VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

SROVNÁNÍ DATABÁZOVÝCH SYSTÉMŮ PRO GENEALOGICKÉ ÚČELY

COMPARISON OF DATABASES SYSTEMS FOR GENEALOGY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **DMYTRO SADOVSKYI**
AUTHOR

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

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

155759

Ústav: Ústav inteligentních systémů (UITS)
Student: **Sadovskyi Dmytro**
Program: Informační technologie
Název: **Srovnání databázových systémů pro genealogické účely**
Kategorie: Databáze
Akademický rok: 2023/24

Zadání:

1. Seznamte se s procesem digitalizace matričních údajů a nároky kladenými na práci s těmito daty.
2. Prostudujte problematiku databázových systémů a jejich varianty (relační, objektové, dokumentové, apod.) Seznamte se s volně dostupnými databázovými systémy.
3. Proveďte analýzu požadavků na práci s genealogickými daty a vhodnosti jednotlivých typů databázových systémů.
4. Po dohodě s vedoucím práce vyberte několik systémů pro porovnání. Navrhněte či zvolte metodu pro srovnání těchto systémů pro aplikační doménu genealogických dat.
5. Vytvořte odpovídající modely pro vybrané databázové systémy a vhodnou testovací datovou sadu.
6. Porovnejte zejména výkonnost různých databázových operací a vyhodnoťte vhodnost jednotlivých typů databází pro zpracovávaná data.

Literatura:

- The MongoDB Documentation, dostupné on-line https://docs.mongodb.com/
- Hibatullah Alzahrani. Evolution of Object-Oriented Database Systems. Global Journal of Computer Science and Technology: C, Software & Data Engineering vol. 16, issue 3, 2016.
- E. Olsson, O. Nordkvist. Object Oriented Databases. Dostupné elektronicky http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.462.892&rep=rep1&type=pdf, 2018.

Při obhajobě semestrální části projektu je požadováno:
První tři body zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Kočí Radek, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1.11.2023
Termín pro odevzdání: 9.5.2024
Datum schválení: 6.11.2023

Abstrakt

Tato bakalářská práce se zaměřuje na výběr nejvhodnějšího systému pro správu databází pro ukládání genealogických dat. V rámci práce budou vybrány databáze různých typů, které budou porovnávány podle předem definovaných kritérií. Kritéria hodnocení se zaměří jak na rychlost databáze, tak na uživatelskou přívětivost a licenci, pod kterou je databáze distribuována. Vybrané databáze budou také porovnány se stávající relační databází, a výsledky ukáží, zda je vhodné setrvat u stávajícího modelu, nebo přejít na jiný.

Abstract

This bachelor's thesis aims to select the most suitable database management system for storing genealogical data. The study will involve selecting databases of various types, which will be compared based on predefined criteria. Evaluation criteria will focus on database performance, ease of use, and the licensing terms under which the database is distributed.The chosen databases will also be compared with the current relational database, and the results will indicate whether it is advisable to stay with the current model or switch to a different one.

Klíčová slova

Systémy pro správu databází, grafové databáze, dokumentově orientované databáze, objektově orientované databáze, Neo4j, MongoDB, ZODB, ZEO.

Keywords

Database management systems, graph databases, document-oriented databases, object-oriented databases, Neo4j, MongoDB, ZODB, ZEO.

Citace

SADOVSKYI, Dmytro. *Srovnání databázových systémů pro genealogické účely.* Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Kočí, Ph.D.

Srovnání databázových systémů pro genealogické účely

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Kočího, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Dmytro Sadovskyi
7. května 2024

Poděkování

ímto bych rád poděkoval mému vedoucímu práce panu Ing. Radkovi Kočímu, Ph.D. za rady, čas strávený konzultacemi a za odborné vedení mé bakalářské práce.

Obsah

Kapitola 1

Úvod

Moderní svět informačních technologií prochází bouřlivým vývojem a důležitost efektivního ukládání a správy dat nebyla nikdy tak naléhavá. Genealogické informace, které obsahují údaje o původu rodin a rodů, představují historickou hodnotu a mají aktuální význam pro mnoho lidí po celém světě. Hledání rodinných vazeb, historický výzkum - to vše činí genealogické databáze obzvláště důležitými a výběr nejvhodnějšího systému pro ukládání a správu dat pro tento účel představuje důležitý úkol.

V této bakalářské práci bude provedeno srovnání různých typů systémů pro správu databází (DBMS) s cílem určit nejvhodnější systém pro ukládání a správu genealogických dat. Analyzovány budou NoSQL databáze různých typů, jako jsou grafové, dokumentově orientované a objektově orientované. Hlavním cílem výzkumu je identifikovat nejlepší praxi pro zajištění efektivity a snadného přístupu k genealogickým datům.

Práce se člení na kapitoly. První kapitola slouží k seznámení čtenáře s touto bakalářskou prací. Druhá kapitola slouží k prozkoumání problematiky této práce, a je rozdělena do dvou částí. První část se zabývá genealogií a specifiky genealogických dat. Porozumění specifickým vlastnostem genealogických dat a jejich zvláštnostem umožní lépe přizpůsobit systém ukládání požadavkům této konkrétní oblasti. Dále následuje seznámení s datovými bázemi a jejich typy. Ve třetí kapitole analyzujeme požadavky na databázi, na základě kterých budeme vybírat nejvhodnější. Ve čtvrté kapitole provedeme přenos dat z původní databáze a otestujeme rychlost provádění našich dotazů. V páté kapitole zhodnotíme veškeré informace získané o vybraných DBMS. Ve šesté kapitole zhodnotíme naše dosažené výsledky.

Tento výzkum je aktuální vzhledem k rostoucí popularitě genealogického výzkumu a rodinné historie, stejně jako s růstem objemu informací a rostoucími požadavky na bezpečnost a snadný přístup k datům. Výběr nejvhodnějšího systému pro ukládání dat může výrazně zlepšit uživatelský zážitek.

Kapitola 2

Studium problematiky

Tato část je zaměřena na seznámení s problematikou úkolu. Bude rozdělena do dvou částí. V první části se seznámíme s genealogií, matričními záznamy a specifiky ukládání těchto informací. Ve druhé části se seznámíme s databázemi, jejich typy a vlastnostmi.

2.1 Genealogie

Pojem genealogie pochází z řeckého slova „genos“ (rod, generace) a „logos“ (věda, znalost). Je to starobylá vědecká disciplína, která studuje historii a vztahy jednotlivých rodin. Genealogie zkoumá pokrevní vazby mezi lidmi pocházejícími z jejich společného rodového původu a studuje jejich historické, sociální a biologické důsledky [6].

2.1.1 Matrika

Matriky jsou oficiální záznamy o narozeních, sňatcích a úmrtích, vytvářené církvemi nebo oficiálními orgány. Tyto záznamy se obvykle vedou ve formě knih, ve kterých jsou zaznamenány klíčové události. Matriky jsou cenným zdrojem pro genealogický výzkum, protože poskytují důležitá data pro sledování genealogické linie a historie rodin [9].

Matrika narozených

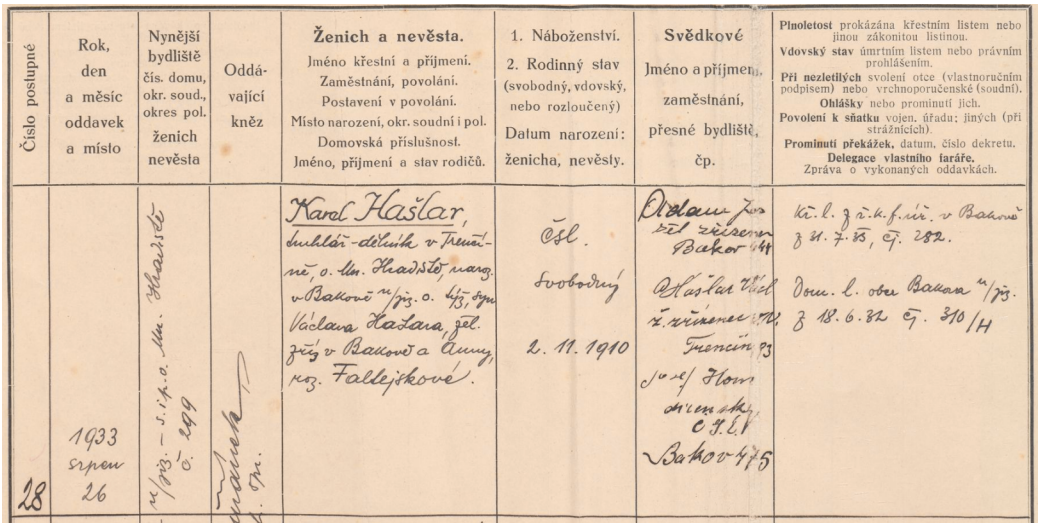
Základní informace obsažené v záznamech by měly zahrnovat zejména datum narození, datum křtu, informace o pokřtěné osobě, jejích rodičích, kmotrech, porodní asistentce a o tom, kdo provedl křest.

Matrika oddaných

Údaje zapsané do těchto matrik zahrnují datum svatby, jméno ženicha, jméno nevěsty, jméno kněze, který provedl obřad, jména svědků, bydliště ženicha a nevěsty, jejich věk a náboženství [9].

Matrika zemřelých

Většinou obsahovaly pouze jméno a příjmení zemřelého, místo bydliště, věk a příčinu smrti.



Číslo postupné	Rok, den a měsíc oddavek a místo	Nynější bydliště čís. domu, okr. soud., okres pol. ženich nevěsta	Oddávající kněz	Ženich a nevěsta. Jméno křestní a příjmení. Zaměstnání, povolání. Postavení v povolání. Místo narození, okr. soudní i pol. Domovská příslušnost. Jméno, příjmení a stav rodičů.	1. Náboženství. 2. Rodinný stav (svobodný, vdovský, nebo rozloučený) Datum narození: ženicha, nevěsty.	Svědkové Jméno a příjmení, zaměstnání, přesné bydliště, čp.	Plnoletost prokázána křestním listem nebo jinou zákonitou listinou. Vdovský stav úmrtním listem nebo právním prohlášením. Při nezletilých svolení otce (vlastnoručním podpisem) nebo vrchnoporučenské (soudní). Ohlášky nebo prominutí jich. Povolení k sňatku vojen. úřadu; jiných (při strážnicích). Prominutí překážek, datum, číslo dekretu. Delegace vlastního faráře. Zpráva o vykonaných oddavkách.
28	1933 srpen 26			Karel Hašlar,	čsl.		

Obrázek 2.1: Záznam z matriční knihy oddaných [1].

2.1.2 Projekt DEMoS

Projekt DEMoS se zabývá digitalizací informací z historických pramenů[7]. V rámci tohoto projektu probíhá digitalizace dat z matričních knih. Jedním z cílů je přenesení všech údajů z matričních záznamů do databáze, což ovlivňuje strukturu databáze, protože stejný typ záznamů se může lišit přítomností různých údajů (podrobnější popis v kapitole 3).

2.2 Databáze

Databáze představují jakýkoli organizovaný soubor dat, který slouží k efektivnímu ukládání, správě a přístupu k informacím.

Systém pro správu databází

Velmi často se pod pojmem databáze rozumí systém pro správu databází. Je potřeba rozlišovat dva termíny. DBMS (systém pro správu databází) je software určený k vytváření, správě a zpracování databází. DBMS zajišťuje strukturování dat v pohodlném formátu, zajišťuje bezpečnost a efektivní přístup k informacím. Umožňují uživatelům provádět dotazy, přidávat, aktualizovat a mazat data v databázích, a také zajišťují integritu a konzistenci dat[12, s.1].

DBMS se objevily v 60. letech v reakci na počítačovou správu komerčních dat. Databáze lze rozdělit podle struktury organizace uložených dat. Mezi hlavní typy patří:

- Relační.
- Dokumentově orientované.
- Klíč-hodnota.
- Grafové.
- Objektově orientované.

- Sloupcové.

Některé z těchto typů se překrývají, např. dokumentově orientované databáze jsou zároveň databázemi klíč-hodnota. Podobná situace je s objektově orientovanou databází ZODB.

2.2.1 Relační databáze

Relační databáze (RDBMS) je organizována podle principu relačního modelu dat, který navrhl Edgar Codd v roce 1970. Hlavními prvky struktury relační databáze jsou tabulky (vztahy), klíče a vztahy mezi tabulkami.

Hlavní komponenty struktury relační databáze jsou[12, s. 37-46]:

- Tabulky (Vztahy): Hlavní složkou relační databáze jsou tabulky, které představují dvourozměrné datové struktury skládající se z řádků a sloupců. Každá tabulka představuje vztah, kde každý řádek odpovídá samostatnému záznamu a každý sloupec atributu nebo charakteristice těchto záznamů.

- Sloupce (Atributy): Každý sloupec tabulky představuje atribut nebo charakteristiku dat uložených v tabulce, např. v tabulce zaměstnanců mohou sloupce představovat ID zaměstnance, jméno, adresu, pozici atd.

- Řádky (Záznamy): Každý řádek tabulky představuje samostatný záznam nebo n-tici dat. Tyto záznamy obsahují skutečné datové hodnoty uložené v tabulce.

- Klíče: Klíče jsou speciální atributy nebo kombinace atributů, které jednoznačně identifikují každý záznam v tabulce. Klíče se používají k zajištění jedinečnosti dat a vazeb mezi tabulkami.

- Vztahy (Vztahy mezi tabulkami): Relační databáze podporují vztahy mezi tabulkami, které jsou definovány pomocí klíčových polí.

Spojení relačních databází s matematikou je založeno na matematické teorii vztahů. Relační model dat je postaven na konceptech z relační algebry a relačního kalkulu, které jsou matematickými formalizmy pro práci se vztahy a operacemi nad nimi. Tyto matematické koncepty poskytují přísný formalizmus pro definování datové struktury, provádění dotazů a zajištění integrity dat v relačních databázích. Relační databáze jsou nejoblíbenější, a protože se pro dotazy do databáze používá jazyk SQL, tak se nerelační databáze nazývají také NoSQL databázemi.

2.2.2 Databáze klíč-hodnota

Protože některé další databáze jsou podmnožinou databází tohoto typu, zmíníme se nejprve o ní. Databáze „klíč-hodnota“ fungují na principu organizace všech dat ve formě sady párů „klíč-hodnota“. Klíč lze považovat za otázku a hodnotu za odpověď.

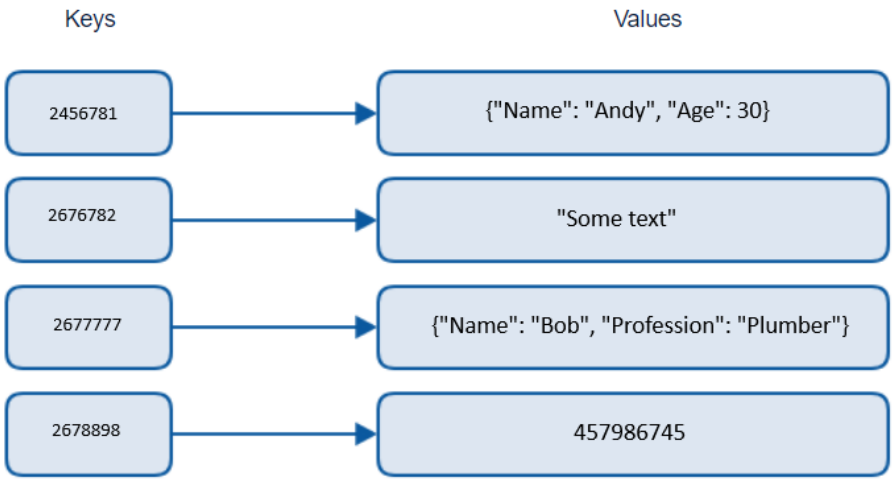
Na rozdíl od tabulek v relační databázi mohou mít volnou strukturu uložených dat. Většina databází „klíč-hodnota“ pracuje podle následujících pravidel:

- Prvek lze identifikovat pouze podle klíče.

- Hašovací funkce deterministicky převede klíč na celé číslo.

- Neprovádíme žádné agregační operace nebo omezujeme jejich rozsah.

- Při aktualizaci se změní celá hodnota, protože databáze nezná schéma ukládání prvků. Existují však některé databáze, které toto pravidlo obcházejí.



Obrázek 2.2: Databáze klíč-hodnota.

Výhody modelu „Klíč-hodnota“[11]:

- Vysoký výkon: Rychlý přístup k datům podle klíče činí tento model vynikající volbou pro ukládání do mezipaměti a rychlé vyhledávání dat.
- Jednoduchost: Jednoduchá struktura dat zjednodušuje vývoj a údržbu aplikací.
- Škálovatelnost: Mnoho databází NoSQL s modelem „Klíč-hodnota“ se může škálovat horizontálně a zpracovávat tak velké objemy dat a vysoké zatížení.
- Flexibilita: Hodnoty mohou být různých typů, což umožňuje ukládat různá data.

