť na akomkoľvek systéme, na ktorom je možné spustiť Docker. Presná konfigurácia kontaineru je uložená v `Dockerfile`. Pri distribúcii kontajneru je tým pádom zaručené, že všetky obrazy vytvorené rovnakou konfiguráciou spomenutého súboru budú fungovať identicky.

Ďalším dôvodom, prečo som sa rozhodol pre Docker je možné využitie kontajnerov a konfigurácie databáz v aplikácii, pre ktorú sú v tejto práci porovnávané. Využitie kontajrnerov môže výrazne pomôcť vyššie spomenutými výhodami (a mnohými ďalšími[2] pri vývoji aplikácie. Podrobný rozbor vhodnosti využitia Dockeru a jeho vhodnosti pre priebežnú integráciu (CI - Continuous Integration) je nad rozsah zadania tejto práce.

4.3 Naplnenie databáz vlastným nástrojom

Dôležitou súčasťou tejto práce je vytvorenie testovacej dátovej sady pre porovnanie rýchlosti a zložitosti rôznych databázových operácií. Dátová sada musí spĺňať požiadavky návrhu databázy podľa ERD v predošlej časti práce. Zároveň je potrebné dbať na to, aby vygenerované dáta pre dokumentovú databázu odpovedali rovnakým dátam pre databázu relačnú. V opačnom prípade by nebolo možné posúdiť relevantnosť výsledkov testovania.

Jednou z možností bolo vyhľadanie voľne dostupného nástroja pre generovanie testovacích dát vo formáte JSON pre dokumentovú databázu, resp. nástroj generujúci výstup INSERT príkazov v jazyku SQL databázu relačnú. Problémom takýchto nástrojov sú obmedzenia vo vytváraní komplexnejších vzťahov medzi generovanými dátami rovnako ako počet možných vytvorených záznamov. Napríklad nástroj *generatedata.com*[3] bez platenej registrácie obmedzuje počet vygenerovaných záznamov na 100, čo je pre účely testovania žalostne málo.

Po neúspešnom pátraní po vhodnom generátore testovacích dát som sa priklonil k druhej možnosti - vytvoriť vlastný skript, ktorý vygeneruje všetky potrebné tabuľky v SQL pre PostgreSQL a zároveň im odpovedajúce dáta vo formáte JSON pre databázový systém MongoDB.

Použitie Node.js

Keďže sú dokumenty v MongoDB ukladané vo formáte JSON (*JavaScript Object Notation*), najvhodnejším adeptom na vytvorenie požadovaného skriptu je jazyk JavaScript. Pomocou JavaSriptu je tiež jednoduché pracovať so súbormi (pre požadovaný výstup hlavne zápis do súborov) a zároveň sa tento jazyk pýši veľkým množstvom knižníc a balíčkov, ktoré pomôžu pri generovaní náhodných údajov pre testovacie dáta.

Pre uľahčenie práce so skriptom bol vytvorený jednoduchý *Node.js* program, ktorý skript spúšťa a stará sa o všetky potrebné balíčky (*packages*), ktoré skript používa. Tieto balíčky sú JavaScriptové knižnce a SDK inštalované z *npm*[4] (Node.js package manager) - správca balíčkov pre JavaScript a prostredie Node.js. Node.js tiež používa vlastné rozhranie príkazového riadku (*npm-cli*[5]) pre jednoduché inštalovanie pomocných knižníc a samotné spustenie skriptu.

[2] https://www.linode.com/docs/applications/containers/when-and-why-to-use-docker/
[3] https://www.generatedata.com/
[4] https://www.npmjs.com/
[5] https://docs.npmjs.com/cli-documentation/

V skripte bol pre prácu s databázou PostgreSQL použitý klient `node-postgres`[6]. Pre vkladanie záznamov do databázy MongoDB bol použitý oficiálny `MongoDB Node.JS Driver`[7] podporujúci funkcie ES6 (ECMAScript 6).

Implementácia

Skript je rozdelený na tri fázy. Samotné generovanie testovacej sady prebieha v dvoch fázach. V prvej fáze sú pre každú tabuľku, resp. entitnú množinu vytvorené jej inštancie. Skript do výstupného súboru generuje INSERT príkazy pre relačnú databázu z každej vytvorenej inštancie (entity) pre vyplnenie danej tabuľky. Všetky entity sú ukladané a zoradené v príslušnom poli, z ktorého sa v druhej fáze na základe vzťahov medzi danými entitami vyberajú prvky pre vytvorenie odpovedajúceho dokumentu pre dokumentovú databázu. Týmto spôsobom je zaručené, že si matričné záznamy v oboch databázach budú svojim počtom, vzťahmi a hodnotami atribútov vzájomne odpovedať. V tretej fáze sú vygenerované množiny dát vkladané do príslušnej databázy.

4.3.1 Prvá fáza: generovanie entít

Pre vygenerovanie, čo najrealistickejších dát bolo potrebné vytvoriť sadu dátových súborov. Súbory boli vytvorené kombináciou voľne dostupných zdrojov na internete a z analyzovaných súborov prepísaných matrík. Súbory boli vytvorené pre:

- **mená a priezviská** osôb z najčastejších výskytov v Českej Republike
- najčasejšie **príčiny smrti** z analyzovaných matričných záznamov
- zoznam najbežnejších **povolaní** v 18. a 19. storočí
- **tituly** oddávajúcich, zaopatrovateľov a pochovávajúcich z analyzovaných matričných záznamov
- zoznam **miest** a **obcí** v ČR z voľne dostupných zdrojov

Po načítaní vyššie vymenovaných zoznamov skript generuje entity v cykloch pre každú tabuľku. Pre 16 tabuliek v návrhu je to 16 cyklov. Počet iterácií každého cyklu je nastavený v rôznych pomeroch. Pomery hodné spomenutia sú počty vygenerovaných osôb ku počtu matričných záznamov svadieb a matričných záznamov úmrtí. Tento pomer je zhruba 8:2:1 (osoby : úmrtné záznamy : svadobné záznamy).

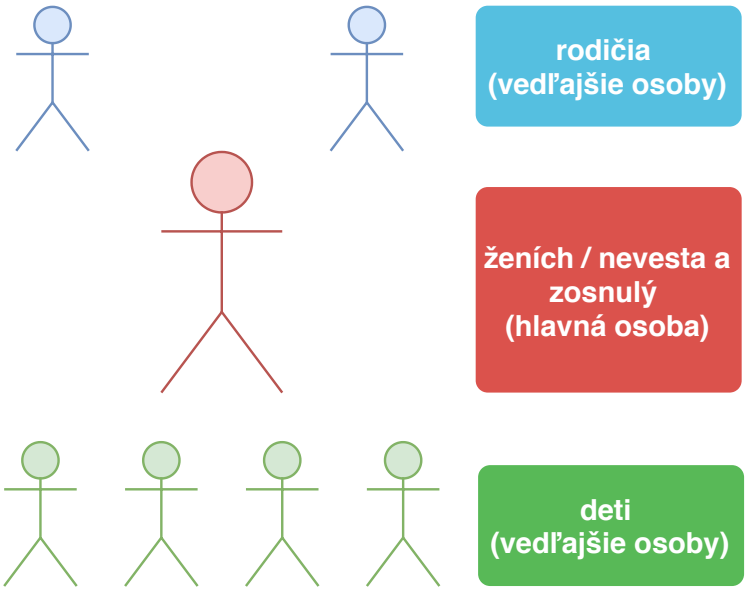
„Jednoduché“ tabuľky

Prvé cykly generujú entity tabuliek `Register`, `User`, `Officiant`, `Celebrant`, `Director`, `Name` a `Occupation`. Pre každú z nich sú vytvorené náhodné atribúty odpovedajúce dátovým súborom popísaným vyššie. Pre entity tabuliek `Officiant`, `Celebrant` a `Director` jedno až dve krstné mená, ktorých princíp generovania je popísaný v 4.3.1 nižšie.