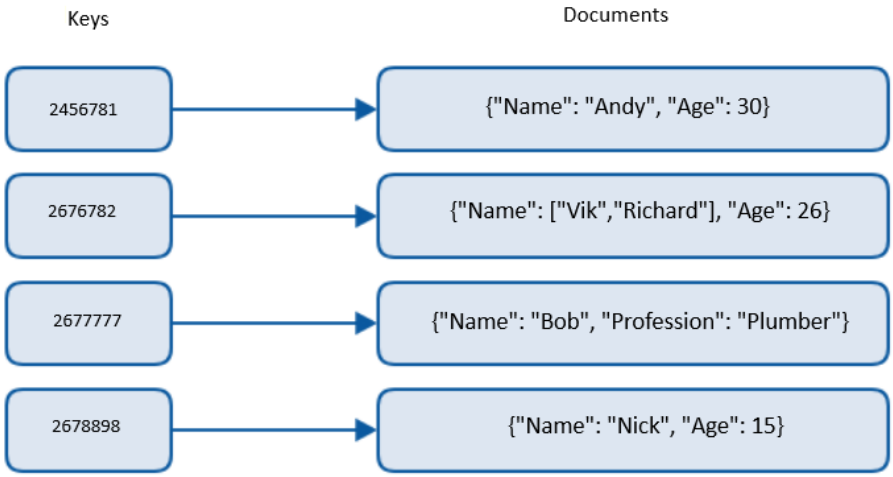
Nevýhody modelu „Klíč-hodnota“:

- Chybí složité dotazy: Tento model omezuje schopnost provádět složité dotazy a analýzy, které jsou charakteristické pro relační databáze.
- Obtížnost zpracování vazeb: Pokud potřebujete ukládat a pracovat s daty, která mají složité vzájemné vztahy, může model „Klíč-hodnota“ vyžadovat další práci.
- Omezení velikosti hodnoty: Některé databáze mohou omezovat velikost hodnot, což může být problém pro ukládání velkých objemů dat.

2.2.3 Dokumentově orientované databáze

Hlavním rozdílem mezi databázemi klíč-hodnota a dokumentově orientovanými databázemi je koncept dokumentu. I když se každá implementace dokumentově orientované databáze liší v detailech této definice, obecně všechny předpokládají, že dokumenty zapouzdřují a kódují data (nebo informace) v nějakém standardním formátu nebo kódování. Používaná kódování zahrnují XML, YAML, JSON a také binární formy, jako je BSON.[10]

To je důležité, protože pochopení formátu dat umožňuje databázi provádět s těmito daty operace na serveru. V případě většiny dokumentových databází to znamená schopnost provádět dotazy na data dokumentu. Při použití známého formátu je také mnohem snazší vytvářet nástroje pro práci s databází, protože data lze zobrazit, prohlížet a upravovat pomocí již známých nástrojů.



Obrázek 2.3: Dokumentově orientovaná databáze.

Výhody dokumentově orientovaných datových modelů:

- Flexibilita: Schéma dat je flexibilní, což umožňuje ukládat a měnit data bez nutnosti měnit celé schéma. To je zvláště užitečné v situacích, kdy se struktura dat v průběhu času mění.
- Výkonné dotazy: Dokumentově orientované databáze poskytují výkonné nástroje pro dotazování dat, včetně možnosti indexování, filtrování a agregace dat.
- Rychlost přístupu: Dokumenty lze rychle načíst pomocí jedinečného identifikátoru, což zajišťuje vysoký výkon.
- Vhodné pro velká data: Mnoho dokumentově orientovaných databází se může škálovat pro zpracování velkých objemů dat a vysokého zatížení.

Nevýhody dokumentově orientovaných datových modelů:

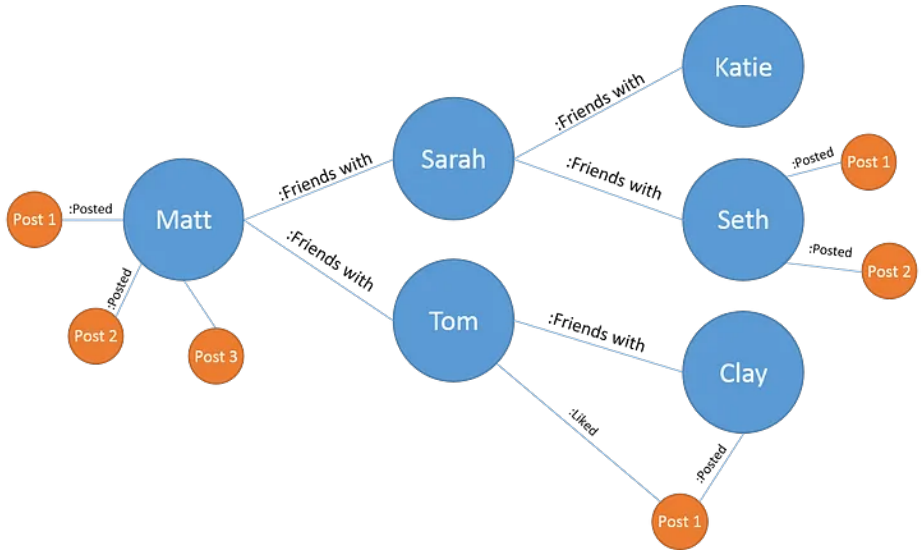
- Obtížnost dotazů mezi dokumenty: Ve scénářích, kde je potřeba provádět složité dotazy související s daty z více dokumentů, se dokumentově orientované databáze mohou ukázat jako méně vhodné.
- Redundance dat: Protože data mohou být duplikována v různých dokumentech, existuje riziko redundance dat a nekonzistence dat mezi dokumenty.
- Nevhodné pro propojená data: V případě, že je potřeba pracovat s daty, která mají složité vzájemné vztahy, jako jsou transakce a vazby mezi entitami, může dokumentový model vyžadovat další zpracování.

2.2.4 Grafové databáze

Grafové databáze představují výkonné nástroje pro modelování a ukládání dat s ohledem na složité vzájemné vztahy. Grafový datový model je postaven na grafu, který se skládá z vrcholů (uzlů) a hran (vazeb) mezi nimi[4].

Důležité koncepty grafových datových modelů zahrnují:

- Vrcholy (uzly): Vrcholy představují datové entity, jako jsou osoby, místa, produkty a další objekty. Každý vrchol může mít jedinečný identifikátor (např. ID uživatele).
- Hrany (vazby): Hrany představují vztahy nebo vazby mezi vrcholy. Mohou mít směr (např. od uživatele k jeho přátelům) a mohou obsahovat informace o typu vazby (např. „je přítelem s“).
- Vlastnosti vrcholů a hran: Každý vrchol a hrana může obsahovat další atributy nebo vlastnosti, např. vrchol uživatele může mít vlastnosti, jako je jméno, věk a umístění.
- Štítky vrcholů a hran: Štítky (labels) umožňují klasifikovat vrcholy a hrany na základě jejich typu nebo účelu, např. vrcholy mohou být označeny jako „uživatel“, „město“ nebo „produkt“ a hrany jako „je přítelem s“, „žije v“ atd.



Obrázek 2.4: Grafová databáze. [8]

Výhody grafových databází:

- Modelování složitých vztahů: Grafové databáze jsou ideální pro modelování dat, kde vztahy mezi entitami hrají důležitou roli. Mohou být např. použity pro analýzu sociálních sítí, doporučujících systémů a sémantických sítí.
- Flexibilní a intuitivní dotazy: Grafové databáze poskytují intuitivní dotazovací nástroje, které umožňují snadno vyhledávat a analyzovat data v grafu. Dotazovací jazyky, jako je Cypher, jsou navrženy speciálně pro práci s grafy.
- Vysoký výkon pro specifické úlohy: V úlohách, kde je potřeba najít vazby mezi vrcholy, mohou grafové databáze poskytovat vysoký výkon a efektivitu.

Nevýhody grafových databází:

- Neefektivita pro určité dotazy: Grafové databáze se mohou ukázat jako neefektivní pro dotazy, které nesouvisejí s analýzou vztahů mezi entitami.
- Obtížnost modelování dat: V případě, že struktura dat nemá vyjádřené vztahy, může být použití grafového modelu nadbytečné a zkomplikovat vývoj a údržbu aplikace.
- Obtížnost škálování: Škálování grafových databází se může ukázat jako složitý úkol, zejména v případě velkých a vysoce zatížených grafů.

2.2.5 Objektově orientované databáze

Termín „objektově orientovaná databáze“ byl poprvé zaveden v roce 1985 a pochází z objektově orientovaného programování. V databázích tohoto typu jsou data modelována jako objekty, jejich atributy, metody a třídy, což zjednodušuje modelování složitých datových struktur[3].

Výhody objektově orientovaných databází:

- Objektově orientovaná paradigma je schopna modelovat objekty reálného světa přirozeným způsobem. To je užitečné při podpoře dat, když je můžeme ukládat jako objekt, a ne jako tabulku, a poté s nimi provádět manipulace.
- Objektově orientované databáze poskytují možnost vytvářet nové datové typy z existujících pomocí konceptů objektově orientovaného programování, jako je dědičnost a polymorfismus. Žádná z těchto funkcí není k dispozici v relačních databázích.
- Objektově orientované databáze jsou za určitých podmínek rychlejší než relační, protože mají vztah mnoho k mnoha a k objektům lze přistupovat pomocí ukazatelů. Kromě toho není potřeba „join“, protože objekty jsou propojeny ukazateli a jakýkoli konkrétní objekt lze najít sledováním řetězce ukazatelů.

Nevýhody objektově orientovaných databází:

- Relační databáze mají pevný datový model, ve kterém jsou data vždy reprezentována jako tabulky. Pro objektově orientované databáze neexistuje žádný takový standard, ve kterém jsou data modelována jako uživatelské objekty a závisí na typu dat a potřebách aplikace.
- V objektově orientovaných databázích chybí standardní dotazovací jazyk.
- Relační databáze jsou založeny na pevných základech relační algebry a relačního kalkulu. Pro objektově orientované databáze neexistuje žádný takový matematický základ.
- Chybí dotazy za běhu nebo uzávěry. Uzávěr je vlastnost relačních databází, která umožňuje vytvářet vnořené dotazy, při kterých se vytvářejí nové tabulky sloučením stávajících a poté se provede dotaz na novou tabulku.

2.2.6 Sloupcové databáze

Klíčovým rozdílem od relačních databází je to, že data jsou uložena nikoli po řádcích, ale po sloupcích. Proto při čtení dat čteme pouze sloupec, který potřebujeme. Kromě toho sloupcové ukládání dat umožňuje výrazně komprimovat data, protože data v jednom sloupci tabulky jsou obvykle stejného typu, což se o řádku říci nedá[5]. Takové databáze jsou vhodné pro analýzu dat, ale špatně se hodí pro častý zápis dat a transakce.

Základní koncepty sloupcových databází zahrnují:

- Sloupce: Data jsou uložena ve formě sloupců, kde každý sloupec představuje samostatný atribut nebo datové pole. To se liší od tradičních relačních databází, kde jsou data uložena ve formě řádků, které představují entity jako celek.
- Rodiny sloupců: Sloupce mohou být seskupeny do rodin sloupců, což umožňuje logicky uspořádat data, např. všechna data o uživatelích mohou být uložena v jedné rodině sloupců a data o produktech v jiné.
- Široké a úzké řádky: V sloupcových databázích mohou být řádky „široké“ nebo „úzké“. Široké řádky obsahují mnoho sloupců, což zajišťuje rychlý přístup k datům. Úzké řádky naopak obsahují pouze několik sloupců a jsou určeny pro rychlé přidávání nových dat.

Výhody sloupcových databází:

- Vysoký výkon: Sloupcové databáze poskytují vysoký výkon pro dotazy, které vyžadují přístup k omezené sadě sloupců. To je zvláště užitečné v analytických aplikacích a zpracování velkých dat.
- Horizontální škálování: Mnoho sloupcových databází, jako jsou Cassandra a HBase, podporuje horizontální škálování, což umožňuje zpracovávat rostoucí objemy dat.
- Efektivní komprese dat: Díky uspořádání dat do sloupců mohou sloupcové databáze efektivně komprimovat data, což snižuje nároky na úložiště.

Nevýhody sloupcových databází:

- Obtížnost dotazů na čtení celého řádku: V případě, že je potřeba číst data z více sloupců jednoho řádku, může být dotaz méně efektivní.
- Obtížnost modelování vztahů: V modelu sloupcových databází mohou být vztahy mezi daty složité pro reprezentaci a zpracování.
- Omezená podpora transakcí: Některé sloupcové databáze mohou mít omezenou podporu transakcí, což je činí méně vhodnými pro aplikace vyžadující vysokou konzistenci dat.

Kapitola 3

Analýza požadavků

V této kapitole se budeme zabývat kritérii pro výběr DBMS a analýzou dotazů, které je potřeba přepsat pro vybrané DBMS.

3.1 Kritéria hodnocení

Pro výběr nejvhodnější databáze v této práci byla definována následující kritéria:

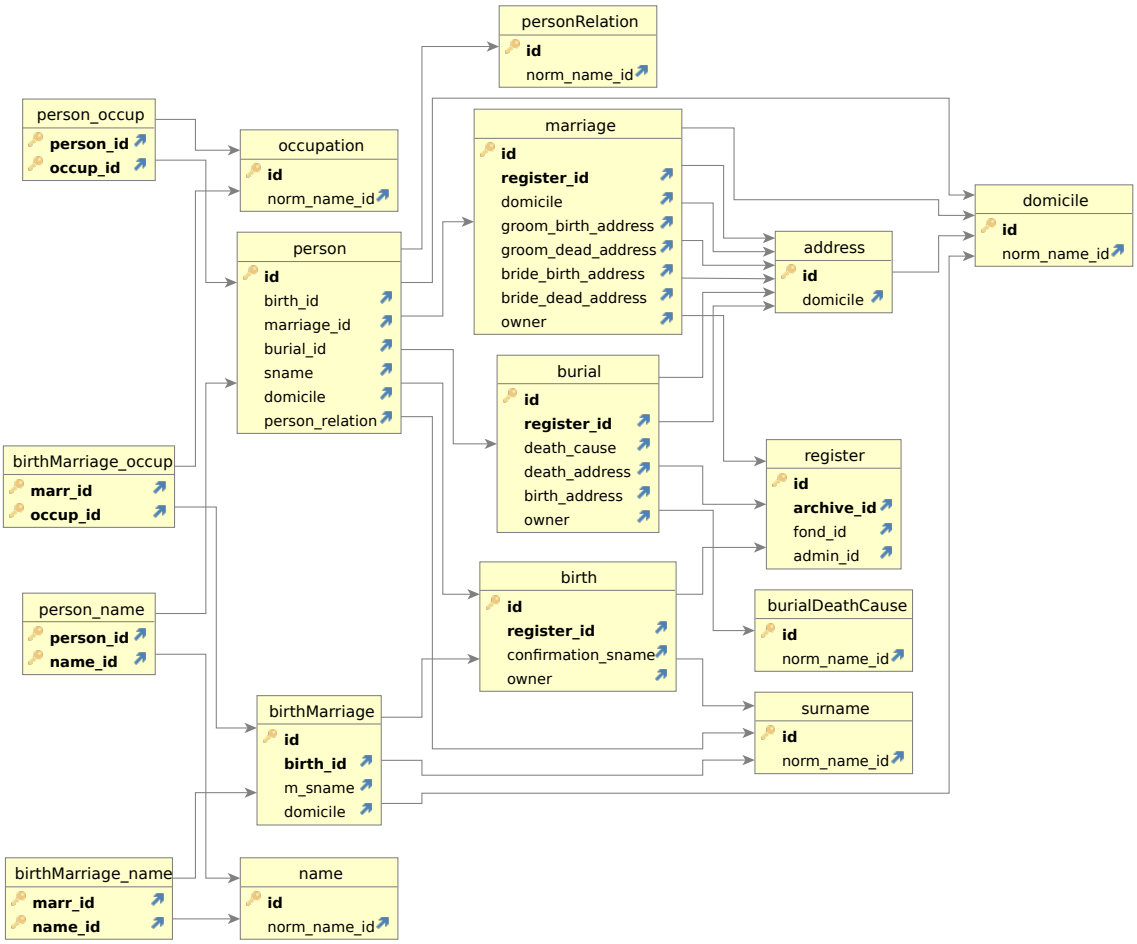
1. Podporované platformy a jazyky: pokud DBMS podporuje různé platformy (Windows, Linux) a existuje možnost použití databáze ve spojení s různými jazyky, bude to výhoda.
2. Licenční politika, náklady: je nutné zvážit licenční omezení a cenové politiky daných DBMS, stejně jako možnost open source.
3. Omezení: omezení CPU, RAM, počtu relací, vazeb, maximální velikost databáze.
4. Prostorová složitost: je potřeba porovnat, kolik místa daná databáze zabere v různých schématech ukládání dat.
5. Časová složitost vybraných operací: na základě vybraných operací bude proveden srovnávací test rychlosti provádění daných operací.
6. Přehlednost schématu a jednoduchost práce s ním: jednoduchost pochopení datového modelu, jednoduchost práce s daným schématem, složitost vytváření dotazů atd. Jedná se o subjektivní aspekt.