[6] https://node-postgres.com/
[7] https://mongodb.github.io/node-mongodb-native/

Osoby

Ku každej vygenerovanej osobe, ktorá bude figurovať ako zosnulý, ženích, či nevesta sú v jednej iterácii cyklu generovania entít tabuľky `Person` vygenerované entity pre matku, otca a pre 50% z nich je s rovnomernou pravdepodobnosťou vygenerované jedno, až štyri deti. Ďalej budú pre zrozumiteľnosť v texte tieto osoby označované ako **hlavné** (osoby figurujúce v záznamoch ako ženích, nevesta alebo zosnulý) a **vedľajšie** osoby (deti a rodičia hlavných osôb). Cyklus je znázornený na obrázku 4.1 nižšie. Všetky menované osoby (aj vedľajšie) sú generované pre tabuľku `Person`.



Obr. 4.1: **Generovanie osôb.** V jednej iterácii cyklu generovania dát pre tabuľku `Person` je vygenerovaná hlavná osoba, ktorá bude v záznamoch figurovať ako ženích, nevesta alebo zosnulý + dvaja rodičia (pre dáta k svadobnému aj úmrtnému záznamu) + 0 - 4 detí (pre dáta k úmrtnému záznamu).

Mená a povolania

Ako sme si mohli všimnúť na začiatku tejto sekcie vyššie, generovanie tabuliek krstných mien a tabuliek povolaní prebieha v cykle ešte pred generovaním osôb. Až keď sú všetky osoby vygenerované, začne generovanie väzobných tabuliek `PersonName` pre náhodne priradené mená a `PersonOccupation` pre náhodne priradené povolania. Počet iterácií cyklov pre obe tabuľky je určený celkovým počtom všetkých osôb (aj vedľajších). Mená sú prideľované náhodne vďaka pomocnej funkcii `randomIndexFrom()`[8], ktorá slúži ako iterátor nad náhodným rozdelením indexov polí. Počet mien osôb sa líši. Pre každú osobu je vytvorená jedna, až dve tabuľky `PersonName`. S povolaniami je to obdobné, ich počet je pre 80% ľudí 1 až 3 povolania. Zvyšných 20% všetkých ľudí nemá vygenerované povolanie.

[8]Iterátor náhodných indexov randomIndexFrom() je použitý aj v ďalších prípadoch (náhodné vyberanie ženíchov, atď.) Podrobnejší popis je v zdrojovom kóde skriptu.

Svadby a svedkovia

V predposlednom cykle sú generované záznamy svadieb (tabuľka `Marriage`). Nevesty sú všetky ženy z vygenerovaných hlavných osôb a ženíchovia všetci muži. Počet iterácii (čiže počet svadobných záznamov) je stanovený polovicou počtu všetkých hlavných osôb + 20%. Pohlavia generované hlavným osobám sú v pomere 1:1, čiže ženíchov bude rovnaký počet ako neviest. Počet iterácií je navýšený ešte o 20%, aby vznikli aj záznamy o ľuďoch, ktorí majú väčší počet svadobných záznamov ako jeden. Z atribútov svadobného záznamu stojí za zmienku generovanie atribútu `relationship` kvôli skúmaniu inbreedingu popísanému v 3.3.1. Z celkového počtu svadobných záznamov bude 5% svadieb medzi príbuznými s rôznymi hodnotami vzťahu (napr. „strýc-neteř“, „bratranec-sestřenice 2. stupně“ a tri ďalšie). Generovanie ohlášok je náhodné. Niektoré svadby majú len prvé, niektoré aj druhé a niektoré všetky tri dátumy ohlášok.

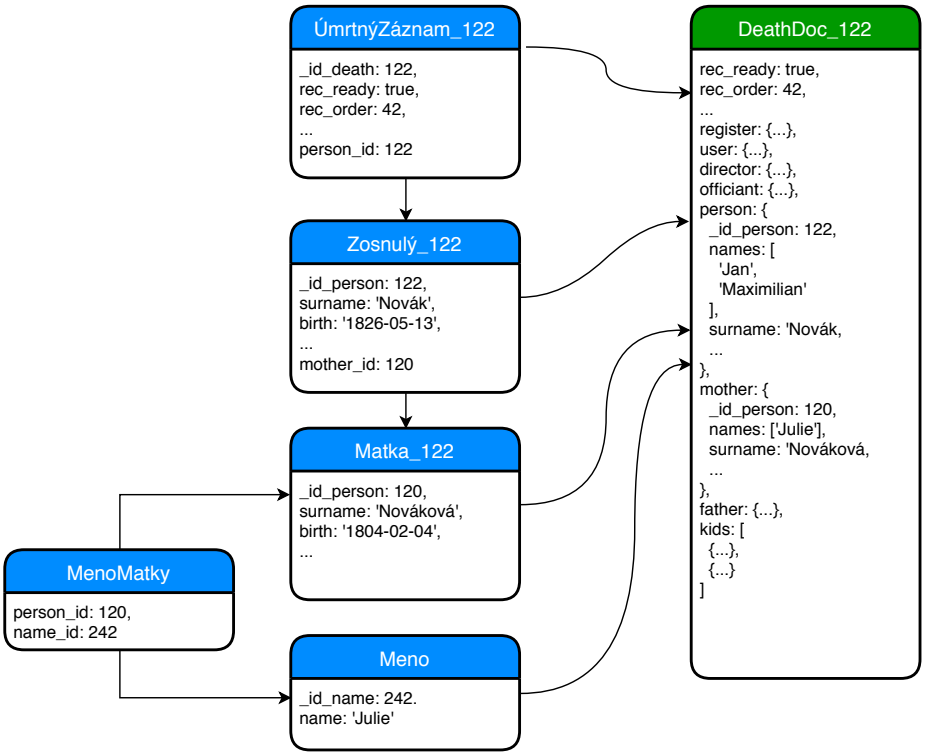
V rámci jednej iterácie generovania svadobného záznamu sú generované dáta pre tabuľky svedkov `Witness`. Svedkovia sa generujú ku každej svadbe, aby bol priradený správny cudzí kľúč svadby do väzobnej tabuľky pre svedka. Pre zjednodušenie už aj tak komplexného generovania sú pre každú svadbu vygenerovaní štyria svedkovia. Dvaja pre nevestinu stranu a dvaja pre ženíchovu. Vzťahy svedkov sú pre jednoduchosť len tri - „sourozenec“, „přítel“ a „jiné“.

Úmrtia

Úmrtné záznamy pre tabuľku `Death` relačnej databázy sa generujú pre všetky vygenerované hlavné osoby. Pre jednoduchšie dotazovanie je podľa návrhu v tabuľke úmrtí ukladaný presný vek úmrtia v atomických atribútoch `age_y`, `age_m`, `age_d`, `age_h`, ktoré sú v iterácii vyrátané z dátumu narodenia osoby a dátumu úmrtia v úmrtnom zázname. Niektoré atribúty v rôznych pravdepodobnostiach vynechané. Sú to napr. `inspection_by`, death_cause, a ďalšie. Úmrtný záznam má vygenerovaný buď dátum úmrtia a dátum pohrebu alebo dátum provízie, nakoľko v analyzovaných dátach sa vyskytovali len prípady týchto možností. Každý desiaty záznam má vyplnené poznámky, každý druhý príčinu úmrtia (najčastejšie „souchotiny“ 20% a „osýpky“ 10%. Ostatné príčiny smrti sú náhodne vyberané z vytvoreného dátového súboru príčin úmrtí.

4.3.2 Druhá fáza: mapovanie entít do kolekcií dokumentov

Druhá fáza generovacieho skriptu spočíva v mapovaní vytvorených svadobných a úmrtných entít do formy dokumentov. Znamená to, že pre každú entitu sú vďaka ukladaným ID (cudzím kľúčom) namapované odpovedajúce tabuľky, s ktorými je entitná množina vo vzťahu. Celá tabuľka, ktorá odpovedá cudziemu kľúču vzťahu je namapovaná do dokumentu úmrtného, resp. svadobného záznamu ako vnorený dokument. Napríklad, ak má cudzí kľúč `mother_id` pre osobu namapovanú do úmrtného záznamu hodnotu 120, je vyhľadaná entita osoby s indexom 120 v poli `persons` všetkých osôb. Pre vyhľadanú entitu sú obdobným spôsobom cez vzťahové tabuľky vyhľadané povolania a krstné mená, ktoré sú namapované do novo vytvoreného dokumentu `deathDocument.mother`. Obdobne sú namapované všetky zvyšné vzťahy z pôvodných relačných tabuliek. Každý namapovaný dokument je pridaný do poľa dokumentov `deathDocuments`, resp. `marriageDocuments` pre dokumenty svadieb, ktoré budú využité v ďalšej fáze - samotnom vkladaní do databázy. Práve popísaný postup druhej fázy môžeme vidieť na nasledujúcom obrázku 4.2.