Výsledkem této práce může být také závěr, že původní DBMS splňuje dané požadavky lépe než ostatní, a proto není nutné přecházet na jiný DBMS.

3.2 Současná varianta

Prostudování aktuální struktury zdrojové databáze nám pomůže s analýzou dotazů, které je potřeba přepsat. Za tímto účelem bychom se měli podívat na účel konkrétních tabulek a jejich vztahy s matrikami. V současné variantě se používá bezplatná relační databáze MySQL. Schéma dat, která nás zajímají, lze vidět na obrázku 3.1:

Struktura této databáze byla vyvinuta s ohledem na potřeby projektu DEMoS [7]. Projekt má za cíl ukládat záznamy tak, jak byly nalezeny ve zdrojích, včetně chyb, protože



Obrázek 3.1: Základní tabulky.

jeden člověk může mít více záznamů a tyto záznamy se mohou lišit. Z tohoto důvodu mohou existovat různé záznamy osob v databázi pro stejnou osobu. Dalším faktorem ovlivňujícím strukturu databáze je počet atributů, které se v záznamech ne vždy vyskytují, což by vedlo k řídké databázi (většina atributů by obsahovala hodnotu NULL), pokud by byly všechny informace uloženy v jedné tabulce. Dalším důvodem je normalizace dat, která umožňuje zjednodušit zpracování dat a zabránit duplikaci informací.

3.2.1 Tabulka person

Tato tabulka reprezentuje osobu, o které v matrice jde. Může se jednat jak o ústřední osobu záznamu, jako je novorozenec nebo zemřelý, tak o vedlejší osobu, jako je kněz, porodní asistentka nebo rodič. V samotné tabulce máme data, jako je vztah osoby k záznamu, její titul, náboženství, datum narození, úmrtí, informace o tom, zda je osoba stále naživu atd.

Osoba může mít vazby na následující tabulky:

- surname: V této tabulce jsou uložena příjmení.
- name: V této tabulce ukládáme křestní jména. Protože jmen může být více a mají také pořadí, potřebujeme další tabulku pro vytvoření vazby typu „mnoho k mnoha“.

V tomto případě se jedná o tabulku person_name, která obsahuje ID jména, ID osoby a pořadí jména.

- occupation: V této tabulce máme povolání osoby. Protože povolání může být více, potřebujeme stejně jako v případě jmen další tabulku, v tomto případě person_occup.
- personRelation: V této tabulce je uložen vztah osoby k záznamu (např. „jeho manželka“).
- domicile: Zde můžeme získat informace o tom, kde daná osoba žila.

Záznam v tabulce person může být také propojen s tabulkami birth, marriage a burial. Tyto tabulky reprezentují záznamy v matrikách narození, sňatků a úmrtí a obsahují typická data pro tyto matriky.

Všechny tři tabulky mají vazbu na tabulku user. To nám dává vědět, který uživatel danou tabulku zpracoval.

3.2.2 Tabulka birth

Tabulka obsahuje typická data pro záznam o narození a je propojena s tabulkami:

- birth_coinfirm_name - prostřednictvím teto tabulky, tabulka birth je propejena s křestními jmény v tabulce name
- surname - tabulka pro prijmení.
- birthMarriage - tabulka určená pro případy kdy záznam o sňatku se nachází v knize narození. Má analogicky k tabulce person vazbu na tabulky occupation a name prostřednictvím tabulek birthMarriage_occup a birthMarriage_name a vazbu na tabulky surname a domicile.

3.2.3 Tabulka burial

Tabulka obsahuje typické informace pro matriku úmrtí a je propojena s tabulkami:

- burialDeathCause - obsahuje příčinu smrti.
- address - místo narození a místo úmrtí.

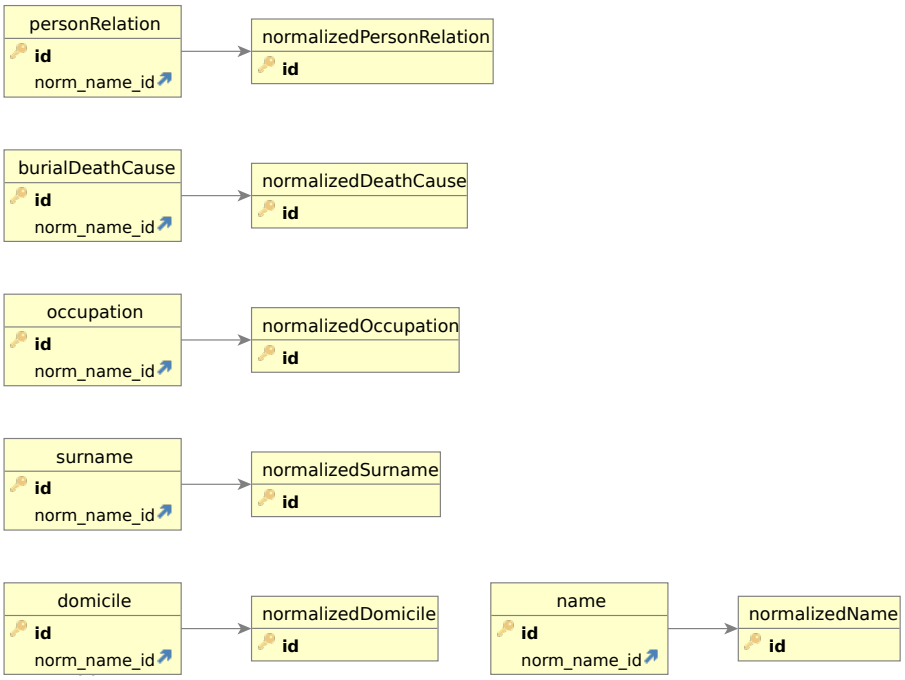
3.2.4 Tabulka marriage

Tabulka je propojena s tabulkami:

- address, která obsahuje informace o adrese narození a úmrtí ženicha a nevěsty.
- domicile, která obsahuje informace o tom, kde se svatba konala

3.2.5 Tabulky pro normalizaci textu

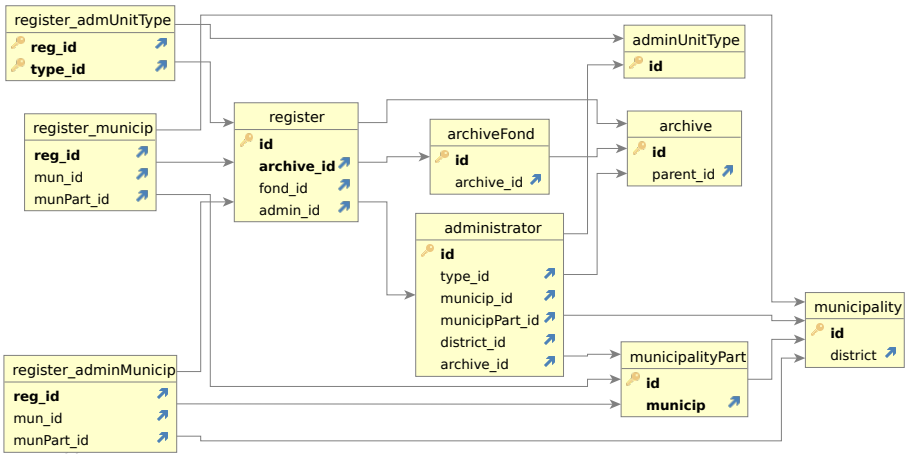
Pro některé záznamy, např. pro povolání nebo jméno, existují tabulky s normalizovanou variantou záznamu (obrázek 3.2). Důvodem je, že v těchto tabulkách je text uložen přesně tak, jak se vyskytuje v původním záznamu, a proto nemusí být vždy jasné, o co přesně jde. To také ztěžuje uživateli vyhledávání.



Obrázek 3.2: Tabulky pro normalizaci textu.

3.2.6 Ostatní tabulky

Tabulky birth, marriage a burial jsou také propojeny s tabulkou register. To nám dává informace o tom, z které knihy záznam pochází, kolik záznamů je v dané matrice, jazyk a podobné údaje. Prostřednictvím tabulek archivefond a archive se můžeme dozvědět informace o tom, odkud byla daná matrika získána (obrázek 3.3).



Obrázek 3.3: Informace o matrice.

Tabulky municipality, distinct, county a state nám dávají informace o umístění matriky.

Taková schéma vede k tomu, že vytváření dotazů pro získání těchto informací je poměrně složité a navíc se provádí velké množství operací JOIN, což ovlivňuje rychlost provádění dotazu. Dalším problémem je, že schéma dat v relační databázi je striktní, zatímco záznamy

v matrikách mohou obsahovat pouze část dat, která jsou v schématu přítomna. To vede k velkému množství NULL polí.

3.3 Analýza dotazů

Nyní, když máme znalost struktury databáze, můžeme analyzovat dotazy, které budeme muset implementovat pro vybrané databáze. Zde budou uvedeny často používané dotazy, na základě kterých budeme schopni pochopit, která databáze bude zpracovávat naše dotazy rychleji.

1. Rekonstrukce záznamu o narození

Dotaz rekonstruuje úplné informace o záznamu o narození a také o tom, kdo ho zpracoval. Zde bude tento dotaz rozložen na části, v příloze B bude poskytnut úplně. Tento dotaz je uloženou procedurou a má jeden parametr - recid (id záznamu o narození).

V prvním fragmentu (vypis 3.1) získáváme jména spojená s narozením a ukládáme data do dočasné tabulky tempNames (dočasná tabulka je speciální typ tabulky, který umožňuje uchovávat dočasnou sadu výsledků, kterou lze znovu použít několikrát během jedné relace).

```
create temporary table tempNames
    select birth.id as id, group_concat(name.name separator ' ') as names
      from birth
        join (birth_confirm_name, name)
             on birth_confirm_name.birth_id=birth.id and
             birth_confirm_name.name_id=name.id
        group by birth.id;
```

Výpis 3.1: První část prvního dotazu.

Dočasná tabulka je užitečná, když nelze nebo je nákladné dotazovat se na data vyžadující jediný operátor SELECT. V těchto případech můžeme použít dočasnou tabulku k uchování okamžitého výsledku a použít jiný dotaz k jeho zpracování. Použitím agregátové funkce group_concat provedeme konkatenaci jmen osoby pro případ, že má více jmen (výpis 3.2).

```
insert into TEMP_personFull(birth_id, marriage_id, burial_id, rel,
    title, snameFull, nameFull, domicileFull, street, descr_num,
    religion, birth_date, dead, waif, category, occNames,
    personRelationFull, widow, legitimate, dead_date, work_place,
    age)
    select person.birth_id, person.marriage_id, person.burial_id,
    rel, title, surname.name as snameFull,
    group_concat(name.name separator ' ') as nameFull,
    domicile.name as domicileFull, street, descr_num, religion,
    birth_date, dead, waif, category, occups.occNames,
    personRelation.name as personRelationFull, widow,
    person.legitimate, dead_date, work_place, age from person
    left join (person_name, name)
        on person_name.person_id=person.id and
        person_name.name_id=name.id and person.birth_id = recid
    left join (surname)
        on person.sname=surname.id
    left join (domicile)
```

```
19          on person.domicile=domicile.id
20      left join (personRelation)
21          on person.person_relation=personRelation.id
22      left join
23          (select person.id, group_concat(occupation.name separator '; ') as
                occNames
24           from person
25           join (person_occup, occupation)
26               on person_occup.person_id=person.id and
27               person_occup.occup_id=occupation.id group by person.id)
28           as occups
29          on person.id=occups.id
30      WHERE birth_id=recid
31      group by rel;
```

Výpis 3.2: Druhá část prvního dotazu.

V této části vkládáme do dočasné tabulky TEMP_personFull úplná data o osobách, jejichž birth_id se rovná zadanému parametru recid, jako je jejich jméno, příjmení, místo bydliště, jejich vztah k záznamu, a také jejich profese (stejně jako v předchozí části dotazu, zde se také používá funkce group_concat, pro získání úplného jména a všech profesí, pokud jich je více).

Vzhledem k tomu, že v poslední části bude použit pohled birthmarriagefull, je nutné uvést příklad kódu 3.3. K zkrácení dotazů byla část dotazu mírně zkrácena nahrazením vybraných polí za *.

```
SELECT
   *
    FROM
        (((((`birthmarriage`
        LEFT JOIN (`birthmarriage_name`
        JOIN `name`) ON ((((`birthmarriage_name`.`marr_id` = `birthmarriage`.`id`)
            AND (`birthmarriage_name`.`name_id` = `name`.`id`))))
        LEFT JOIN (SELECT
            `birthmarriage`.`id` AS `id`,
                GROUP_CONCAT(`occupation`.`name`
                    SEPARATOR '; ') AS `occNames`
        FROM
            (`birthmarriage`
        JOIN (`birthmarriage_occup`
        JOIN `occupation`) ON
        ((((`birthmarriage_occup`.`marr_id` = `birthmarriage`.`id`) AND
        (`birthmarriage_occup`.`occup_id` = `occupation`.`id`))))
        GROUP BY `birthmarriage`.`id`) `occups` ON
        (`birthmarriage`.`id` = `occups`.`id`)))
        LEFT JOIN `surname` ON ((`birthmarriage`.`m_sname` = `surname`.`id`)))
        LEFT JOIN `domicile` ON ((`birthmarriage`.`domicile` = `domicile`.`id`)))
```

Výpis 3.3: Třetí část prvního dotazu.

V posledním fragmentu (výpis 3.4) získáváme různé metainformace, jako je uživatel, který přidal záznam do archivu, fond, a pak získáváme data o osobách na základě jejich vztahu k záznamu pomocí atributu rel, a data z birthMarriage, stejně jako jméno a pří-

jmení, které ukládáme v tabulce tempNames. To nám poskytuje úplné informace o záznamu narození.

```
select *
    from TEMP_personFull, birth as b
    join (register) on b.register_id=register.id
    left join (user) on b.owner=user.id
    left join (archive) on register.archive_id=archive.id
    left join (archiveFond) on register.fond_id=archiveFond.id
    left join (TEMP_personFull as main) on main.birth_id=b.id and
        main.rel='main'
    left join (TEMP_personFull as gr) on gr.birth_id=b.id and gr.rel='granted'
    left join (TEMP_personFull as mw) on mw.birth_id=b.id and mw.rel='midwife'
    left join (TEMP_personFull as f) on f.birth_id=b.id and f.rel='f'
    left join (TEMP_personFull as m) on m.birth_id=b.id and m.rel='m'
    left join (TEMP_personFull as ff) on ff.birth_id=b.id and ff.rel='f_f'
    left join (TEMP_personFull as fm) on fm.birth_id=b.id and fm.rel='f_m'
    left join (TEMP_personFull as fmf) on fmf.birth_id=b.id and fmf.rel='f_m_f'
    left join (TEMP_personFull as fmm) on fmm.birth_id=b.id and fmm.rel='f_m_m'
    left join (TEMP_personFull as mf) on mf.birth_id=b.id and mf.rel='m_f'
    left join (TEMP_personFull as mm) on mm.birth_id=b.id and mm.rel='m_m'
    left join (TEMP_personFull as mmf) on mmf.birth_id=b.id and mmf.rel='m_m_f'
    left join (TEMP_personFull as mmm) on mmm.birth_id=b.id and mmm.rel='m_m_m'
    left join (TEMP_personFull as gf1) on gf1.birth_id=b.id and gf1.rel='gf_1'
    left join (TEMP_personFull as gf2) on gf2.birth_id=b.id and gf2.rel='gf_2'
    left join (TEMP_personFull as gf3) on gf3.birth_id=b.id and gf3.rel='gf_3'
    left join (TEMP_personFull as gf4) on gf4.birth_id=b.id and gf4.rel='gf_4'
    left join (TEMP_personFull as gfrel1) on gfrel1.birth_id=b.id and
    gfrel1.rel='gfrel_1'
    left join (TEMP_personFull as gfrel2) on gfrel2.birth_id=b.id and
    gfrel2.rel='gfrel_2'
    left join (TEMP_personFull as gfrel3) on gfrel3.birth_id=b.id and
    gfrel3.rel='gfrel_3'
    left join (TEMP_personFull as gfrel4) on gfrel4.birth_id=b.id and
    gfrel4.rel='gfrel_4'
    left join (TEMP_personFull as h) on h.birth_id=b.id and h.rel='husband'
    left join (birthMarriageFull as bm1) on bm1.birth_id=b.id and bm1.num=1
    left join (birthMarriageFull as bm2) on bm2.birth_id=b.id and bm2.num=2
    left join
        tempNames as conf_name
        on b.id=conf_name.id
    left join (surname)
        on b.confirmation_sname=surname.id
    WHERE b.id = recid
group by user.name;
```

Výpis 3.4: Čtvrtá část prvního dotazu

2. Matriky obsahující záznamy o narození

Dotaz je zaměřen na hledání matrik, které obsahují záznamy o narozeních. Podoba dotazu je na výpisu 3.5.

```
select register.id, signature
from birth
join register on birth.register_id = register.id
group by register.id;
```

Výpis 3.5: Druhý dotaz.