Obr. 4.2: **Mapovanie relačných tabuliek do dokumentov.**

4.3.3 Trietia fáza: naplnenie databáz

V poslednej fáze sa skript pripojí pomocou klientských rozhraní `node-postgres` a `Node.js MongoDB Driver` ku príslušným databázam.

PostgreSQL

V prípade databázy PostgreSQL je vyčistená stará databáza (ak nejaká už bola vytvorená) pomocou sady nasledujúcich príkazov:

```
DROP SCHEMA public CASCADE;
CREATE SCHEMA public;
GRANT ALL ON SCHEMA public TO postgres; --- názov testovacej DB
GRANT ALL ON SCHEMA public TO public;
```

Nasleduje nové vytvorenie tabuliek, ktorých schéma je v priloženom zdrojovom súbore rovno s indexami pre cudzie kľúče. Po vytvorení schémy a indexov nasleduje nahranie všetkých záznamov **naraz** vo veľkom konkatenovanom reťazci zloženého z INSERT príkazov. Tento postup bol jediný možný pri doposiaľ obmedzenej implementácie (neoficiálneho) API `node-postgres`, ktorý nemôže vytvoriť tzv. „bulk insert“ vložením poľa reťazcov, či objektov, ako to dokáže jeho testovaný (oficiálny) náprotivok `Node.js MongoDB Driver`, v ktorého prípade nebol problém vložiť 100 tisíc komplexných dokumentov. Rýchlosti vkladania záznamov pre konkrétny počet záznamov bude popísaná v nasledujúcej kapitole.

MongoDB

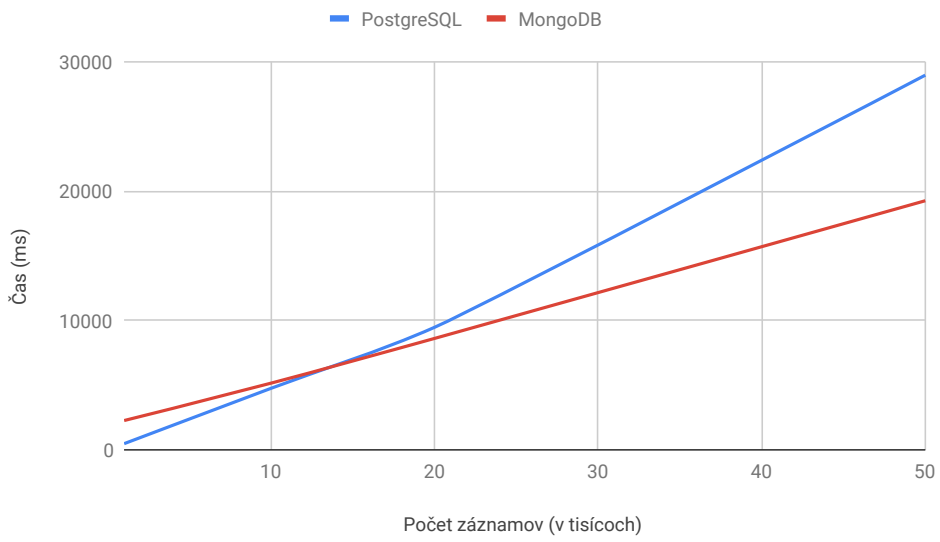
V prípade MongoDB je pre vkladanie jednotlivých kolekcií vytvorené nové pripojenie. V rámci jedného pripojenia sa po vložení dát vytvárajú aj indexy pre danú kolekciu. Vytvorené sekundárne indexy pre atribúty a vnorené dokumenty každej kolekcie sú vypísané v zdrojovom kóde skriptu. Pre vloženie dát v jednom príkaze je použitý spomínaný oficiálny `Node.js MongoDB Driver`. Tak ako pre PostgreSQL, aj pre MongoDB budú rýchlosti vkladania konkrétneho počtu dokumentov popísané v ďalšej kapitole.

4.4 Výsledky testov

V nasledujúcich podkapitolách budú uvedené všetky dosiahnuté výsledky porovnania na základe rôznych počtov testovacích matričných záznamov.

4.4.1 Hromadné vkladanie

Časy vkladania záznamov do databáz boli merané pre všetky veľkosti testovacích sád. V nasledujúcom grafe je vidieť rozdiely medzi hromadným vkladaním dát pre matričné záznamy do databáz PostgreSQL a MongoDB.



Obr. 4.3: **Rýchlosť hromadného vkladania matričných dát.**

Je vidieť, že pri menšom počte matričných záznamov (1 až 10 tisíc) databáza MongoDB mierne zaostáva za databázou PostgreSQL. Od tohto bodu sa to ale mení vďaka tomu, že čas vkladania dát do MongoDB je lineárny, zatiaľ čo každou väčšou dátovou sadou vkladanie matričných záznamov do relačného PostgreSQL začína divergovať. Je to primárne z nasledujúceho dôvodu.

Do databázy MongoDB bol vkladaný počet dokumentov odpovedajúci vkladanému počtu záznamov. Za to na druhej strane, u relačnej databázy je to trochu inak. Keďže navrhnutý model obsahuje až 16 rôznych tabuliek (vrátane vzťahových) je nutné uvedomiť si, že jeden záznam v relačnej databáze nebude odpovedať jednému dokumentu v databáze dokumentovej. V konečnom dôsledku pomer `INSERT` príkazov (záznamov v relačnej DB) ku

počtu vložených dokumentov bol približne 12:1. Napríklad pre generovanú sadu desiatich tisícov matričných záznamov to bolo presne 10014 dokumentov dokumentovej databázy a 125976 záznamov v relačnej databáze.

4.4.2 Veľkosť

V nasledujúcej tabuľke sú uvedené veľkosti databáz po vložení príslušného počtu záznamov, resp. dokumentov. Hodnoty sú uvádzané pre databázy s indexami a bez nich.

Tabuľka 4.1: Porovnanie veľkosti vytvorených databáz

Počet záznamov	**PostgreSQL s indexami**	**PostgreSQL bez indexov**	**MongoDB s indexami**	**MongoDB bez indexov**
1 000	10MB	8.8MB	3.67MB	2.24
10 000	23MB	17MB	26.4MB	22.8MB
20 000	38MB	25MB	54MB	47MB
50 000	82MB	45MB	122MB	108MB

Z nameraných údajov v tabuľke 4.1 je vidieť, že veľkosť dokumentovej databázy narastá rýchlejšie s vyšším počtom dokumentov. Na druhej strane v relačnej databáze je veľkým faktorom rozdiel veľkosti s indexov a bez nich. Samozrejme vždy záleží na užívateľovi, koľko indexov definuje. Pre PostgreSQL boli vytvorené indexy pre každý primárny kľúč automaticky a všetky cudzie kľúče reprezentujúce vzťahy medzi tabuľkami, ako boli popísané v kapitole 3.4. V dokumentovej databáze sú indexované atribúty, ktoré budú s najväčšou pravdepodobnosťou vyhľadávané, napr. miesta úmrtia, vek nevesty a ženícha, mená atď. Počet indexov a bližie informácie boli uvedené v kapitole 3.5.

4.4.3 Vyhľadávanie

Pre porovnanie rýchlosti vyhľadávania bola vytvorená sada 10 vyhľadávacích dotazov. Testovanie prebehlo na sadách dát o rôznych veľkostiach. Popis každého z nasledujúcich desiatich testov bude obsahovať:

- **zadanie** testu
- **SQL príkaz** pre PostgreSQL
- vyhľadávaciu, prípadne agregačnú **operáciu nad kolekciou** v MongoDB
- **čiarový graf**, ktorý na vodorovnej osi znázorňuje počet záznamov a na zvislej osi odpovedajúci čas vyhľadávania milisekundách

Test č. 1

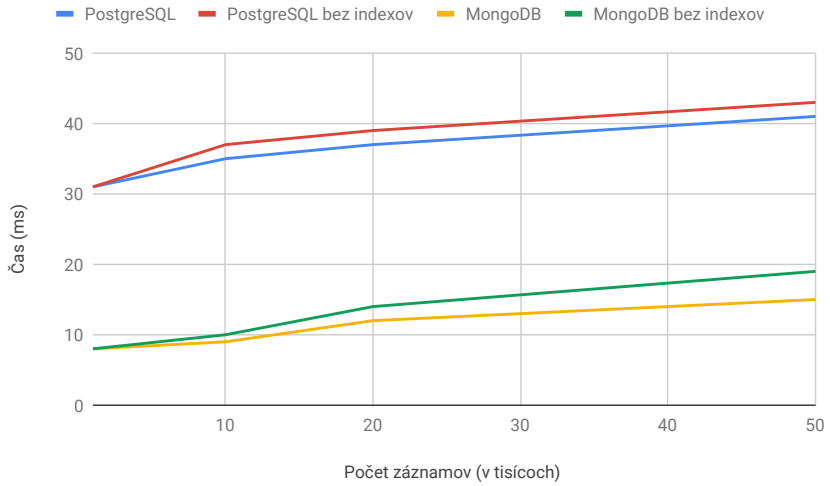
Zadanie: Nájdi všetky svadobné záznamy z Brna, ktoré sa konali v roku 1859 a vypíš ich počet.

Výpis 4.1: Príkaz pre PostgreSQL

```
SELECT COUNT(_id_marriage)
FROM "Marriage"
WHERE "village"='Brno' AND EXTRACT(YEAR FROM "date") = 1859;
```

Výpis 4.2: Príkaz pre MongoDB

```
db.marriages.aggregate([
  {$match: {village: 'Brno', date: {$gte: '1859-01-01',$lte: '1859-12-31'}}},
  {$count: "records"}
])
```



Obr. 4.4: **Výsledky testu č. 1:** S jednoduchou agregáciou kolekcie si MongoDB dokáže poradiť. Bez indexov spomaľuje výraznejšie od 20 tisíc záznamov. PostgreSQL rastie konštantne s použitím indexov, aj bez nich.

Test č. 2

Zadanie: Zisti, ktorí muži mali viac ako jednu svadbu. Vypíš príslušné ID, priezvisko a výstup zoraď od najväčšieho počtu svadieb.

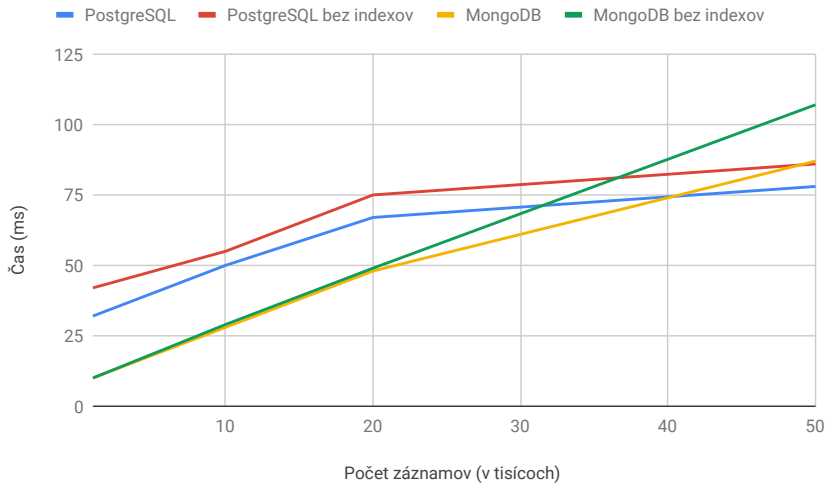
Výpis 4.3: Príkaz pre PostgreSQL

```
SELECT COUNT("Marriage"."_id_marriage") as mc, _id_person, surname
FROM "Person"
JOIN "Marriage" ON "Marriage"."groom_id"="Person"."_id_person"
GROUP BY "Person"."_id_person" HAVING COUNT(*) > 1
ORDER BY mc DESC;
```

Výpis 4.4: Príkaz pre MongoDB

```
db.marriages.aggregate([
  {
    $group: {
      _id: {_id_person: "$groom._id_person", surname: "$groom.surname"},
      total: {$sum: 1}
    }
  },
  {$match: {total: {$gt: 1}}},
```

```
  {$sort: {total: -1}}
])
```



Obr. 4.5: **Výsledky testu č. 2:** Náročnejšia agregácia robí databáze MongoDB väčšie problémy ako PostgreSQL. Relačná databáza tu víťazí pri väčšom počte matričných záznamov pre agregovanie.

Test č. 3

Zadanie: Zobraz všetky úmrtné záznamy z archívu *ARCH2* pre mesto *Brno*, ktoré má na starosti zapisovateľ *Slavomír Sedláček* a nie sú dokončené. Záznamy zoraď podľa čísla signatúry.

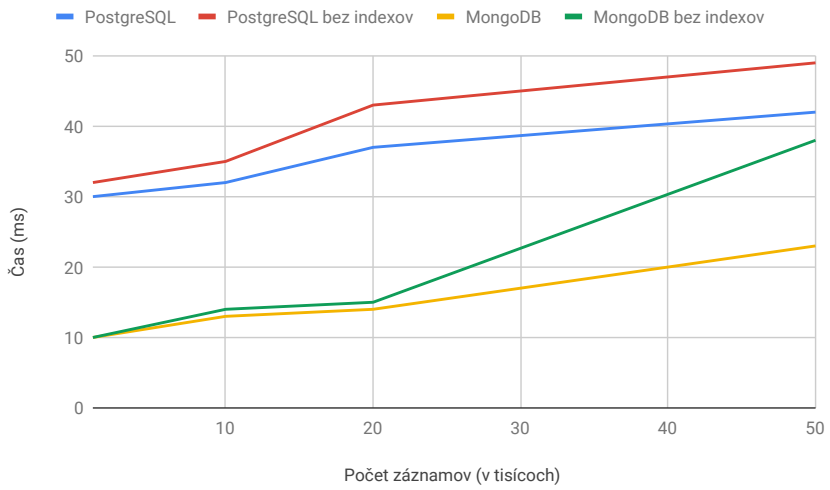
Výpis 4.5: Príkaz pre PostgreSQL

```
SELECT archive, signature, _id_death
FROM "Register"
JOIN "Death" ON "Register"."_id_register"="Death"."register_id"
JOIN "User" ON "User"."_id_user"="Death"."user_id"
WHERE "Register"."archive"='ARCH2'
  AND "Death"."death_village"='Brno'
  AND "Death"."rec_ready"=false
  AND "User"."name"='Slavomír Sedláček'
ORDER BY signature;
```

Výpis 4.6: Príkaz pre MongoDB

```
db.deaths.find({ // filter pre zhodu
  "register.archive": 'ARCH2',
  death_village: 'Brno',
  rec_ready: false,
  "user.name": 'Slavomír Sedláček'
}, { // projekcia - _id dokumentu je vypisované implicitne
  "register.archive": 1,
```

```
  "register.signature": 1
})
.sort({"register.signature": 1}) // zoradiť vzostupne (-1 zostupne)
```



Obr. 4.6: **Výsledky testu č. 3:** Rýchlosť MongoDB bez použitia indexov výrazne diverguje kvôli viacerým filtrovacím podmienkam, ktoré nemajú index a tým pádom musí databáza preskenovať celú kolekciu dokumentov.

Test č. 4

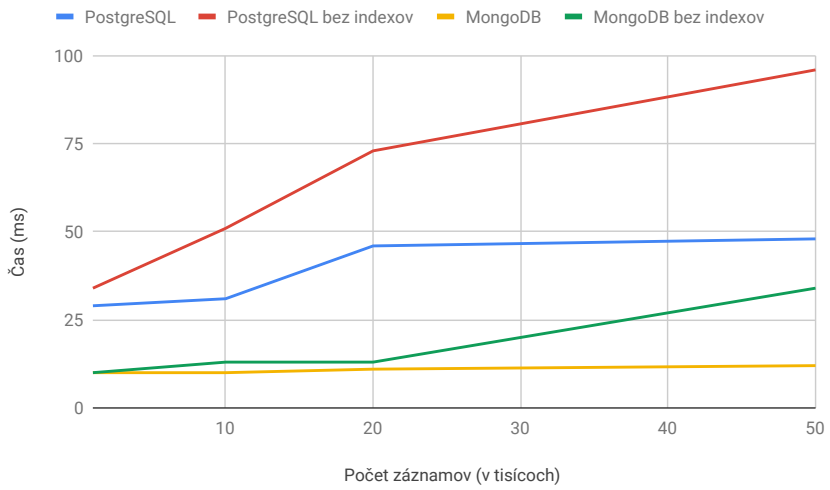
Zadanie: Zisti, ako sa volali deti zosnulého *Michala Čermáka* z *Havířova*.