3. Počet záznamů o narození v matrikách

Dotaz slouží k odhadu počtu záznamů v každé matriční knize. Podoba dotazu je na výpisu 3.6.

```
select count(*) as nums, register.id
from birth
join register on birth.register_id = register.id
group by register.id
order by nums;
```

Výpis 3.6: Třetí dotaz.

4. Záznamy o narození v matrice

Dotaz slouží k vyhledání záznamů o narození ve čtvrté matriční knize. Podoba dotazu je na výpisu 3.7.

```
select archive.short as archive, archiveFond.name as fond,
    register.signature, birth.id, birth.scan, birth.pos, birth.lay
from birth
join (register) on birth.register_id=register.id
left join (archive) on register.archive_id=archive.id
left join (archiveFond) on register.fond_id=archiveFond.id
WHERE register_id=10914
order by scan, pos;
```

Výpis 3.7: Čtvrtý dotaz.

Zde hledáme záznam o narození, který odpovídá identifikátoru hledané matriky, a také příslušející matriku, archiv a fond.

5. Četnost výskytu příjmení

V tomto dotazu počítáme, jak často se v záznamech vyskytují příjmení. Podoba dotazu je na výpisu 3.8.

```
select count(*) as nums, person.sname, surname.name
from person
join surname on person.sname = surname.id
group by person.sname
order by nums;
```

Výpis 3.8: Pátý dotaz.

6. Vyhledávání záznamů podle ID příjmení

Tento dotaz pomůže posoudit rychlost vyhledávání záznamů, ve kterých se vyskytuje osoba s určitým příjmením podle jejího identifikátoru. Podoba dotazu je na výpisu 3.9.

```
select *
from surname
join person
   on person.sname = surname.id
join birth
    on birth.id = person.birth_id
left join (domicile)
    on person.domicile=domicile.id
left join (personRelation)
    on person.person_relation=personRelation.id
left join
    (select person.id, group_concat(occupation.name separator '; ') as occNames
      from person
      join (person_occup, occupation)
          on person_occup.person_id=person.id and
              person_occup.occup_id=occupation.id WHERE person.sname = 34435
              group by person.id)
     as occups
    on person.id=occups.id
left join
    (select person.id, group_concat(name.name separator '; ') as PersonNames
      from person
      join (person_name, name)
          on person_name.person_id=person.id and person_name.name_id=name.id
              WHERE person.sname = 34435 group by person.id)
     as FullName
    on person.id=FullName.id
join (register) on birth.register_id=register.id
left join (archive) on register.archive_id=archive.id
left join (archiveFond) on register.fond_id=archiveFond.id
WHERE surname.id = 34435;
```

Výpis 3.9: Šestý dotaz.

V tomto dotazu se vyhledává příjmení podle ID, poté pomocí operátoru JOIN hledáme lidi, kteří jsou spojeni s tímto příjmením, a informace o osobě, včetně jejích jmen a povolání, a také informace o záznamu.

Pro rekonstrukci jména a povolání se používají vnořené dotazy 3.10.

```
    select person.id, group_concat(name.name separator '; ') as PersonNames
    from person
    join (person_name, name)
    on person_name.person_id=person.id and person_name.name_id=name.id
    WHERE person.sname = 34435
    group by person.id
```

Výpis 3.10: Sedmý dotaz.

Je také důležité omezit hledání podle ID příjmení, protože jinak hledání výrazně zpomaluje.

7. Vyhledávání záznamů podle příjmení

Tento dotaz je možné najít v příloze C. Dotaz je podobný předchozímu, ale vyhledávání se provádí ne podle ID, ale podle samotného příjmení.

8. Četnost výskytu různých forem normalizovaných jmen

Dotaz slouží k počítání četnosti výskytu různých podob normalizovaných jmen. Podoba dotazu je na výpisu 3.11.

```
1  select count(*) as nums, normalizedName.name
2  from name
3  join normalizedName on name.norm_name_id = normalizedName.id
4  group by normalizedName.id;
```

Výpis 3.11: Osmý dotaz.

9. Vyhledávání záznamů podle normalizovaného jména

Tento dotaz je možné najít v příloze D slouží k hledání záznamů o narození, ve kterých je obsažena osoba s hledanou normalizovanou variantou jména. Dotaz je podobný šestému dotazu, ale tentokrát hledáme podle normalizovaného jména a potřebujeme pouze vnořený k získání povolání.

3.4 Použité technologie

3.4.1 Python

Pro práci s databázemi byl použit Python verze 3.11.2. Python je interpretovaný, multi-paradigmatický, dynamicky typovaný programovací jazyk na vysoké úrovni pro všeobecné použití. Tento jazyk byl vybrán, protože se snadno používá a existuje pro něj velké množství ovladačů pro práci s databázemi.

Použité knihovny

- mysql - ovladač pro práci s původní databází MySQL.
- pymongo - ovladač pro práci s databází MongoDB.
- bson - knihovna pro práci s BSON.
- neo4j - ovladač pro práci s grafovou databází Neo4j.
- csv - knihovna pro práci se soubory CSV.
- ZODB - knihovna databáze Knihovna ZODB.
- ZEO - knihovna pro prací s serverem ZEO.
- BTrees - knihovna implementující B-Stromy.
- time - knihovna pro prací s časem.
- requests - knihovna pro odesílání HTTP dotazů.

3.4.2 MySQL

Původní relační databáze distribuovaná pod licencí GNU GPL2. Má konektory pro velké množství programovacích jazyků. Pro dotazy se používá jazyk SQL.

3.4.3 Neo4j

Jedna z nejpopulárnějších grafových databází s otevřeným zdrojovým kódem, implementovaná v Javě. Má ovladače pro mnoho populárních jazyků a také REST API.

V DBMS se používá vlastní dotazovací jazyk - Cypher. Tento jazyk je založen na ASCII art (uzly jsou reprezentovány v kulatých závorkách, vztahy jsou zobrazeny šipkami (orientovanými a neorientovanými), typ vztahu lze zadat v hranatých závorkách) a proto dotazy vypadají poměrně názorně.

```
1  MATCH (u:user)-[:likes]->(:rock)
2  RETURN u
```

Výpis 3.12: Příklad dotazu v jazyce Cypher.

Tento dotaz vrátí všechny uživatele, kteří mají rádi rockový žánr. Tato databáze byla vybrána z několika důvodů:

- Umožňuje snadnou práci se složitými schématy, která mají velké množství vazeb.
- Každý uzel může obsahovat různá data, takže nemusíme ukládat obrovské množství NULL polí.
- Grafová databáze umožňuje snadno vytvářet vazby mnoho k mnoha, což umožní zbavit se nadbytečných tabulek, které máme v původní databázi.

Tento DBMS je distribuován v běžné a cloudové variantě.

Běžná varianta

V běžné variantě existují dvě varianty: Community, která je distribuována pod licencí GPLv3, a Enterprise, bohužel cena druhé varianty není veřejně dostupná. Důležité rozdíly pro nás lze vidět v tabulce 3.1.

	Community	Enterprise
Limit velikosti grafu	34 miliard	Neomezeno
Shlukování a samooprava	Ne	Ano
Vícegrafová architektura	Ne	Ano
Open Source licence	GPLv3	Ne
Podpora	Ne	Ano
Počet podporovaných databází	1	Neomezeno
Podporované platformy	Linux, Windows, MacOS	Linux, Windows, MacOS

Tabulka 3.1: Rodzdily mezí Community a Enterprise

Cloudová varianta

Cloudová varianta má tři plány: Free, Enterprise a Professional. Rozdíly lze vidět v tabulce3.2.

	Free	Professional	Enterprise
Cena	Zdarma	Od 65$	Dohodou
Omezení paměti	200 tis. uzlů a 400 tis. vazeb	64 GB paměti	384 GB paměti
Odolnost proti selhání	Ne	Ano	Ano
SLA	Ne	Ne	99.95%
Automatické zálohování	Ne	Denně	Každou hodinu
On-demand snímky v čase	1	Ano	Ano

Tabulka 3.2: Rodzdily mezí cloduvymi varianty

Rozšíření možností

Neo4j podporuje vytváření vlastních procedur nebo funkcí pomocí jazyka Java v případě potřeby rozšířit možnosti neo4j. Na webu neo4j je také sekce labs, kde jsou k dispozici hotové sady knihoven vytvořené komunitou i zaměstnanci neo4j.

3.4.4 MongoDB

Populární dokumentově orientovaná databáze, která podporuje volné schéma dat, což nám umožňuje zbavit se nadbytečných NULL polí. Distribuuje se pod licencí SSPL. Existuje velké množství oficiálních i neoficiálních ovladačů pro různé programovací jazyky. MongoDB používá vlastní dotazovací jazyk s názvem MQL - MongoDB Query Language, který používá strukturu JSON a JavaScript. Operační systémy: Linux, macOS, Windows a OpenBSD.

3.4.5 ZODB

Objektově orientovaná databáze implementovaná v jazyce Python. Distribuuje se pod licencí Zope Public License. Tato licence je podobná známé licenci BSD, ale ZPL také přidává ustanovení, která zakazují používání ochranné známky a vyžadují dokumentování všech změn. Samotná databáze, stejně jako většina databází tohoto typu, je vestavená, ale existuje ZEO - projekt, který poskytuje server a také knihovnu pro komunikaci s ním.

Tato databáze ukládá serializované objekty Pythonu. Pro uložení libovolného objektu je nutné ho připojit k některému z uložených objektů. Přístupovým bodem k datům v databázi je její kořen - root, přístup k datům se získává pomocí jejich klíče. Autoři nedoporučují ukládat data přímo do kořene, předpokládá se použití jiných struktur, jako jsou slovníky. Je však třeba vzít v úvahu, že při použití slovníku se musí načíst do paměti celý, proto autoři doporučují používat B-Stromy z knihovny BTrees.[2] Je také vhodné zmínit, že v databázi je vhodné ukládat pouze neměnné objekty, takže třídy by měly být dědici třídy Persistent. To souvisí se zvláštností ukládání měnitelných objektů, ukládání odkazů a také sledování změn objektů. V tomto DBMS chybí dotazovací jazyk, takže uživatel musí vyhledávání v databázi provádět sám.

3.4.6 Elasticsearch

Jako databáze byl také zvažován vyhledávač Elasticsearch. Zvážit tuto myšlenku mi navrhl vedoucí této práce pan Kočí Radek. Po vyhledání informací o této možnosti bylo zjištěno, že tento engine jako databázi používají některé obchody. Tato myšlenka byla zamítnuta kvůli nízké rychlosti a složitosti dotazů při vyhledávání objektů s velkým počtem vazeb.

3.5 Návrh

Pro posouzení rychlosti provádění a objemu databáze je potřeba přenést data do dostupných databází.

1. Nejprve je potřeba analyzovat, jak jsou data uložena v konkrétních databázích, abychom měli představu o tom, jak přenést data z tabulek, např. v MongoDB lze „ekvivalentem“ tabulek nazvat kolekce.
2. Musíme extrahovat data z původní databáze a přenést je do cílové.
3. Po přenosu dat je potřeba analyzovat dotazy, pomocí kterých budeme databáze testovat, a vytvořit potřebné indexy.
4. Na základě existujících dotazů do DBMS MySQL je potřeba vytvořit analogické dotazy do vybraných DBMS (v případě ZODB napsat funkce pro vyhledávání v jazyce Python).
5. Poté je potřeba otestovat rychlost provádění dotazů.
6. Po práci s analyzátorem dotazů, v případě zjištění věcí, které lze optimalizovat, je potřeba změnit strukturu na vhodnější, případně přidat potřebné indexy a zopakovat předchozí bod.

Kapitola 4

Implementace

V této kapitole bude popsána metodika a důležité body implementace, analýza možných chyb a jejich oprava. V případě, že je dotaz příliš dlouhý, bude přesunut do příloh, a v této kapitole budou popsány důležité body dotazu.

Také je třeba říci, že i když v některých databázích není nutné přenášet identifikátor záznamu, bylo to uděláno záměrně, protože některé podobné záznamy mohou mít různé identifikátory, např. příjmení „Stikar“ se v tabulce „surname“ vyskytuje 5krát.

4.1 MySQL

Nejprve bylo nutné provést dotazy na původní databázi, abychom měli informace o tom, jak dlouho trvají dotazy na původní databázi. Samotný dump databáze byl získán ve formě SQL skriptu, který je třeba spustit, aby byla databáze naplněna daty.

4.2 Neo4j

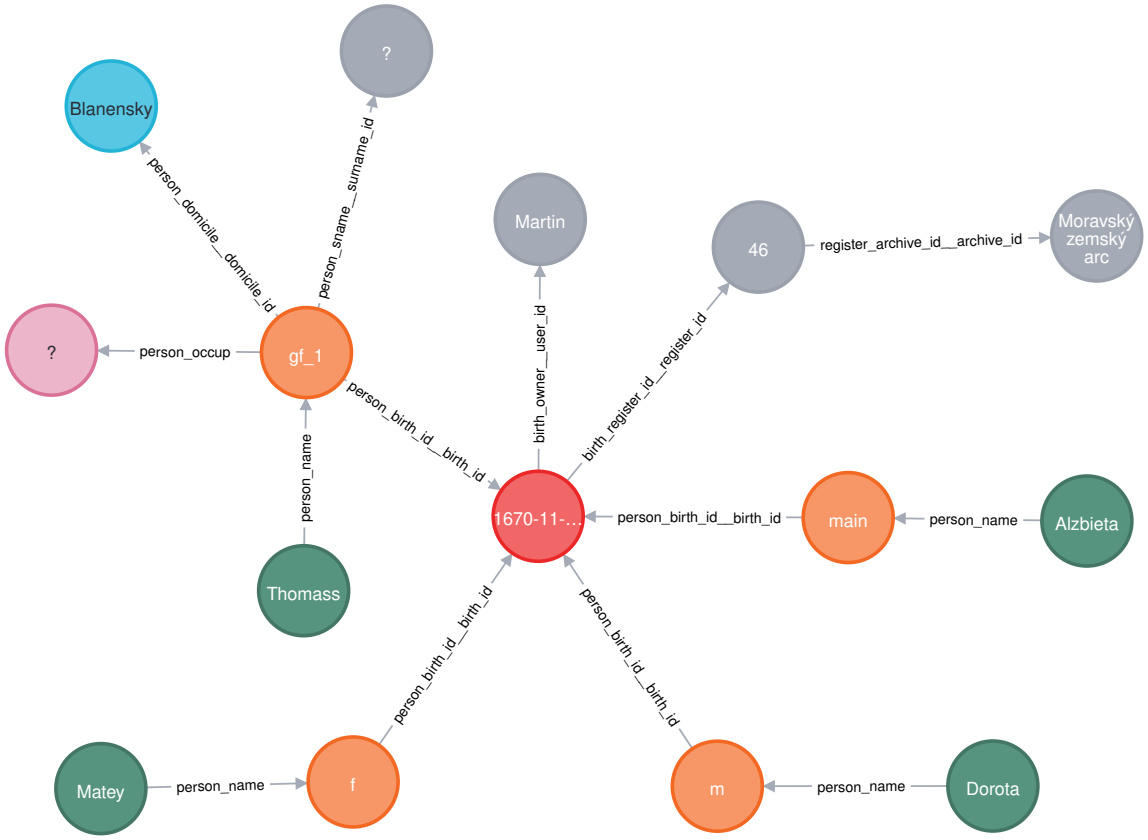
4.2.1 Datové schéma

Protože je Neo4j grafová databáze, neobsahuje tabulky. K rozlišení uzlů se používá vlastnost „label“. Všechna data přenesená z jedné tabulky proto budou mít stejný atribut „label“. Tento typ databáze dobře modeluje vztahy a jelikož relační databáze byla také založena na vztazích, nebudeme muset datové schéma výrazně měnit. Data budou přenesena z tabulky do uzlů se stejným názvem. Data z tabulek potřebných pro vytvoření vztahů typu „many-to-many“ v relační databázi však můžeme odstranit a např. pořadí jména z tabulky „personName“ přenést přímo do atributů vztahu.

4.2.2 Přenos dat

Pro automatizaci procesu přenosu dat byly definovány tabulky, které je potřeba přenést. Samotný přenos pro každou tabulku proházel takhle:

1. Proběhl dotaz pro získání názvů sloupců tabulky.
2. Dalším krokem bylo získání datových typů sloupců podle jejich názvů.
3. Bylo potřeba vybrat typy používané v Neo4j, které nahradí typy MySQL.
4. Dále proběhl dotaz pro získání dat z tabulky.



Obrázek 4.1: Příklad záznamu o narození.

5. Poté byla data shromážděna do jednoho velkého dotazu pro vytvoření uzlů.

Při práci s databází byla použita oficiální knihovna Neo4j pro Python. Pro snadnou práci s databází byla použita třída Neo4jConnection, kterou lze vidět na vypíšu 4.1. Třída pomáhá otevírat a zavírat připojení k databázi, pracovat s odpověďmi a zpracovávat chyby [13].

```
class Neo4jConnection:
    def __init__(self, uri, user, password):
        self.driver = GraphDatabase.driver(uri, auth=(user, password))

    def close(self):
        if self.driver is not None:
            self.driver.close()

    def query(self, query, db=None):
        assert self.driver is not None, "Driver not initialized!"
        session = None
        response = None
        try:
            session = self.driver.session(database=db) if db is
            not None else self.driver.session()
            response = list(session.run(query))
        except Exception as e:
```

```
            print("Query failed:", e)
        finally:
            if session is not None:
                session.close()
        return response
```

Výpis 4.1: Třída Neo4jConnection.