Výpis 4.7: Príkaz pre PostgreSQL

```
SELECT "Person"."_id_person",
    string_agg(n.name, ' ') as kid_name,
    "Person"."surname"
FROM "Person"
JOIN "PersonName" ON "Person"."_id_person"="PersonName"."person_id"
JOIN "Name" AS n ON "PersonName"."name_id"=n."_id_name"
WHERE father_id = (
  SELECT DISTINCT ON (surname)_id_person
  FROM "Death"
  JOIN "Person" ON "Death"."person_id"="Person"."_id_person"
  JOIN "PersonName" ON "Person"."_id_person"="PersonName"."person_id"
  JOIN "Name" ON "PersonName"."name_id"="Name"."_id_name"
  WHERE "Person"."surname"='Čermák'
    AND "Name"."name"='Michal'
    AND "Death"."death_village"='Havířov'
)
GROUP BY 1;
```

Výpis 4.8: Príkaz pre MongoDB

```
db.deaths.find({
  "death_village": 'Havířov',
  "person.surname": 'Čermák',
  "person.name": 'Michal'
}, {
  "_id": 0, // nezobrazuj _id
  "kids.name": 1,
  "kids.surname": 1
})
```



Obr. 4.7: **Výsledky testu č. 4:** Aj kvôli dotazom takéhoto typu je dôležité indexovať. Časy vyhľadávania vo veľkom množstve záznamov pre obe databázy letia strmo nahor.

Test č. 5

Zadanie: Zisti v ktorých rokoch zomrelo najviac detí do 18 rokov. Záznamy zoraď zostupne.

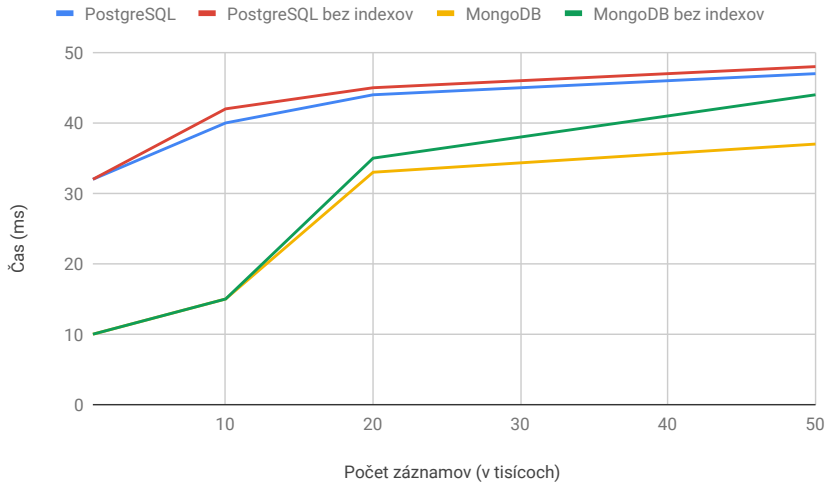
Výpis 4.9: Príkaz pre PostgreSQL

```
SELECT COUNT(_id_death) as dc, EXTRACT(YEAR FROM "death_date") as y
FROM "Death"
WHERE "death_date" IS NOT NULL AND "Death"."age_y" < 18
GROUP BY y
ORDER BY dc DESC;
```

Výpis 4.10: Príkaz pre MongoDB

```
db.deaths.aggregate([
  {$match: {age_y: {$lt: 18}, death_date: {$ne: null}}},
  {
    $group: {
      _id: {$year: {$dateFromString: {dateString: "$death_date"}}},
```

```
      total: {$sum: 1}
    }
  },
  {$sort: {total: -1}}
])
```



Obr. 4.8: **Výsledky testu č. 5:** Pozorujúc stúpanie času MongoDB bez indexov je vyvoditeľné, že pri väčšom počte záznamov bude na PostgreSQL strácať. Znova sa potvrdzuje dôležitosť indexov v MongoDB.

Test č. 6

Zadanie: Vypíš všetky príčiny úmrtia žien v rokoch 1850 až 1860. Zoraď od najvyššieho počtu pre danú príčinu.

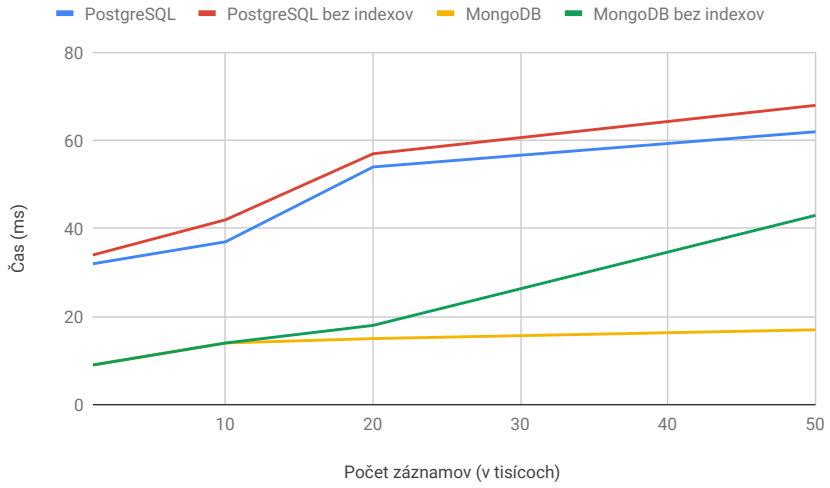
Výpis 4.11: Príkaz pre PostgreSQL

```
SELECT COUNT(death_cause) AS ct, death_cause
FROM "Death"
JOIN "Person" ON _id_person=person_id
WHERE death_cause IS NOT NULL
  AND sex='žena'
  AND EXTRACT(YEAR FROM "death_date") BETWEEN 1850 AND 1860
GROUP BY death_cause
ORDER BY ct DESC;
```

Výpis 4.12: Príkaz pre MongoDB

```
db.deaths.aggregate([
  {
    $match: {
      "person.sex": 'žena',
      death_date: {$gte: '1850-01-01',$lt: '1860-01-01'},
      death_cause: {$ne: null}
```

```
    }
  },
  {$group: {_id: "$death_cause", total: {$sum: 1}}},
  {$sort: {total: -1}}
])
```



Obr. 4.9: **Výsledky testu č. 6:** Agregovanie veľkého počtu dokumentov je pre MongoDB bez použitia indexov znova náročné. Na druhej strane, s užívateľsky vytvorenými sekundárnymi indexami je od PostgreSQL výrazne rýchlejšie.

Test č. 7

Zadanie: Koľko svadieb vykonal oddávajúci *Dominik Urban* v roku 1852?

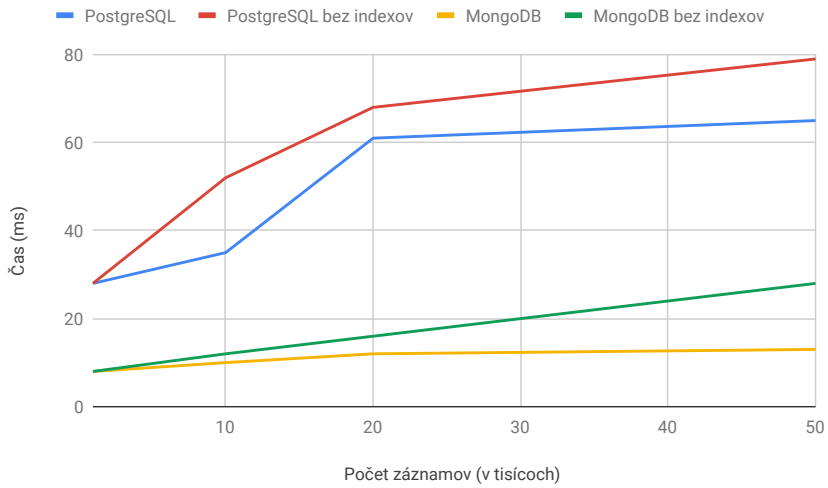
Výpis 4.13: Príkaz pre PostgreSQL

```
FROM "Officiant"
JOIN "OfficiantName" ON officiant_id=_id_officiant
JOIN "Name" ON _id_name=name_id
JOIN "Marriage" ON "Marriage"."officiant_id"=_id_officiant
WHERE "name"='Dominik'
    AND surname='Urban'
    AND EXTRACT(YEAR FROM date)=1852
GROUP BY _id_officiant;
```

Výpis 4.14: Príkaz pre MongoDB

```
db.marriages.aggregate([
  {
    $match: {
      "officiant.name": 'Dominik',
      "officiant.surname": 'Urban',
      date: {$gte: '1852-01-01', $lte: '1852-12-31'}
    }
```

```
  },
  {$count: "records"}
])
```



Obr. 4.10: **Výsledky testu č. 7:** Ďalšie vyhľadávanie s agregáciou je pre MongoDB s indexami otázkou zhruba 15ms aj pre 50 tisíc matričných záznamov.

Test č. 8

Zadanie: Kto boli svedkovia na svadbe *Jana Nováka* a *Edity Hájekovej*. Vypíš meno (mená) a priezvisko.

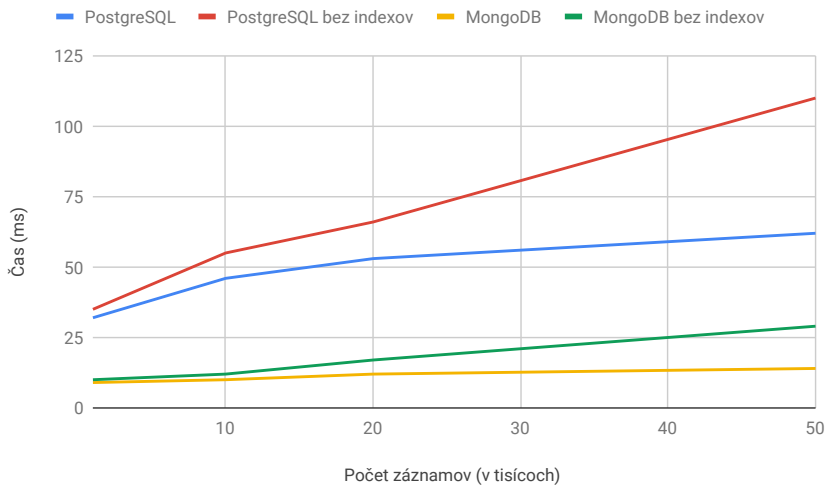
Výpis 4.15: Príkaz pre PostgreSQL

```
SELECT wp._id_person, wp.surname, string_agg(wn.name, ' ') AS witness_name
FROM "Person" AS wp
JOIN "PersonName" AS wpn ON wp._id_person=wpn.person_id
JOIN "Name" AS wn ON wpn.name_id=wn._id_name
JOIN "Witness" as wit ON wp._id_person=wit.person_id
WHERE wit.marriage_id = (
  SELECT _id_marriage
  FROM "Marriage" AS mar
  JOIN "Person" AS gro ON mar.groom_id=gro._id_person
  JOIN "PersonName" AS gpn ON gro._id_person=gpn.person_id
  JOIN "Name" AS gn ON gpn.name_id=gn._id_name
  WHERE gro.surname='Novák' AND gn.name='Jan' AND mar.bride_id IN (
    SELECT _id_person
    FROM "Person" AS bri
    JOIN "PersonName" AS bpn ON bri._id_person=bpn.person_id
    JOIN "Name" AS bn ON bpn.name_id=bn._id_name
    WHERE bri.surname='Hájeková' AND bn.name='Edita'
  )
)
```

```
GROUP BY wp._id_person;
```

Výpis 4.16: Príkaz pre MongoDB

```
db.marriages.find({
  "groom.name": 'Vítězslav',
  "groom.surname": 'Bednář',
  "bride.name": 'Edita',
  "bride.surname": 'Hájeková'
}, {
  "witnesses.surname": 1,
  "witnesses.name": 1
})
```



Obr. 4.11: **Výsledky testu č. 8:** Zložitosť zostavovania dotazu na vyhľadanie v relačnej databáze stúpa každým vzťahom. Tiež je vidieť, že viac JOIN operácií spomaľuje relačnú databázu oproti dokumentovej dosť výrazne.

Test č. 9

Zadanie: Ktoré svadby v Prahe v rokoch 1850 - 1860 boli medzi príbuznými?

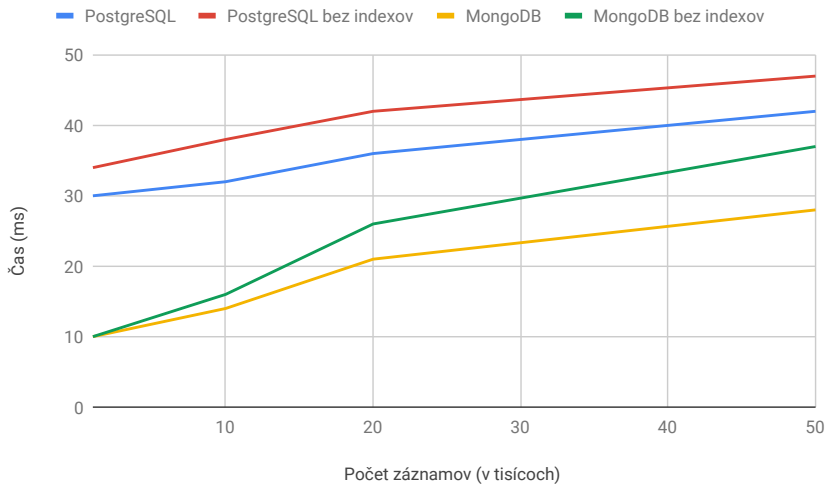
Výpis 4.17: Príkaz pre PostgreSQL

```
SELECT _id_marriage, groom_id, bride_id, relationship
FROM "Marriage"
WHERE EXTRACT(YEAR FROM "date") BETWEEN 1850 AND 1859
    AND village='Praha'
    AND relationship!='ne'
ORDER BY relationship;
```

Výpis 4.18: Príkaz pre MongoDB

```
db.marriages.find({
```

```
  village: 'Praha',
  relationship: {$ne: 'ne'},
  date: {
    $gte: '1850-01-01',
    $lt: '1860-01-01'
  }
}, {
  "groom._id_person": 1,
  "bride._id_person": 1,
  relationship: 1
}).sort({relationship: 1})
```



Obr. 4.12: **Výsledky testu č. 9:** V relatívne jednoduchom `SELECT`, resp. `find()` príkaze majú oba databázové systémy podobnú tendenciu stúpania času vyhľadania požadovaných záznamov.

Test č. 10

Zadanie: Nájdi svadobné záznamy, v ktorých je neplnoletá nevesta. Vypíš mená a priezviská nevesty, ženícha, rodičov a oddávajúceho.