Během testování se ukázalo, že Neo4j má potíže s vytvářením velkého množství uzlů pomocí běžného dotazu CREATE (přenos databáze trval několik hodin). Při hledání řešení bylo zjištěno, že při načítání velkého množství dat funguje mnohem lépe načítání přes soubory CSV. Skript byl proto upraven. Data získaná z MySQL se nejprve uloží do souboru CSV ve složce, která se nachází v adresáři databáze pod názvem import. Načítání dat ze souborů CSV je možné provádět pouze z této složky.

Po vytvoření uzlů bylo nutné vytvořit mezi nimi vazby. K tomu byly pomocí SQL dotazu získány všechny cizí klíče a názvy tabulek (názvy tabulek musely být opraveny, protože jsou uloženy v malých písmenech), na které ukazují, a pomocí těchto dat byly vytvořeny vazby mezi uzly. Pro zajištění jedinečnosti názvu vazby byly názvy ve formátu název_tabulky1_název_klíče1___název_tabulky2_název_klíče2.

V tomto případě nebylo vytvoření většiny indexů nutné, protože na rozdíl od relačních databází jsou v grafové databázi uzly propojeny přímo. V budoucnu budou přidány pouze některé indexy pro optimalizaci dotazů do databáze.

4.2.3 Čištění databáze

Při psaní importů se vyskytly chyby, a proto bylo někdy nutné databázi vyčistit. Nejjednodušší způsob, jak to provést, je pomocí dotazu 4.2.

```
MATCH n
DETACH DELETE n
```

Výpis 4.2: Smazání všech dat z Neo4j.

Pomocí tohoto dotazu najdeme všechny uzly a odstraníme je spolu s vazbami. Pokud nepoužijeme klíčové slovo DETACH, nebudeme moci odstranit uzly, které mají vazby. Tato metoda je skvělá pro malé databáze, ale s větším množstvím dat vyžaduje velké množství paměti RAM, a proto byl pro vyčištění databáze napsán krátký python skript 4.3.

```
# Vytvorime pripojeni
conn = Neo4jConnection("bolt://localhost:7687", "login", "password")
# Ziskame vsechny mozne stitky
res = conn.query('CALL db.labels')
for i in res:
    # Odstranime uzly odpovidajici danym stitkum
    conn.query(f'''MATCH (n:{i[0]}) DETACH DELETE n''')
```

Výpis 4.3: Smazání všech dat z Neo4j pomocí Python skriptu.

4.2.4 Rozšíření možností dotazů

Pro rozšíření možností dotazů byla nainstalována knihovna APOC (Awesome Procedures On Cypher). Knihovna přidává některé funkce, které budeme v budoucnu používat v našich

dotazech. Tuto knihovnu není potřeba stahovat, je distribuována spolu s neo4j. Stačí ji pouze přesunout z adresáře $NEO4J_HOME/labs do adresáře $NEO4J_HOME/plugins.

4.2.5 Realizace dotazů

1. Rekonstrukce záznamu o narození

Celý první dotaz lze nalézt v příloze E, zde budou popsány jeho klíčové části. Najdeme záznam o narození:

```
MATCH (bth:birth{id:3677})
```

Výpis 4.4: Vyhledání uzlu podle štítku a atributu.

Pak pomocí OPTIONAL MATCH, který funguje podobně jako MATCH, ale s tím rozdílem, že pokud výsledek není nalezen, použije OPTIONAL MATCH pro chybějící části vzoru hodnotu null. OPTIONAL MATCH lze vnímat jako analogii OUTER JOIN v SQL. Dále se vyhledaly všechny potřebné uzly s ohledem na jejich vztahy. Nejprve bylo rozhodnuto hledat všechny lidi spojené s záznamem jednotlivě:

```
OPTIONAL MATCH (f:person{rel:'f'})-:person_birth_id__birth_id]-(bth:birth)
OPTIONAL MATCH (m:person{rel:'m'})-:person_birth_id__birth_id]-(bth:birth)
OPTIONAL MATCH (f_f:person{rel:'f_f'})-[:person_birth_id__birth_id]-(bth:birth)
OPTIONAL MATCH (f_m:person{rel:'f_m'})-[:person_birth_id__birth_id]-(bth:birth)
OPTIONAL MATCH (f_m_f:person{rel:'f_m_f'})-[:person_birth_id__birth_id]-(bth:birth
    )
OPTIONAL MATCH (f_m_m:person{rel:'f_m_m'})-[:person_birth_id__birth_id]-(bth:birth
    )
OPTIONAL MATCH (m_f:person{rel:'m_f'})-[:person_birth_id__birth_id]-(bth:birth)
OPTIONAL MATCH (m_m:person{rel:'m_m'})-[:person_birth_id__birth_id]-(bth:birth)
OPTIONAL MATCH (m_m_f:person{rel:'m_m_f'})-[:person_birth_id__birth_id]-(bth:birth
    )
OPTIONAL MATCH (m_m_m:person{rel:'m_m_m'})-[:person_birth_id__birth_id]-(bth:birth
    )
OPTIONAL MATCH (gf_1:person{rel:'gf_1'})-[:person_birth_id__birth_id]-(bth:birth)
```

Výpis 4.5: Hledání osob spojených s záznamem o narození(první varianta).

Ale tato varianta trvala příliš dlouho, a proto bylo rozhodnuto ji optimalizovat. Ve druhé variantě byly všechny osoby vyhledávány současně:

```
OPTIONAL MATCH (psn:person)-[:person_birth_id__birth_id]-(bth)
```

Výpis 4.6: Hledání osob spojených s záznamem o narození(druhá varianta).

Ve druhém případě všechny nalezené osoby budou nakonec shromážděny do jednoho seznamu a při dalším použití bude potřeba tento seznam zpracovat na straně dotazujícího.

Důležitou součástí požadavku je získání úplného jména a povolání.

```
apoc.text.join(COLLECT(DISTINCT psn_pn.name),' ') as psn_name,
apoc.text.join(COLLECT(DISTINCT psn_oc.name),';') as psn_occupation
```

Výpis 4.7: Získání úplného jména a povolání.

Zde shromažďujeme jména z uzlů pomocí apoc.text.join() z knihovny APOC, jejíž instalace byla popsána výše, protože to standardními možnostmi jazyka Cypher nelze provést. Funkce collect() vrátí jeden agregovaný seznam obsahující hodnoty vrácené výrazem. Je důležité si uvědomit, že v Cypheru se GROUP BY provádí implicitně všemi agregačními

funkcemi. V příkazu WITH/RETURN budou všechny sloupce, které nejsou součástí agregátu, klíčem GROUP BY.

Dále shromažďujeme informace o osobě a seskupujeme lidi příslušející k jednomu záznamu do seznamu:

```
COLLECT({rel: psn.rel, dead: psn.dead, title: psn.title, name: psn_name, surname:
    psn_sn.name, occupation:psn_occupation,
domicile:psn_dm.name, street: psn.street, descr_num: psn.descr_num, religion:psn.
    religion, birth_date:psn.birth_date,
waif:psn.waif, category: psn.category, person_relation: psn_pr.name})as person
```

Výpis 4.8: Seskupení informací o lidech.

Je také vhodné zmínit, že při prvním spuštění se složité dotazy provádějí mnohem déle než v následujících. Dotazy se totiž ukládají do mezipaměti. Při standardní konfiguraci mezipaměť obsahuje až 1000 jedinečných dotazů, při změně dat o 75% se dotaz z mezipaměti odstraní.

2. Matriky obsahující záznamy o narození

Prohledáváme uzly „register“ a „birth“, které jsou spojeny vztahem „birth_register_id ___register_id“, a vracíme potřebná data. Na rozdíl od SQL verze, neprovádíme „group by“. V tomto případě můžeme použít „RETURN DISTINCT“ k odstranění duplicit.

```
MATCH (r:register)-[:birth_register_id__register_id]-(b:birth)
WITH r.id as register_id, r.signature as signature
RETURN DISTINCT register_id, signature
```

Výpis 4.9: Matriky obsahující záznamy o narození.

3. Počet záznamů o narození v matrikách

Tento dotaz se příliš neliší od předchozího, ale tentokrát počítáme záznamy.

```
MATCH (b:birth)-[:birth_register_id__register_id]-(r:register)
RETURN COUNT(r.id) as nums, r.id as register_id
ORDER BY nums
```

Výpis 4.10: Počet záznamů o narození v matrikách.

4. Záznamy o narození v matrice

V tomto dotazu hledáme podle atributu „register_id“ uzlu „birth“, proto přidáme index pro tento atribut.

```
MATCH(birth:birth{register_id:10914})
RETURN birth.id, birth.scan, birth.pos, birth.lay
ORDER BY birth.scan, birth.pos;
```

Výpis 4.11: Záznamy o narození v matrice.

5. Četnost výskytu příjmení

```
MATCH (p:person)-[:person_sname__surname_id]-(s:surname)
RETURN COUNT(*) as nums, p.sname, s.name
ORDER BY nums
```

Výpis 4.12: Četnost výskytu příjmení v záznamech.

6. Vyhledávání záznamů podle ID příjmení

V tomto dotazu hledáme záznamy podle ID příjmení, proto vytvoříme index pro tento atribut. Nalezneme příjmení podle jeho ID a poté nalezneme všechny potřebné uzly a rekonstruujeme plné jméno a povolání jako v prvním dotazu.

```
MATCH (sn:surname{id:34435})-[:person_sname__surname_id]-(psn:person)-[:
    person_birth_id__birth_id]-(bth:birth)
MATCH (bth)-[:birth_register_id__register_id]-(reg:register)
OPTIONAL MATCH (reg)-[:register_archive_id__archive_id]-(ar:archive)
OPTIONAL MATCH (reg)-[:register_fond_id__archiveFond_id]-(af:archiveFond)
OPTIONAL MATCH (psn)-[:person_domicile__domicile_id]-(dom:domicile)
OPTIONAL MATCH (psn)-[:person_person_relation__personRelation_id]-(pr:
    personRelation)
OPTIONAL MATCH (psn)-[:person_occup]-(occ:occupation)
OPTIONAL MATCH (psn)-[:person_name]-(name:name)
RETURN DISTINCT ar.short as archive, af.name as fond, reg.signature, bth.id, bth.
    scan, bth.pos, psn.rel,
       psn.title, apoc.text.join(COLLECT(DISTINCT name.name),' ') as FullName, sn.
    name as snameFull,
       dom.name as domicileFull, psn.street, psn.descr_num, psn.religion, psn.
    birth_date,
       psn.dead, psn.waif, psn.category, apoc.text.join(COLLECT(DISTINCT occ.name)
    ,';') as occupation,
       pr.name as personRelationFull, psn.widow, psn.legitimate, psn.dead_date,
    psn.work_place, psn.age
```

Výpis 4.13: Vyhledávání záznamů podle ID příjmení.

7. Vyhledávání záznamů podle příjmení

Tento je podobný šestému, pouze zde se vyhledává podle příjmení místo jeho ID, a pro tento atribut je také potřeba přidat index.

8. Četnost výskytu různých forem normalizovaných jmen

```
MATCH (n:name)-[:name_norm_name_id__normalizedName_id]-(nn:normalizedName)
RETURN COUNT(*) as nums, nn.name
```

Výpis 4.14: Četnost výskytu různých forem normalizovaných jmen.

9. Vyhledávání záznamů podle normalizovaného jména

Stejně jako v případě MySQL je tento dotaz podobný šestému, ale vyhledává se podle normalizované verze jména.

4.3 MongoDB

4.3.1 Datové schéma

V MongoDB lze „ekvivalentem“ tabulek nazvat kolekce. Data ze stejné tabulky se budou zapisovat do kolekce se stejným názvem. Nejprve bylo rozhodnuto přenést data tak, jak jsou, a poté na základě dotazů vytvořit potřebnou strukturu dat, protože v MongoDB je považováno za přípustné denormalizovat data.

4.3.2 Přenos dat

Při přenosu dat do MongoDB je třeba mít na paměti, že pokud existují čísla s desetinnou čárkou, je nutné je převést na typ Decimal128. Pro přenos dat z MySQL byla data z tabulek dotazována a formována do dokumentů a nahrána do MongoDB. Na rozdíl od Neo4j nebyly pozorovány žádné problémy s dobou načítání dat přímo.

Bohužel první dotaz ukázal, že MongoDB si špatně poradí s velkým počtem vazeb, a proto bylo rozhodnuto uchýlit se k denormalizaci dat a změně struktury, která by pomohla zbavit se nadměrného vyhledávání dat v jiných kolekcích.

Vzhledem k tomu, že každý záznam není v relaci s ostatními, např. můžeme mít záznam o narození a smrti téhož člověka, ale pro každý záznam bude vytvořen svůj vlastní záznam „person“, který reprezentuje osobu v daném záznamu, můžeme všechna data spojit do jednoho dokumentu. S tím nám dobře pomáhá databáze Neo4j, která umožňuje pohodlně získávat potřebná data ve vhodné formě pro přenos do MongoDB.

```
MATCH(bth:birth)
OPTIONAL MATCH (bth)--(birth_confirm_surn:surname)
OPTIONAL MATCH (bth)--(bm:birthMarriage)
OPTIONAL MATCH (bm)--(bm_surname:surname)
OPTIONAL MATCH (bm)-[bmn:birthMarriage_name]-(bm_name:name)
OPTIONAL MATCH (bm)--(bm_occ:occupation)
OPTIONAL MATCH (bm)--(bm_dom:domicile)
WITH bth, birth_confirm_surn, bm_surname, bm as birth_m, COLLECT({name_order:bmn.
    name_order, name:bm_name}) as names,
COLLECT(bm_occ) as occupations, bm_dom
WITH bth, birth_confirm_surn, COLLECT({birth_marriage:birth_m, names:names,
    occupations:occupations, surname: bm_surname, domicilie:bm_dom}) as
    birth_marriage
OPTIONAL MATCH (bth)-[bcn:birth_confirm_name]-(bc_name:name)
WITH bth, birth_confirm_surn, birth_marriage, COLLECT({name_order:bcn.name_order,
    name:bc_name}) as birth_confirm_names
OPTIONAL MATCH (bth)--(reg:register)
OPTIONAL MATCH (bth)--(prs:person)
OPTIONAL MATCH (prs)-[pnn:person_name]-(nm:name)
OPtIONAL MATCH (surn:surname)--(prs)
OPTIONAL MATCH (po:occupation)--(prs)
OPTIONAL MATCH (pr:personRelation)--(prs)
OPTIONAL MATCH (dm:domicile)--(prs)
WITH bth,birth_confirm_surn, birth_marriage, prs, surn, pr, dm, reg, COLLECT(
    DISTINCT {name_order:pnn.name_order, name:nm}) as names,
COLLECT(po) as occupations, birth_confirm_names
WITH bth, birth_confirm_surn, birth_marriage, reg, COLLECT(DISTINCT {person:prs,
    surname:surn, names:names,
```

```
occupations:occupations, person_relation:pr, domicile:dm}) as persons,
    birth_confirm_names
WITH bth as birth, birth_confirm_surn, birth_marriage, reg, persons,
    birth_confirm_names
return DISTINCT *
```

Výpis 4.15: Rekonstrukce záznamu pro přenos do MongoDB.

Zde hledáme potřebná data týkající se narození. Pro data jako informace o osobě probíhá jejich seskupení do seznamu.

Následně je třeba odstranit prázdné seznamy a hodnoty null z každého záznamu. K tomu byla napsána funkce, která rekurzivně procházela všechna data a v případě nalezení takových hodnot je odstraňovala.

```
    def fixNulls(data):
    if issubclass(type(data), Iterable):
        delete=[]
        for k,v in enumerate(data):
            if isinstance(data, list) and v == None:
                delete.append(k)
            elif isinstance(data, dict) and data[v] ==None:
                delete.append(v)
            elif type(data) != str and issubclass(type(data[k if isinstance(data,
                list) else v]), Iterable):
                fixNulls(data[k if isinstance(data, list) else v])
                if len (data[k if isinstance(data, list) else v]) == 0:
                    delete.append( k if isinstance(data, list) else v)
        if type(data) == list:
            delete.sort(reverse=True)
        for i in delete:
            del data[i]
```

Výpis 4.16: Úprava dat získaných z Neo4j.

Po provedených manipulacích získáme kolekce birth_record, marriage_record a burial_record obsahující odpovídající záznamy. Příklad záznamu birth_record můžeme vidět v příloze H.

4.3.3 Realizace dotazů

1. Rekonstrukce záznamu o narození

V tomto případě není potřeba hledat různé části záznamu, jsou již všechny spojeny, takže jednoduše vyhledáme záznam podle jeho ID:

```
{
    $match:{_id: 5319}
},
```

Výpis 4.17: Hledáme záznam podle ID

A také najdeme archiv:

```
  {
    $lookup: {
      from: "archive",
      localField: "reg.archive_id",
```

```
      foreignField: "_id",
      as: "archive"
    }
  },
  {
    $unwind: {path: "$archive", preserveNullAndEmptyArrays: true}
  },
```

Výpis 4.18: Vyhledání archivu.