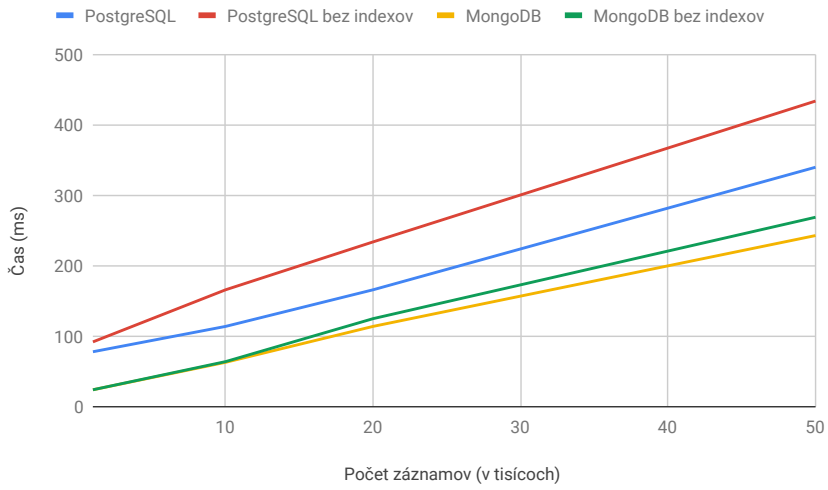
Výpis 4.19: Príkaz pre PostgreSQL

```
SELECT DISTINCT ON (_id_marriage) _id_marriage, mar.bride_y,
    o.surname, ofn.name, bp.surname, bn.name, gp.surname,
    gn.name, bmp.surname, bmn.name, bfp.surname, bfn.name,
    gmp.surname, gmn.name, gfp.surname, gfn.name
FROM "Marriage" as mar

JOIN "Officiant" as o ON mar.officiant_id=o._id_officiant
JOIN "OfficiantName" as oon ON o._id_officiant=oon.officiant_id
JOIN "Name" as ofn ON oon.name_id=ofn._id_name
```

```
JOIN "Person" as bp ON mar.bride_id=bp._id_person
JOIN "PersonName" as bpn ON bp._id_person=bpn.person_id
JOIN "Name" as bn ON bpn.name_id=bn._id_name

JOIN "Person" as gp ON mar.groom_id=gp._id_person
JOIN "PersonName" as gpn ON gp._id_person=gpn.person_id
JOIN "Name" as gn ON gpn.name_id=gn._id_name

JOIN "Person" as bmp ON bp.mother_id=bmp._id_person
JOIN "PersonName" as bmpn ON bmp._id_person=bmpn.person_id
JOIN "Name" as bmn ON bmpn.name_id=bmn._id_name

JOIN "Person" as bfp ON bp.father_id=bfp._id_person
JOIN "PersonName" as bfpn ON bfp._id_person=bfpn.person_id
JOIN "Name" as bfn ON bfpn.name_id=bfn._id_name

JOIN "Person" as gmp ON gp.mother_id=gmp._id_person
JOIN "PersonName" as gmpn ON gmp._id_person=gmpn.person_id
JOIN "Name" as gmn ON gmpn.name_id=gmn._id_name

JOIN "Person" as gfp ON bp.father_id=gfp._id_person
JOIN "PersonName" as gfpn ON gfp._id_person=gfpn.person_id
JOIN "Name" as gfn ON gfpn.name_id=gfn._id_name

WHERE mar.bride_adult=false;
```

Výpis 4.20: Príkaz pre MongoDB

```
db.marriages.find({
  bride_y: {$lt: 18}
}, {
  bride_y: 1,
  "officiant.surname": 1,
  "officiant.name": 1,
  "bride.surname": 1,
  "bride.name": 1,
  "groom.surname": 1,
  "groom.name": 1,
  "bride.mother.surname": 1,
  "bride.mother.name": 1,
  "bride.father.surname": 1,
  "bride.father.name": 1,
  "groom.mother.surname": 1,
  "groom.mother.name": 1,
  "groom.father.surname": 1,
  "groom.father.name": 1,
})
```



Obr. 4.13: **Výsledky testu č. 10:** V poslednom príklade je znázornený dôležitý rozdiel. Nejde ani tak o rýchlosť, ale o zložitosť vytvorenia takéhoto príkazu v prípade relačnej databázy. Kvôli požiadavkam aplikácie pre viac mien, či povolaní vzniká oveľa viac vzťahov v relačnej databáze a tým pádom pre mená, či povolania o 2 JOIN operácie pre každú entitu osoby naviac. Zatiaľ, čo dokumentová databáza žiadne spájanie tabuliek nepotrebuje, pretože všetko je v jednom dokumente.

4.5 Zhrnutie porovnania

Na záver tejto kapitoly nasleduje zhrnutie porovnávania vybraných databázových systémov na základe určených metrík v 4.1 a výberu databázy, ktorá je vhodnejšia pre uchovávanie údajov z matrík.

Flexibilita návrhu a škálovanie

Z hľadiska jednoduchosti návrhu a flexibility uchovávaných dát, ktorá pravdepodobne bude v dohľadnej budúcnosti nutná aj pre dáta v genealógii by som sa v riešení uchovávania týchto dát priklonil k použitiu MongoDB.

Čo sa týka dlhodobého rastu objemu dát, tiež je jasným víťazom práve dokumentová databáza, ktorá ponúka možnosť lacnejšieho a podstatne menej obmedzeného horizontálneho škálovania. Pravdou ale je, že je zatiaľ otázne, kam až projekt Acta Publica postupom času dospeje. Ak sa s istotou stačí zamerať len na staré záznamy v rámci južnej Moravy, škálovanie s veľkou pravdepodobnosťou nebude potrebné uvažovať a tým pádom je možné použiť aj variantu relačného modelu pre PostgreSQL.

Vkladanie a úprava záznamov

Rýchlosti vkladania záznamov, či ich úpravy pre účely matrík nemajú veľké opodstatnenie, pretože dáta sa do nich vložia raz a väčšina z nich už nebude nikdy upravovaná. Za zmienku stojí akurát rýchlosť hromadného vkladania takýchto záznamov, ktorá sa môže hodiť pri migrácii dát. V tomto smere však znova z testovania vyplýva víťaz - MongoDB.

Rýchlosť vyhľadávania, indexy a veľkosť

Z výstupu testov môžeme jasne vyvodiť záver, že pre vytvorené modely databázových systémov je MongoDB rýchlejšie vo všetkých vyhľadávaniach. Na druhej strane, PostgreSQL (s indexami, aj bez nich) zvládol do pol sekundy všetky dotazy až pre 50 tisíc matričných záznamov, čo činí až 630 tisíc záznamov vo všetkých tabuľkách. Z grafov je vidieť, že v porovnaní s MongoDB, sa čas vyhľadania záznamu relačnej databázy drží vyššie a v mnohých prípadoch stúpa rýchlejšie. V prípade miliónov záznamov je možné vďaka týmto údajom predpokladať, že rýchlosť dokumentovo orientovanej databázy je pre vyhľadávanie podstatne väčšia ako rýchlosť databázy využívajúcej koncept vzťahov.

Použitie indexov je ďalšia vec, ktorá je hodná zamyslenia. Ako môžeme vidieť z výsledkov vyhľadávacích testov, v MongoDB užívateľsky vytvorené indexy začali výrazne zrýchľovať vyhľadávanie už pri 20 tisícoch matričných záznamov. V prípade oboch databáz bez indexov, PostgreSQL zvláda vyhľadávanie lepšie pre väčší počet záznamov. MongoDB využívajúce len, viackrát spomínané, `_id` dokumentu bez užívateľsky definovaných indexov bude pri stovkách tisícoch záznamov pomalšie. Na druhej strane, ak sú vytvorené vhodné indexy, veľkosť databázy neovplyvnia až tak ako v prípade PostgreSQL a tým pádom je MongoDB bez pochýb víťazom tohto porovnania.

Kapitola 5

Záver

Úlohou tejto práce bolo na základe naštudovania matričných záznamov svadieb a úmrtí navrhnúť a porovnať vhodnosť relačných, resp. dokumentových databáz pre uchovávanie takýchto údajov. Pre účely porovnania dvoch diametrálne odlišných konceptov pre uchovávanie dát boli zvolené voľne dostupné databázové systémy PostgreSQL, ako zástupca relačných databáz a MongoDB, ako zástupca dokumentovo orientovaných databáz. Primárne boli zvolené na základe popularity, ich využívania niekoľkými svetovými technologickými gigantami, voľnej dostupnosti a sľubovanému veľkému výkonu, ktorý sa neskôr potvrdil u oboch systémov.

V prvej časti práce boli popísané a porovnané vlastnosti relačných a dokumentových databáz vo všeobecnosti. Ďalej bolo vysvetlené indexovanie, ktoré do značnej miery ovplyvňuje ich výkon a ukážkami bola porovnaná syntax operácií nad týmito databázami.

Ďalšia časť pozostávala z uvedenia do problematiky matrík a ich digitalizácie v Českej Republike, po ktorej nasledovala analýza matričných údajov o svadbách a úmrtiach. Na základe analyzovaných matričných záznamov bol vytvorený relačný model pomocou ER diagramu a naznačená flexibilná schéma kolekcií pre úmrtné a svadobné záznamy.

Tretia a posledná časť tejto práce bola venovaná popisu vlastného riešenia generovania testovacích matričných záznamov a nakoniec vytvoreniu sady metrík, z ktorých hlavné boli použiteľnosť návrhu a rýchlosť vyhľadávania v testovaných databázach. Výkon čítania a agregovania dát v PostgreSQL a v MongoDB bol zisťovaný desiatimi testovacími úlohami s vypísanými príkazmi a grafmi, ktoré zahŕňali výsledky týchto testov pre každú databázu s využitím indexov a bez nich.

Pre účely uchovávania matričných údajov bola na základe porovnania vlastností a výkonnostných testov určená, ako vhodnešia, dokumentová databáza - MongoDB.

Na záver by som chcel dodať, že vďaka tejto práci som sa naučil lepšie rozmýšľať nad rôznymi aspektami modelovania databázových systémov pomocou relačných diagramov, spoznal svet dokumentových databáz a naučil sa používať MongoDB a osviežil si programovanie v JavaScripte a môžem povedať, že všetky tieto poznatky ešte v blízkej budúcnosti určite využijem.

Literatúra

[1] Chodorow, K.: *MongoDB: The Definitive Guide*, kapitola 2. O'Reilly, druhé vydanie, 2013.

[2] Codd, E.: A relational model of data for large shared data banks. *Communications of the ACM*, ročník 26, č. 1, 1983: s. 64–69, ISSN 1557-7317.

[3] Comeford, A.: What is cursor in MongoDB and its use? Online; navštívené 7.4.2019.
URL https://www.quora.com/What-is-cursor-in-MongoDB-and-its-use

[4] Darwen, H.: *An Introduction to relational Database Theory*. Ventus Publishing ApS, 2012, ISBN 978-87-403-0202-8.

[5] Method of calculating the scores of the DB-Engines Ranking. Online; navštívené 7.4.2019.
URL https://db-engines.com/en/ranking_definition

[6] The document database defined. Amazon AWS Documentation.
URL https://aws.amazon.com/nosql/document/

[7] Elmasri, R.; Navathe, S. B.: *Fundamentals of Database System*, kapitola 18.3. Addison-Wesley, 6 vydanie, 2011, ISBN 978-0-136-08620-8.

[8] Gray, J.: The Transaction Concept: Virtues and Limitations. 1981.

[9] Inbreeding.
URL http://en.wikipedia.org/wiki/Inbreeding

[10] Zákon č. 301/2000 Sb. o matrikách.

[11] Momjian, B.: *PostgreSQL Introduction and Concepts*, kapitola 1. Addison-Wesley, 2001, ISBN 0-201-70331-9.

[12] Momjian, B.: *PostgreSQL Introduction and Concepts*, kapitola 11. Addison-Wesley, 2001, ISBN 0-201-70331-9.

[13] Indexes. Official MongoDB Documentation.
URL https://docs.mongodb.com/manual/indexes/

[14] MZA, A. P.: Acta Publica: MZA.
URL http://actapublica.eu/matriky/brno/

[15] NoSQL; Online; navštívené 15.4.2019.
URL http://nosql-database.org/

[16] Platil, D.: Za kvalitní digitalizaci matričních údajů nemůže stát, ale mormoni. Online; navštívené 4.4.2019.
URL https://www.irozhlas.cz/zpravy-domov/za-kvalitni-digitalizaci-matricnich-udaju-nemuze-stat-ale-mormoni_1706241330_pla

[17] Česká genealogická a heraldická společnost v Praze: Digitalizace. Online; navštívené 17.03.2019.
URL http://www.genealogie.cz/aktivity/digitalizace/

[18] Občanský zákoník: Zákon č. 89/2012 Sb. § 656 odst 1, 2 o vzniku manželství.

[19] Tiwari, S.: *Professional NoSQL*. Wrox, prvé vydanie, 2011, ISBN 9780470942246.

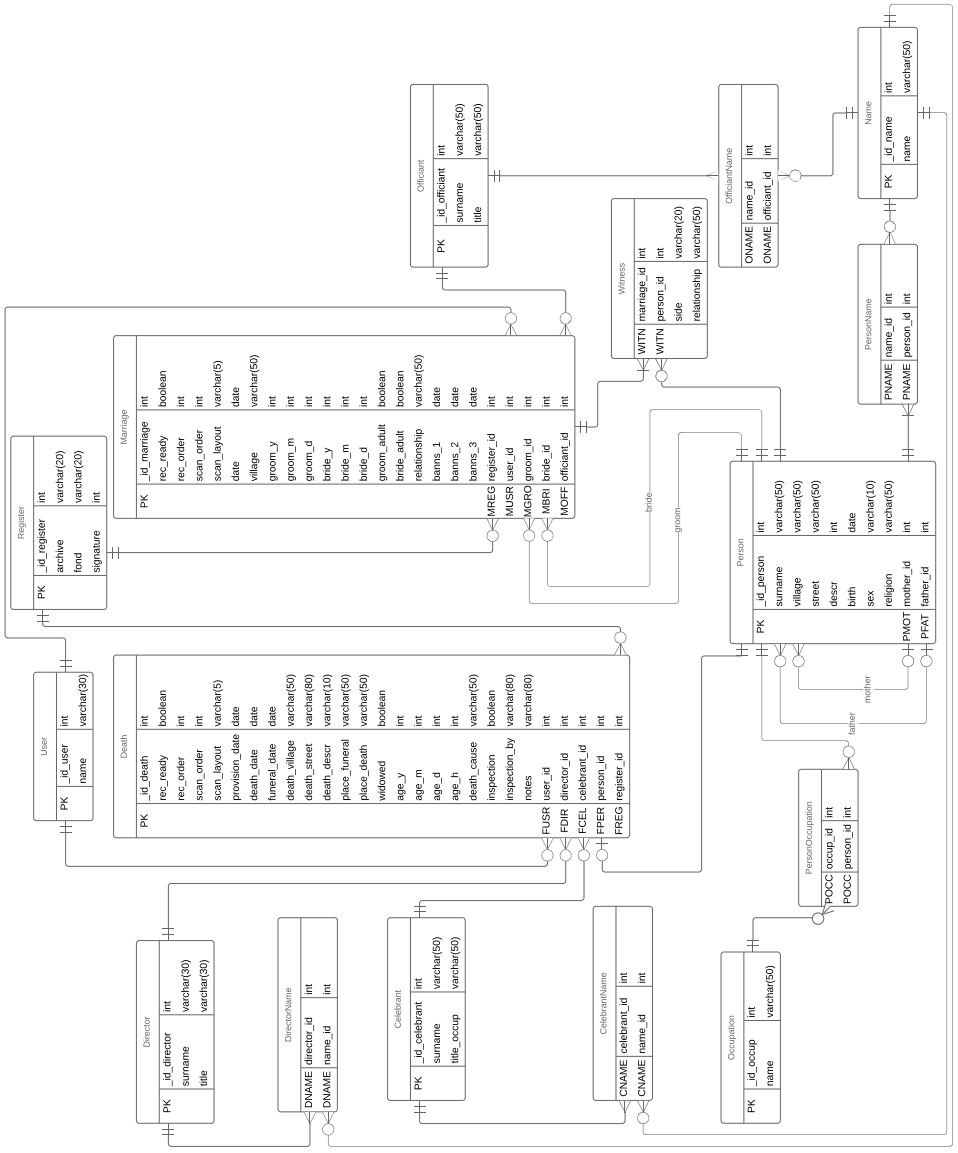
[20] Genealogy.
URL http://en.wikipedia.org/wiki/Genealogy

[21] Zendulka, J.; Rudolfová, I.: *Databázové systémy - IDS*, kapitola 3.4. FIT VUT Brno, 2006.

[22] Oddělení zvláštní matrika.
URL http://www.brno-stred.cz/odbory/odbor-matrika

Príloha A

Model relačnej databázy



Obr. A.1: **ERD**

Príloha B

Obsah priloženého pamäťového média

Priložené CD obsahuje všetky zdrojové súbory pre skompilovanie textu práce a zdrojové súbory implementovaného generátora dát. Adresárová štruktúra je nasledovná:

- **gen** - zdrojové súbory generátora vrátane manuálu v `README.md`
- **pdf** - PDF textu tejto práce
- **tex** - zdrojové súbory LaTeXtejto práce