Pak hledáme fond:

```
  {
    $lookup: {
      from: "archiveFond",
      localField: "reg.fond_id",
      foreignField: "_id",
      as: "archiveFond"
    }
  },
  {
    $unwind: {
      path: "$archiveFond",
      preserveNullAndEmptyArrays: true
    }
  }
```

Výpis 4.19: Vyhledání fondu.

Tato varianta je velmi rychlá, protože máme všechna data již na jednom místě.

2. Matriky obsahující záznamy o narození

V tomto dotazu potřebujeme pouze seskupit data podle ID matriky a čísla knihy a provést projekci požadovaných údajů.

```
db.getCollection('birth_record').aggregate(
  [
    {
      $group: {_id: {id: '$reg.id', signature: '$reg.signature'}}
    },
    {
      $project: {_id: 0, id: '$_id.id', signature: '$_id.signature'}
    }
  ],
);
```

Výpis 4.20: Matriky obsahující záznamy o narození.

3. Počet záznamů o narození v matrikách

Je třeba seskupit záznamy podle ID matriky a spočítat jejich počet:

```
db.getCollection('birth_record').aggregate(
  [{$group: {_id: '$birth.register_id', count: { $count: {} }}}]);
```

Výpis 4.21: Počet záznamů o narození v matrikách.

4. Záznamy o narození v matrice

Prostě hledáme záznamy, jejichž ID matriky rovna se zadanému.

```
db.getCollection('birth_record').aggregate([{ $match: {'reg.id': 10914}}]);
```

Výpis 4.22: Záznamy o narození v matrice.

5. Četnost výskytu příjmení

Jelikož jsou příjmení v tomto schématu uložena v různých kolekcích, je třeba je spojit pomocí $unionWith.

```
db.getCollection('birth_record').aggregate(
[
  { $unionWith: { coll: "marriage_record"}},
  { $unionWith: { coll: "burial_record"}},
  { $unwind: "$persons" },
  { $match: { "persons.person.sname": { $exists: true } } },
  { $project: {_id: 0, name: "$persons.surname.name", id: '$persons.surname.id'}},
  { $group: { _id: {name:"$name", id:"$id"}, count: { $sum: 1 }}},
]
);
```

Výpis 4.23: Četnost výskytu příjmení.

6. Vyhledávání záznamů podle ID příjmení

```
db.getCollection('birth_record').aggregate(
  [
    { $match: { 'persons.person.sname': 34435 } },
    { $unwind: '$persons' },
    { $match: { 'persons.person.sname': 34435 } },
    {
      $project: {
        id: '$birth.id',
        sname: '$persons.surname.id',
        surname: '$persons.surname.name',
        rel: '$persons.person.rel'
      }
    },
    {
      $lookup: {
        from: "archive",
        localField: "reg.archive_id",
        foreignField: "_id",
        as: "archive",
    },
    },
    {$unwind:{path: "$archive", preserveNullAndEmptyArrays: true,},},
    {
      $lookup: {
        from: "archiveFond",
        localField: "reg.fond_id",
        foreignField: "_id",
```

```
        as: "archiveFond",
    },
    },
    {$unwind:{path: "$archiveFond", preserveNullAndEmptyArrays: true,},
    },
  ],
);
```

Výpis 4.24: Vyhledávání záznamů podle ID příjmení.

7. Vyhledávání záznamů podle příjmení

Stejně jako v jiných DBMS tento dotaz od šestého se liší pouze hledáním podle příjmení.

8. Četnost výskytu různých forem normalizovaných jmen

```
db.getCollection('birth_record').aggregate(
  [
    { $unionWith: { coll: "marriage_record"}},
    { $unionWith: { coll: "burial_record"}},
    {
      $match: {
        'persons.names': {$elemMatch: {'name.norm_name_id': { $ne: null }}}
      }
    },
    {
      $unwind: {path: '$persons', preserveNullAndEmptyArrays: true}
    },
    {
      $unwind: {path: '$persons.names',preserveNullAndEmptyArrays: true}
    },
    {
      $match: {'persons.names.name.norm_name_id': {$ne: null}}
    },
    {
      $project: {
        n_n_id:'$persons.names.name.norm_name_id',
        n_id: '$persons.names.name.id'
      }
    },
    {
      $group: {_id: { n_n_id: '$n_n_id', n_id: '$n_id' }}
    },
    {
      $group: {_id: { n_n_id: '$_id.n_n_id' },cnt: { $count: {} }}
    },
    { $sort: { '_id.n_n_id': 1 } }
  ]
);
```

Výpis 4.25: Vyhledávání záznamů podle příjmení.

9. Vyhledávání záznamů podle normalizovaného jména

V tomto dotazu začínáme hledání v kolekci normalizedName, protože nám to umožní zpracovat méně dokumentů.

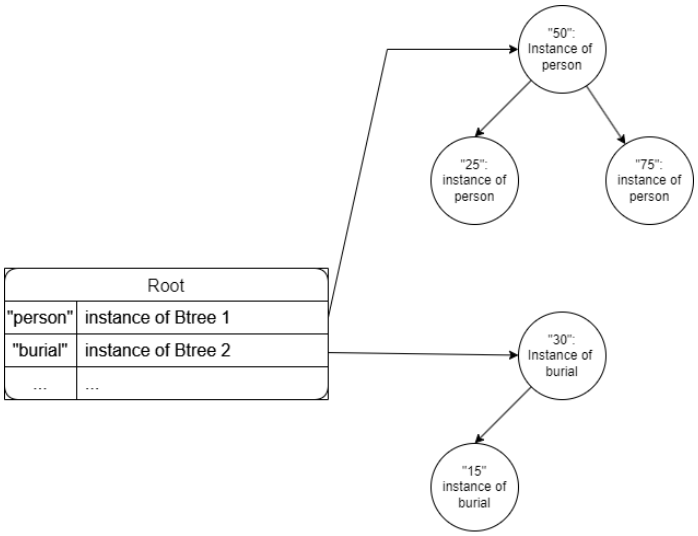
```
db.getCollection('normalizedName').aggregate(
  [
    { $match: {name: 'František'}},
    {
      $lookup: {
        from: 'birth_record',
        localField: '_id',
        foreignField:
          'persons.names.name.norm_name_id',
        as: 'birth'
      }
    },
    { $unwind: {path: '$birth', preserveNullAndEmptyArrays: true} },
    { $unwind: {path: '$birth.persons', preserveNullAndEmptyArrays: true} },
    { $match: {'birth.persons.names': {$ne: null}} },
    {
      $match: {
        $expr: {
          $gt: [
            {
              $size: {
                $filter: {
                  input: '$birth.persons.names',
                  as: 'name',
                  cond: {
                    $eq: ['$$name.name.norm_name_id','$_id']}}}},
               0]}}}
    {
      $lookup: {
        from: 'archive',
        localField: 'birth.reg.archive_id',
        foreignField: '_id',
        as: 'archive'
      }
    },
    {$unwind:{path: '$archive', preserveNullAndEmptyArrays: true}},
    {
      $lookup: {
        from: 'archiveFond',
        localField: 'birth.reg.fond_id',
        foreignField: '_id',
        as: 'archiveFond'
      }
    },
    {$unwind:{path: '$archiveFond', preserveNullAndEmptyArrays: true}
    }]);
```

Výpis 4.26: Vyhledávání záznamů podle normalizovaného jména.

4.4 ZODB

4.4.1 Datové schéma

V této databázi neexistují žádné ekvivalenty tabulek. Data lze vkládat přímo do kořene, ale to by bylo velmi nepraktické. Proto pro každou tabulku vytvoříme B-Strom, který vložíme do kořene pod klíč odpovídající názvu tabulky. Tento strom naplníme daty z odpovídající tabulky. Klíči záznamů budou ID záznamu, pokud záznam žádné nemá, použijeme pořadové číslo záznamu při zpracování.



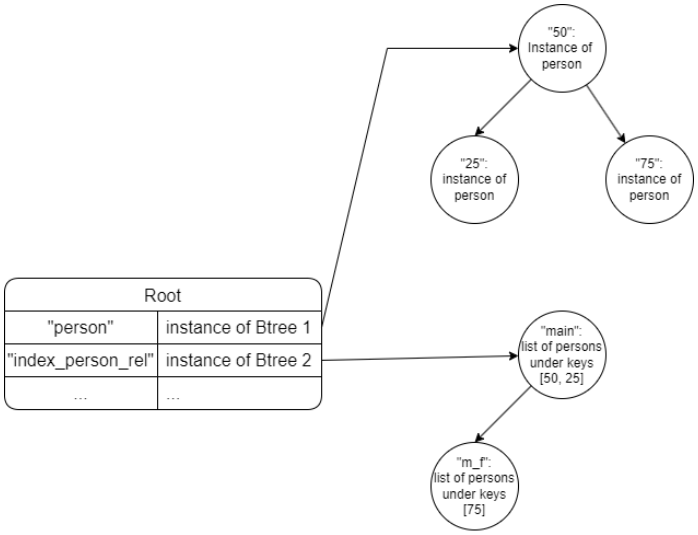
Obrázek 4.2: Příklad uložení dat.

4.4.2 Přenos dat

Nejprve musíme vytvořit samotné třídy. Třídy by bylo možné generovat přímo během běhu skriptu, ale pro další práci je budeme potřebovat, protože při práci s touto databází je nutné, aby daná třída byla importována do našeho skriptu. Je třeba pamatovat na to, že třída musí být dědicem třídy Persistent. Za tímto účelem vytvoříme jednoduchý skript, který vyžádá názvy všech polí pro danou tabulku a vytvoří třídu se stejným názvem jako tabulka. Poté musíme vytvořit skript pro přenos dat z MySQL do ZODB.

Dále musíme propojit všechna data v databázi mezi sebou. Musíme projít všechny záznamy a cizí klíče nahradit odkazy na potřebné objekty. Také pro optimalizaci vyhledávání budeme muset vytvořit indexy, protože jinak, pokud bychom chtěli najít něco jiného než podle klíče, museli bychom projít každý záznam tabulky. Index bude vypadat jako strom se seznamy objektů, které mají stejnou hodnotu indexovaného atributu. Každý záznam bude mít klíč odpovídající hodnotě indexovaného pole.

Tato diplomová práce je zaměřena především na testování určitých často používaných dotazů, a proto není v našem případě důležité udržovat aktuálnost indexu, protože dotazy na zápis neprovádíme. Je však třeba říci, že udržováním tohoto indexu v aktuálním stavu



Obrázek 4.3: Příklad indexu.

by se měl zabývat programátor. Je nutné, aby při aktualizaci dat pole došlo i k aktualizaci indexu. Poté můžeme přistoupit k testování rychlosti provádění dotazů.

4.4.3 Realizace dotazů

1. Rekonstrukce záznamu o narození

Nejprve potřebujeme najít záznam o narození. Pro tento účel potřebujeme zjistit, zda existuje takový záznam, a pokud ano, najít ho.

```
if not root['birth'].has_key(recid):
    return None
birth: dbclasses.birth = root['birth'][recid]
```

Výpis 4.27: Kontrola existence záznamu.

Při přístupu k jiným objektům je třeba ověřit, zda se nesměřujeme na hodnotu None. Například při hledání tvůrce záznamu:

```
owner_name = None if birth.owner == None else birth.owner.name
```

Výpis 4.28: Kontrola na hodnotu None.

Také důležitým je hledání jména:

```
def GetName(key, index:str, root):
    names = []
    fullname = None
    if root[index].has_key(key):
        for i in root[index][key]:
            names.append(i.name_id.name)
        fullname = ' '.join(names)
```

```
8       return fullname if fullname else None
```

Výpis 4.29: Hledání jména

Zde hledáme, zda je v indexu pod daným klíčem jméno. Pokud ano, všechna jména shromažďujeme do seznamu a provádíme konkatenaci jmen.

2. Matriky obsahující záznamy o narození

Zde projdeme prvky indexu index_birth_register_id a najdeme potřebné informace o matrice.

```
1  def GetRegisterWithBirthRecords(root):
2      records = root['index_birth_register_id'].values()
3      result = [{'register':rec[0].register_id.id, 'signature':rec[0].register_id.
           signature} for rec in records]
4      return result
```

Výpis 4.30: Matriky obsahující záznamy o narození.

3. Počet záznamů o narození v matrikách

Jelikož jsou záznamy v indexu uloženy pod jedním klíčem jako seznam, můžeme pro jejich počet použít funkci len().

```
1  def BirthRecCountByRegister(root):
2      records = root['index_birth_register_id'].values()
3      result = [{'register':rec[0].register_id.id, 'count':len(rec)} for rec in
           records]
4      return result
```

Výpis 4.31: Počet záznamů o narození v matrikách.

4. Záznamy o narození v matrice

```
1  def BirthRecInRegister(register_id,root):
2      if root['index_birth_register_id'].has_key(register_id):
3          records = root['index_birth_register_id'][register_id]
4          result = [{'short': rec.register_id.archive_id.short, 'fond': rec.
               register_id.fond_id.name,
5                     'signature': rec.register_id.signature,'id':rec.id, 'scan':rec.
                          scan,
6                     'pos': rec.pos, 'lay': rec.lay} for rec in records]
7          result.sort(key=lambda x: (x['scan'], x['pos']))
8          return result
9      return []
```

Výpis 4.32: Záznamy o narození v matrice.

5. Četnost výskytu příjmení

```
def SurnameCountInRecords(root):
    res = [{'count':len(j), 'sname':j[0].sname.id, 'surname':j[0].sname.name} for
        j in root['index_person_sname'].values() if j[0].sname is not None]
    res.sort(key= lambda x: x['count'])
    return res
```

Výpis 4.33: Četnost výskytu příjmení.

6. Vyhledávání záznamů podle ID příjmení

```
def BirthRecordBySnameId(id, root):
    if root['index_person_sname'].has_key(id):
        records = root['index_person_sname'][id]
        return [{'short': None if rec.birth_id.register_id.archive_id is None else
            rec.birth_id.register_id.archive_id.short, 'fond': None if rec.birth_id
            .register_id.fond_id is None else rec.birth_id.register_id.fond_id.name
            , 'signature': rec.birth_id.register_id.signature, 'id':rec.birth_id.id
            , 'scan':rec.birth_id.scan, 'pos': rec.birth_id.pos, 'rel':rec.rel, '
            title': rec.title, 'FullName': GetName(rec.id, '
            index_person_name_person_id', root), 'surname': rec.sname.name, '
            domicile': rec.domicile,'street': rec.street, 'descr_num': rec.
            descr_num, "religion": rec.religion, 'birth_date': rec.birth_date, '
            dead': rec.dead, 'waif': rec.waif, 'category': rec.category, 'occups':
            GetOccupation(rec.id, 'index_person_occup_person_id', root), '
            person_relation': None if rec.person_relation is None else rec.
            person_relation.name, 'widow': rec.widow, 'legitimate': rec.legitimate,
             'dead_date': rec.dead_date, 'work_place': rec.work_place, 'age': rec.
            age}
            for rec in records if rec.birth_id is not None]
    return []
```

Výpis 4.34: Vyhledávání záznamů podle ID příjmení.

7. Vyhledávání záznamů podle příjmení

Tento dotaz najde příjmení spojená s normalizovanou variantou a pro každé zavolá předchozí dotaz.

```
def BirthRecordBySurname(surname: str, root):
    if root['index_surname_name'].has_key(surname):
        result = []
        for i in root['index_surname_name'][surname]:
            result.extend(BirthRecordBySnameId(i.id, root))
        return result
    return []
```

Výpis 4.35: Vyhledávání záznamů podle příjmení.

8. Četnost výskytu různých forem normalizovaných jmen

```
def CountNames(root):
    return [{'count': len(i), 'name': i[0].norm_name_id.name} for i in root['
        index_name_norm_name_id'].values() if i[0].norm_name_id is not None]
```

Výpis 4.36: Četnost výskytu různých forem normalizovaných jmen.

9. Vyhledávání záznamů podle normalizovaného jména

V tomto dotazu (výpis 4.37) využijeme index pro normalizované jméno, pomocí kterého najdeme všechna jména spojená s normalizovanou variantou. Poté, když máme jméno, můžeme využít index pro třídu person_name. Nyní máme přístup ke všem potřebným informacím, zbývá je seskupit a vrátit uživateli.

```
def RecWithName(name, root):
    result = []
    if root['index_normalizedName_name'].has_key(name):
        for i in root['index_normalizedName_name'][name]:
            for j in root['index_name_norm_name_id'][i.id]:
                for n in root['index_person_name_name_id'][j.id]:
                    if n.person_id.birth_id is not None:
                        result.append({'short': None if n.person_id.birth_id.
                            register_id.archive_id is None else n.person_id.birth_id
                            .register_id.archive_id.short,
                        'fond': None if n.person_id.birth_id.register_id.fond_id is
                            None else n.person_id.birth_id.register_id.fond_id.name,
                        'signature': n.person_id.birth_id.register_id.signature, 'id
                            ':n.person_id.birth_id.id, 'scan':n.person_id.birth_id.
                            scan,
                        'pos': n.person_id.birth_id.pos, 'rel':n.person_id.rel, '
                            title': n.person_id.title, 'name': n.name_id.name,
                        'norm_name': n.name_id.norm_name_id.name, 'surname': None if
                             n.person_id.sname is None else n.person_id.sname.name,
                        'domicile': None if n.person_id.domicile is None else n.
                            person_id.domicile.name, 'street': n.person_id.street, '
                            descr_num': n.person_id.descr_num, "religion": n.
                            person_id.religion,
                        'birth_date': n.person_id.birth_date, 'dead': n.person_id.
                            dead, 'waif': n.person_id.waif, 'category': n.person_id.
                            category,
                        'occups': GetOccupation(n.person_id.id, '
                            index_person_occup_person_id', root),
                        'person_relation': None if n.person_id.person_relation is
                            None else n.person_id.person_relation.name, 'widow': n.
                            person_id.widow,
                        'legitimate': n.person_id.legitimate, 'dead_date': n.
                            person_id.dead_date, 'work_place': n.person_id.
                            work_place, 'age': n.person_id.age})
    return result
```

Výpis 4.37: Vyhledávání záznamů podle normalizovaného jména.

Kapitola 5

Zhodnocení získaných informací

V této kapitole porovnáme databáze na základě informací získaných v předchozích částech.

	MySQL	Neo4j	MongoDB	ZODB
Podporované platformy	Windows/linux	Windows/linux	Windows/linux	Windows/linux
Podporované jazyky	Většina populárních	Většina populárních	Většina populárních	Python
Licence	GPLv2	GPLv3	GNU AGPL v3.0	ZPL
Prostorová složitost	202 mb	372 mb	50,5 mb	1,1 gb
Jednoduchost používání	2	1	3	4

Tabulka 5.1: Porovnávaní DBMS

5.1 Základní informace

Jak vidíme, všechny databáze splňují naše požadavky na podporované platformy, a licence jejich bezplatných verzí nám vyhovuje. Všechny zkoumané DBMS s výjimkou ZODB mají ovladače pro většinu populárních programovacích jazyků.

5.2 Prostorová složitost

Z hlediska prostorové složitosti je MongoDB výrazně před ostatními, zatímco ZODB zabírá podstatně více místa než ostatní DBMS, což souvisí s tím, že uchovává serializované objekty Pythonu. V MongoDB lze prostorovou složitost vyhodnotit pomocí dotazu 5.1.

```
1 db.stats()
```

Výpis 5.1: prostorová složitost MongoDB

V případě MySQL lze tento kritérium také posoudit pomocí dotazu 5.2.

```
1 SELECT table_schema "DB Name",
2        SUM(data_length + index_length)
3 FROM information_schema.tables
4 GROUP BY table_schema;
```

Výpis 5.2: prostorová složitost MySQL

Pro Neo4j a ZODB se podíváme na velikost databázových souborů, protože první umožňuje zjistit velikost databáze pouze v Enterprise Edition a druhá tuto funkcionalitu vůbec nemá.

5.3 Omezení

MySQL

Hlavním omezením je limit počtu sloupců v tabulce - 4096, a maximální velikost řádku je 65 535 bajtů. Jinak jsme omezeni zdroji našeho zařízení.

Neo4j

Community verze Neo4j je omezena, např. limit počtu uzlů je 34 miliard. Existují také omezení funkcí, např. nelze provádět zálohy, ale je možné vytvářet dumpy.

MongoDB

Velikost dokumentu je omezena na 16 megabajtů, podporuje se ne více než 100 úrovní vnoření dokumentů. Kolekce může mít ne více než 64 indexů.

ZODB

Nepodařilo se najít žádné informace o omezeních kromě toho, že velikost databáze je omezena zdroji systému.

5.4 Jednoduchost používání

Nejjednodušší k používání databáze bude označena číslem 1 a nejméně pohodlná číslem 4.

Neo4j

Jako nejpohodlnější byla vybrána Neo4j, protože psaní a čtení dotazů zde je intuitivně srozumitelné, dotazy jsou relativně krátké a databáze má dobře pochopitelný profiler dotazů. Schéma databáze je také jednoduché a zbavuje nás nadbytečných entit, jako jsou tabulky pro vytváření vztahů many-to-many.

MySQL

Na druhém místě jsem umístil MySQL, protože její schéma je sice složitější než u Neo4j, ale stále poměrně pohodlné, a navíc má databáze srozumitelný profiler dotazů.

MongoDB

Poté následuje MongoDB, protože např. pro mě je méně pohodlné pracovat s JSON než s datovým modelem v předchozích databázích, a také není příliš pohodlné psát a číst dotazy. Další nevýhodou je, že pokud uděláte chybu v dotazu a zadáte nesprávné nebo chybějící pole, nebudete informováni o chybě. Další nevýhodou je složitost práce s indexy.

ZODB

Na posledním místě je ZODB, protože téměř vše, co chceme dělat, musíme implementovat ručně, např. ručně vytvářet a udržovat indexy.

5.5 Rychlost provádění dotazů

Testování rychlosti provádění dotazů se provádí na straně klienta. Pro přesnější výsledek se dotaz spustí několikrát a výsledek se průměruje. Pro Neo4j a ZODB je třeba mít na paměti, že první spuštění dotazu bude trvat déle než následující. V prvním případě je to způsobeno tím, že plán provádění dotazu bude později uložen do mezipaměti a dotaz bude vykonán rychleji. Ve druhém případě je to způsobeno tím, že klient ukládá objekty do mezipaměti, což umožňuje urychlit provádění dotazů. Cachovaná data jsou uložena na disku v cache souboru, při připojení je možné konfigurovat velikost souboru.

Bylo zjištěno, že u některých dotazů v Neo4j se doba vykonání pomocí ovladače pro Python a přes Neo4j Browser liší několikanásobně, proto bylo rozhodnuto provádět dotazy prostřednictvím HTTP API Neo4j.

Výsledky našich měření(tabulka 5.2) nám umožňují učinit několik závěrů:

1. Rychlost vyhledávání je vysoká, pokud jsou data uložena v mezipaměti klienta, ale pokud data v mezipaměti chybí, pak je vyhledávání velmi pomalé. Hlavním problémem je to, že u velkých databází je drahé udržovat část databáze na straně klienta.

2. Neo4j dobře si poradí s rekonstrukcí záznamů (1. dotaz), ale v ostatních případech se většina dotazů provádí pomaleji než v MySQL.

3. MongoDB si dobré poradila v případech, kdy lze využít index a jsme již předem shromáždili veškerá potřebná data do jednoho dokumentu(dotazy č. 1 a 9), ale v našem případě to nebylo vždy možné, a proto některé dotazy probíhaly pomalu.

	MySQL	Neo4j	MongoDB	ZODB	ZODB(prázdná mezipaměť)
1. dotaz	211.83	5.98	1.02	0.63	48.14
2. dotaz	16.89	65.36	43.27	3.81	308.34
3. dotaz	17.81	44.85	40.46	0.16	1.0
4. dotaz	25.41	169.17	240.76	36.68	2492.39
5. dotaz	355.77	635.74	674.81	426.99	34082.01
6. dotaz	6.33	14.22	63.23	10.29	569.56
7. dotaz	6.20	14.43	63.27	2.61	19.99
8. dotaz	0.55	3.37	206.93	0.53	53.40
9. dotaz	190.35	125.23	35.41	103.07	6708.18

Tabulka 5.2: Rychlost provádění dotazů (ms).

Kapitola 6

Závěr

Cílem této diplomové práce bylo vybrat nejvhodnější databázi pro ukládání genealogických dat. Tento cíl byl dosažen výběrem předpokládaně vhodných variant a jejich porovnáním mezi sebou a s původní databází podle zadaných kritérií. Pro úspěch v tomto úkolu bylo důležité pochopit, jak vypadají vstupní data a jak jsou uložena v původní databázi. Poté bylo provedeno zkoumání existujících typů NoSQL databází, jejich výhod a nevýhod, nejpopulárnějších variant a také jejich licenčních omezení. Následně bylo nutné přenést data z původní databáze do vybraných databází pro srovnání, v případě potřeby změnit jejich strukturu a přidat indexy. V grafové a objektově orientované databázi bylo také nutné propojit data mezi sebou. Poté, co byly databáze připraveny k použití, bylo nutné adaptovat často používané SQL dotazy pro databáze, které porovnáváme. V tomto procesu hrála důležitou roli analýza dotazů, která pomáhá odhalit úzká místa našich dotazů a optimalizovat je. Po optimalizaci dat a dotazů bylo nutné porovnat dvě věci: jejich správnost a rychlost provádění. Za tímto účelem bylo nutné vytvořit skript, který by volal potřebné dotazy a měřil rychlost jejich provádění.

Výsledkem testování databází jsme získali rychlost provádění potřebných dotazů, což je jedno z kritérií pro hodnocení vybraných DBMS. Na základě těchto výsledků bude vybrána nejvhodnější databáze, což je úkolem této práce. Z výsledků testování vyplynulo, že nejrychlejší DBMS pro naše účely je ZODB v případě, že máme data v mezipaměti na straně klienta. Po porovnání databází podle zadaných kritérií lze za nejvhodnější DBMS pro naše účely označit MySQL. Pomocí ZODB bychom mohli výrazně zrychlit některé dotazy, ale v takovém případě jsme omezeni použitím jazyka Python a také se setkáváme s problémy jako je objem databáze, potřeba ukládat data do mezipaměti na straně klienta a také s obtížností práce s touto databází.

V budoucnu by bylo možné tuto práci vylepšit zvážením DBMS ArcadeDB. Jedná se o multimodelární DBMS, která podporuje dokumentově orientovaný model, ve kterém jsou podporovány odkazy na jiné dokumenty. To by mohlo při zachování výhod MongoDB eliminovat její nevýhodu v rychlosti provádění dotazů při spojování dokumentů pomocí $lookup.

Literatura

[1] *Bakov nad Jizerou-čs 03* [online]. Státní oblastní archiv v Praze [cit. 2024-04-08]. Dostupné z: https://ebadatelna.soapraha.cz/d/14706/2.

[2] *Create a graph database in Neo4j using Python* [online]. Zope Foundation, 2021 [cit. 2024-04-08]. Dostupné z: https://zodb.org/en/latest/tutorial.html.

[3] Alzahrani, H. Evolution of Object-Oriented Database Systems. *Global Journal of Computer Science and Technology*. Červenec 2016, s. 37–40, [cit. 2024-04-08]. Dostupné z: https://gjcst.com/index.php/gjcst/article/view/778.

[4] Bryce Merkl Sasaki, J. C. . R. H. *Graph Databases for Beginners* [online]. Neo4j, 2018 [cit. 2024-04-08].

[5] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A. et al. Bigtable: A Distributed Storage System for Structured Data. *ACM Trans. Comput. Syst.* New York, NY, USA: Association for Computing Machinery. jun 2008, sv. 26, č. 2. DOI: 10.1145/1365815.1365816. ISSN 0734-2071. Dostupné z: https://doi.org/10.1145/1365815.1365816.

[6] KOTYLOVÁ, A. *Genealogie jako kulturní fenomén [online]*. 2014. [cit. 2024-04-08]. Diplomová práce. Západočeská univerzita v Plzni, Fakulta filozofickáPlzeň. SUPERVISOR: Mgr. Lenka Jakoubková Budilová, Ph.D. Dostupné z: https://theses.cz/id/jl68qa/.

[7] Kočí, R., Rozman, J. a Zbořil, F. Database Concept for Transcription of Registry Records into Digital Form. In: *Proceedings of the 3rd International Conference on Software Engineering and Information Management - ICSIM'20*. Association for Computing Machinery, 2020, s. 21–25. DOI: 10.1145/3378936.3378974. ISBN 978-1-4503-7690-7. Dostupné z: https://www.fit.vut.cz/research/publication/12114.

[8] Ogidan, B. *Graph databases, Why are they important* [online]. The Andela Way, říjen 2018 [cit. 2024-04-08]. Dostupné z: https://medium.com/the-andela-way/graph-databases-why-are-they-important-c438e1a224ae.

[9] Procházková, M. *Matriky a jejich minulost, současnost a budoucnost [online]*. 2020 [cit. 2024-04-08]. [cit. 2024-04-08]. Diplomová práce. Masarykova univerzita, Filozofická fakulta, Brno. SUPERVISOR : Jiří Smitka. Dostupné z: https://is.muni.cz/th/ubws2/.

[10] Rahien, A. *That No SQL Thing – Document Databases* [online]. FIT VUT v Brně, duben 2010 [cit. 2024-04-08]. Dostupné z: https://ayende.com/blog/4459/that-no-sql-thing-document-databases.

[11] SHUBHANJAYTIWARI. *Key-Value Data Model in NoSQL* [online]. Státní oblastní archiv v Praze, únor 2017 [cit. 2024-04-08]. Dostupné z: https://www.geeksforgeeks.org/key-value-data-model-in-nosql/.

[12] SILBERSCHATZ, A., KORTH, H. F. a SUDARSHAN, S. *Database system concepts*. 7. vyd. McGraw-Hill, 2019 [cit. 2024-04-08]. ISBN 9780078022159.

[13] SULLIVAN, C. *Create a graph database in Neo4j using Python* [online]. Leden 2021 [cit. 2024-04-08]. Dostupné z: https://towardsdatascience.com/create-a-graph-database-in-neo4j-using-python-4172d40f89c4.

Příloha A

Obsah přiloženého paměťového média

- doc/ – adresář s technickou zprávou v PDF a s jejími zdrojovými kódy.
- MongoDB/ - adresář s dumpem databáze MongoDB, a skripty pro konverzi databáze a testování dotazů.
- MySQL/ - adresář s dumpem databáze MySQL, a skripty pro testování dotazů.
- Neo4j/ - adresář s dumpem databáze Neo4j, a skripty pro konverzi databáze a testování dotazů.
- ZODB/ - Adresář s databázovým souborem ZODB, a skripty pro konverzi databáze a testování dotazů.
- Readme.md - Uživatelská příručka.
- requirements.txt - soubor, ve kterém je popsán seznam potřebných modulů a balíčků.

Příloha B

První SQL dotaz

```
CREATE DEFINER=‘root‘@‘localhost‘ PROCEDURE ‘birthRecord2‘(IN recid int(10)
    unsigned)
BEGIN
   SET SESSION sql_mode=(SELECT REPLACE(@@sql_mode,'ONLY_FULL_GROUP_BY',''));
   drop temporary table if exists tempNames;
   drop table if exists TEMP_personFull;

    create temporary table tempNames
    select birth.id as id, group_concat(name.name separator ' ') as names
         from birth
         join (birth_confirm_name, name)
             on birth_confirm_name.birth_id=birth.id and
                 birth_confirm_name.name_id=name.id group by birth.id;
    create table TEMP_personFull (id int auto_increment, birth_id int,
        marriage_id int, burial_id int, rel ENUM('main', 'f', 'm', 'midwife',
        'granted', 'f_f', 'f_m', 'f_m_f', 'f_m_m', 'm_f', 'm_m', 'm_m_f',
        'm_m_m', 'gf_1', 'gf_2', 'gf_3', 'gf_4', 'gfrel_1', 'gfrel_2',
        'gfrel_3', 'gfrel_4', 'husband', 'bapt_husband', 'mar_groom',
        'mar_bride', 'mar_priest', 'mar_widower', 'mar_g_f', 'mar_g_m',
        'mar_g_m_f', 'mar_g_m_m', 'mar_g_fost', 'mar_widow', 'mar_b_f',
        'mar_b_m', 'mar_b_m_f', 'mar_b_m_m', 'mar_b_fost', 'mar_sv_1',
        'mar_svrel_1', 'mar_sv_2', 'mar_svrel_2', 'mar_sv_3', 'mar_svrel_3',
        'mar_sv_4', 'mar_svrel_4', 'mar_speaker', 'mar_stara', 'mar_bestman',
        'mar_bridesmaid', 'bur_examinator', 'bur_keeper', 'bur_gravedigger',
        'bur_main', 'bur_f', 'bur_m', 'bur_m_f', 'bur_m_m', 'bur_husband',
        'bur_wife', 'bur_son', 'bur_daughter', 'bur_rel1'),
    title varchar(255), snameFull varchar(255), nameFull varchar(255),
        domicileFull varchar(255), street varchar(255), descr_num varchar(255),
    religion ENUM('catholic', 'protestant', 'jew', 'none'),
    birth_date varchar(255), dead tinyint(1), waif tinyint(1), category
        char(1), occNames varchar(255),
    personRelationFull varchar(255), widow tinyint(1), legitimate tinyint(1),
        dead_date varchar(255), work_place varchar(255),
    age DECIMAL(3,2), primary key(id));
    insert into TEMP_personFull(birth_id, marriage_id, burial_id, rel, title,
        sname Full, nameFull, domicileFull, street,
    descr_num, religion, birth_date, dead, waif, category, occNames,
        personRelationFull, widow, legitimate, dead_date, work_place,
    age)
```

```
21      select person.birth_id, person.marriage_id, person.burial_id, rel, title,
            surname.name as snameFull,
22            group_concat(name.name separator ' ') as nameFull, domicile.name as
                domicileFull, street, descr_num, religion, birth_date,
23      dead, waif, category, occups.occNames,personRelation.name as
            personRelationFull,
24            widow, person.legitimate, dead_date, work_place, age from person
25      left join (person_name, name)
26          on person_name.person_id=person.id and person_name.name_id=name.id and
                person.birth_id = recid
27      left join (surname)
28          on person.sname=surname.id
29      left join (domicile)
30          on person.domicile=domicile.id
31      left join (personRelation)
32          on person.person_relation=personRelation.id
33      left join
34          (select person.id, group_concat(occupation.name separator '; ') as
                occNames
35            from person
36            join (person_occup, occupation)
37                on person_occup.person_id=person.id and
                      person_occup.occup_id=occupation.id group by person.id)
38           as occups
39          on person.id=occups.id
40      where birth_id=recid
41      group by rel;
42  select user.name, b.fin, archive.short, archiveFond.name, register.signature,
        b.scan, b.lay, b.pos, main.birth_date, b.baptism_date,
43      gr.nameFull as gr_name, gr.snameFull as gr_sname, gr.occNames as gr_occ,
44      main.domicileFull as domicile, main.street as street, main.descr_num as
            dnum,
45      mw.nameFull as mw_name, mw.snameFull as mw_sname, mw.domicileFull as
            mw_domicile, mw.descr_num as mw_dnum,
46      main.dead as dead,
47      main.nameFull as name, main.snameFull as sname, b.mult, b.sex,
            b.legitimate, main.religion, b.parents_marr_when, main.waif, main.title,
48      f.dead as f_dead, f.title as f_title, f.nameFull as f_name, f.snameFull as
            f_sname, f.occNames as f_occ, f.domicileFull as f_domicile,
49      f.street as f_street, f.descr_num as f_dnum, f.religion as f_religion,
            f.birth_date as f_birth_date, f.waif as f_waif,
50      ff.dead as ff_dead, ff.title as ff_title, ff.nameFull as ff_name,
            ff.snameFull as ff_sname, ff.occNames as ff_occ, ff.domicileFull as
            ff_domicile,
51      ff.street as ff_street, ff.descr_num as ff_dnum, ff.waif as ff_waif,
52      fm.dead as fm_dead, fm.title as fm_title, fm.nameFull as fm_name,
            fm.snameFull as fm_sname, fm.occNames as fm_occ, fm.domicileFull as
            fm_domicile,
53      fm.street as fm_street, fm.descr_num as fm_dnum, fm.waif as fm_waif,
54      fmf.title as fmf_title, fmf.nameFull as fmf_name, fmf.snameFull as
            fmf_sname, fmf.occNames as fmf_occ, fmf.domicileFull as fmf_domicile,
55      fmf.street as fmf_street, fmf.descr_num as fmf_dnum, fmf.waif as fmf_waif,
56      fmm.title as fmm_title, fmm.nameFull as fmm_name, fmm.snameFull as
            fmm_sname, fmm.occNames as fmm_occ, fmm.domicileFull as fmm_domicile,
```

```
57      fmm.street as fmm_street, fmm.descr_num as fmm_dnum, fmm.waif as fmm_waif,
58      m.title as m_title, m.nameFull as m_name, m.snameFull as m_sname,
            m.occNames as m_occ, m.domicileFull as m_domicile,
59      m.street as m_street, m.descr_num as m_dnum, m.religion as m_religion,
            m.birth_date as m_birth_date, m.waif as m_waif,
60      mf.dead as mf_dead, mf.title as mf_title, mf.nameFull as mf_name,
            mf.snameFull as mf_sname, mf.occNames as mf_occ, mf.domicileFull as
            mf_domicile,
61      mf.street as mf_street, mf.descr_num as mf_dnum, mf.waif as mf_waif,
62      mm.dead as mm_dead, mm.title as mm_title, mm.nameFull as mm_name,
            mm.snameFull as mm_sname, mm.occNames as mm_occ, mm.domicileFull as
            mm_domicile,
63      mm.street as mm_street, mm.descr_num as mm_dnum, mm.waif as mm_waif,
64      mmf.title as mmf_title, mmf.nameFull as mmf_name, mmf.snameFull as
            mmf_sname, mmf.occNames as mmf_occ, mmf.domicileFull as mmf_domicile,
65      mmf.street as mmf_street, mmf.descr_num as mmf_dnum, mmf.waif as mmf_waif,
66      mmm.title as mmm_title, mmm.nameFull as mmm_name, mmm.snameFull as
            mmm_sname, mmm.occNames as mmm_occ, mmm.domicileFull as mmm_domicile,
67      mmm.street as mmm_street, mmm.descr_num as mmm_dnum, mmm.waif as mmm_waif,
68      gf1.title as gf1_title, gf1.nameFull as gf1_name, gf1.snameFull as
            gf1_sname, gf1.occNames as gf1_occ, gf1.domicileFull as gf1_domicile,
69      gf1.street as gf1_street, gf1.descr_num as gf1_dnum, gf1.category as
            gf1_category, gf1.personRelationFull as gf1_person_relation,
70      gfrel1.title as gfrel1_title, gfrel1.nameFull as gfrel1_name,
            gfrel1.snameFull as gfrel1_sname, gfrel1.occNames as gfrel1_occ,
71      gfrel1.domicileFull as gfrel1_domicile,
72      gf2.title as gf2_title, gf2.nameFull as gf2_name, gf2.snameFull as
            gf2_sname, gf2.occNames as gf2_occ, gf2.domicileFull as gf2_domicile,
73      gf2.street as gf2_street, gf2.descr_num as gf2_dnum, gf2.category as
            gf2_category, gf2.personRelationFull as gf2_person_relation,
74      gfrel2.title as gfrel2_title, gfrel2.nameFull as gfrel2_name,
            gfrel2.snameFull as gfrel2_sname, gfrel2.occNames as gfrel2_occ,
75      gfrel2.domicileFull as gfrel2_domicile,
76      gf3.title as gf3_title, gf3.nameFull as gf3_name, gf3.snameFull as
            gf3_sname, gf3.occNames as gf3_occ, gf3.domicileFull as gf3_domicile,
77      gf3.street as gf3_street, gf3.descr_num as gf3_dnum, gf3.category as
            gf3_category, gf3.personRelationFull as gf3_person_relation,
78      gfrel3.title as gfrel3_title, gfrel3.nameFull as gfrel3_name,
            gfrel3.snameFull as gfrel3_sname, gfrel3.occNames as gfrel3_occ,
79      gfrel3.domicileFull as gfrel3_domicile,
80      gf4.title as gf4_title, gf4.nameFull as gf4_name, gf4.snameFull as
            gf4_sname, gf4.occNames as gf4_occ, gf4.domicileFull as gf4_domicile,
81      gf4.street as gf4_street, gf4.descr_num as gf4_dnum, gf4.category as
            gf4_category, gf4.personRelationFull as gf4_person_relation,
82      gfrel4.title as gfrel4_title, gfrel4.nameFull as gfrel4_name,
            gfrel4.snameFull as gfrel4_sname, gfrel4.occNames as gfrel4_occ,
83      gfrel4.domicileFull as gfrel4_domicile,
84      b.signs, b.confirmation_when, b.confirmation_where, conf_name.names as
            conf_name, surname.name as conf_sname,
85      bm1.m_when as marr1_when, bm1.m_where as marr1_where, bm1.nameFull as
            marr1_name, bm1.snameFull as marr1_sname, bm1.occNames as marr1_occ,
86      bm1.domicileFull as marr1_domicile, bm1.street as marr1_street,
            bm1.descr_num as marr1_dnum,
```

```
87      bm2.m_when as marr2_when, bm2.nameFull as marr2_name, bm2.snameFull as
            marr2_sname, bm2.occNames as marr2_occ,
88      bm2.domicileFull as marr2_domicile, bm2.street as marr2_street,
            bm2.descr_num as marr2_dnum, bm2.m_where as marr2_where,
89      main.dead_date, b.dead_where, b.church_getoff, b.church_getoff_where,
            b.church_reenter,
90      h.dead as h_dead, h.nameFull as h_name, h.snameFull as h_sname, h.occNames
            as h_occ, h.domicileFull as h_domicile,
91      h.street as h_street, h.descr_num as h_dnum,b.comment
92      from TEMP_personFull, birth as b
93      join (register) on b.register_id=register.id
94      left join (archive) on register.archive_id=archive.id
95      left join (archiveFond) on register.fond_id=archiveFond.id
96      left join (TEMP_personFull as main) on main.birth_id=b.id and
            main.rel='main'
97      left join (TEMP_personFull as gr) on gr.birth_id=b.id and gr.rel='granted'
98      left join (TEMP_personFull as mw) on mw.birth_id=b.id and mw.rel='midwife'
99      left join (TEMP_personFull as f) on f.birth_id=b.id and f.rel='f'
100     left join (TEMP_personFull as m) on m.birth_id=b.id and m.rel='m'
101     left join (TEMP_personFull as ff) on ff.birth_id=b.id and ff.rel='f_f'
102     left join (TEMP_personFull as fm) on fm.birth_id=b.id and fm.rel='f_m'
103     left join (TEMP_personFull as fmf) on fmf.birth_id=b.id and fmf.rel='f_m_f'
104     left join (TEMP_personFull as fmm) on fmm.birth_id=b.id and fmm.rel='f_m_m'
105     left join (TEMP_personFull as mf) on mf.birth_id=b.id and mf.rel='m_f'
106     left join (TEMP_personFull as mm) on mm.birth_id=b.id and mm.rel='m_m'
107     left join (TEMP_personFull as mmf) on mmf.birth_id=b.id and mmf.rel='m_m_f'
108     left join (TEMP_personFull as mmm) on mmm.birth_id=b.id and mmm.rel='m_m_m'
109     left join (TEMP_personFull as gf1) on gf1.birth_id=b.id and gf1.rel='gf_1'
110     left join (TEMP_personFull as gf2) on gf2.birth_id=b.id and gf2.rel='gf_2'
111     left join (TEMP_personFull as gf3) on gf3.birth_id=b.id and gf3.rel='gf_3'
112     left join (TEMP_personFull as gf4) on gf4.birth_id=b.id and gf4.rel='gf_4'
113     left join (TEMP_personFull as gfrel1) on gfrel1.birth_id=b.id and
            gfrel1.rel='gfrel_1'
114     left join (TEMP_personFull as gfrel2) on gfrel2.birth_id=b.id and
            gfrel2.rel='gfrel_2'
115     left join (TEMP_personFull as gfrel3) on gfrel3.birth_id=b.id and
            gfrel3.rel='gfrel_3'
116     left join (TEMP_personFull as gfrel4) on gfrel4.birth_id=b.id and
            gfrel4.rel='gfrel_4'
117     left join (TEMP_personFull as h) on h.birth_id=b.id and h.rel='husband'
118     left join (birthMarriageFull as bm1) on bm1.birth_id=b.id and bm1.num=1
119     left join (birthMarriageFull as bm2) on bm2.birth_id=b.id and bm2.num=2
120     left join
121         tempNames as conf_name
122         on b.id=conf_name.id
123     left join (surname)
124         on b.confirmation_sname=surname.id
125     where b.id = recid
126 group by user.name;
127
128 drop table if exists TEMP_personFull;
129 drop temporary table if exists tempNames;
130 END
```

Příloha C

Sedmý SQL dotaz

```
select archive.short as archive, archiveFond.name as fond, register.signature,
    birth.id, birth.scan, birth.pos, rel,
      title, FullName.PersonNames, surname.name as snameFull, domicile.name as
          domicileFull, street, descr_num, religion, person.birth_date, dead,
          waif, category, occups.occNames, personRelation.name as
          personRelationFull, widow, person.legitimate, dead_date, work_place,
          age
from surname
join person
   on person.sname = surname.id
join birth
    on birth.id = person.birth_id
left join (domicile)
    on person.domicile=domicile.id
left join (personRelation)
    on person.person_relation=personRelation.id
left join
    (select person.id, group_concat(occupation.name separator '; ') as occNames
     from person
     join (person_occup, occupation)
         on person_occup.person_id=person.id and
             person_occup.occup_id=occupation.id WHERE person.sname in (SELECT
             id from surname where name = 'Antl') group by person.id)
     as occups
    on person.id=occups.id
left join
    (select person.id, group_concat(name.name separator '; ') as PersonNames
     from person
     join (person_name, name)
         on person_name.person_id=person.id and person_name.name_id=name.id
             WHERE person.sname in (SELECT id from surname where name = 'Antl')
             group by person.id)
     as FullName
    on person.id=FullName.id
join (register) on birth.register_id=register.id
left join (archive) on register.archive_id=archive.id
left join (archiveFond) on register.fond_id=archiveFond.id
where surname.name = 'Antl';
```

Příloha D

Devátý SQL dotaz

```
select archive.short as archive, archiveFond.name as fond, register.signature,
    birth.id, birth.scan, birth.pos, rel,
       title, normalizedName.name, name.name, surname.name as snameFull,
       domicile.name as domicileFull, street, descr_num, religion,
           person.birth_date,
       dead, waif, category, occups.occNames, personRelation.name as
           personRelationFull,
       widow, person.legitimate, dead_date, work_place, age
from normalizedName
join name
    on name.norm_name_id = normalizedName.id
join (person_name, person)
    on person_name.person_id=person.id and person_name.name_id=name.id
join birth
    on birth.id = person.birth_id
left join (surname)
    on person.sname=surname.id
left join (domicile)
    on person.domicile=domicile.id
left join (personRelation)
    on person.person_relation=personRelation.id
left join
    (select person.id, group_concat(occupation.name separator '; ') as occNames
      from person
      join (person_occup, occupation)
          on person_occup.person_id=person.id and
              person_occup.occup_id=occupation.id group by person.id)
     as occups
    on person.id=occups.id
join (register) on birth.register_id=register.id
left join (archive) on register.archive_id=archive.id
left join (archiveFond) on register.fond_id=archiveFond.id

where normalizedName.name = 'František';
```

Příloha E

První Cypher dotaz

```
MATCH (bth:birth{id:53179})
OPTIONAL MATCH (bth)-[:person_birth_id__birth_id]-(main:person{rel:'main'})
OPTIONAL MATCH (bth)-[:birth_confirm_name]-(bcn:name)
OPTIONAL MATCH (bth)--(birth_confirm_surn:surname)
OPTIONAL MATCH (mr1:birthMarriage{num:1})-[:birthMarriage_birth_id__birth_id]-(bth)
OPTIONAL MATCH (mr1)-[:birthMarriage_name]-(mn1:name)
OPTIONAL MATCH (mr1)-[:birthMarriage_occup]-(bmo1:occupation)
OPTIONAL MATCH (mr1)-[:birthMarriage_m_sname__surname_id]-(bms1:surname)
OPTIONAL MATCH (mr1)-[:birthMarriage_domicile__domicile_id]-(bmdom1:domicile)
OPTIONAL MATCH (mr2:birthMarriage{num:2})-[:birthMarriage_birth_id__birth_id]-(bth)
OPTIONAL MATCH (mr2)-[:birthMarriage_name]-(mn2:name)
OPTIONAL MATCH (mr2)-[:birthMarriage_occup]-(bmo2:occupation)
OPTIONAL MATCH (mr2)-[:birthMarriage_m_sname__surname_id]-(bms2:surname)
OPTIONAL MATCH (mr2)-[:birthMarriage_domicile__domicile_id]-(bmdom2:domicile)
OPTIONAL MATCH (psn:person)-[:person_birth_id__birth_id]-(bth)
OPTIONAL MATCH (rg:register)-[:birth_register_id__register_id]-(bth)
OPTIONAL MATCH (ar:archive)-[:register_archive_id__archive_id]-(rg:register)
OPTIONAL MATCH (af:archiveFond)-[:archiveFond_archive_id__archive_id]-(rg:register)
OPTIONAL MATCH (u:user)-[:birth_owner__user_id]-(bth)
OPTIONAL MATCH (psn)-[:person_name]-(psn_pn :name)
OPTIONAL MATCH (psn)-[:person_sname__surname_id]-(psn_sn :surname)
OPTIONAL MATCH (psn)-[:person_domicile__domicile_id]-(psn_dm :domicile)
OPTIONAL MATCH (psn)-[:person_person_relation__personRelation_id]-(psn_pr :personRelation)
OPTIONAL MATCH (psn)-[:person_occup]-(psn_oc :occupation)
WITH u.name as owner, bth.fin as fin, ar.short as short, af.name as af_name, rg.signature as signature, bth.scan as scan,
bth.lay as lay, bth.pos as pos, bth.baptism_date as baptism_date, bth.mult as mult, bth.sex as sex,
bth.legitimate as legetimate, bth.parents_marr_when as parents_marr_when, bth.signs as signs,
bth.confirmation_when as confirmation_when, bth.confirmation_where as confirmation_where, bth.dead_where as dead_where,
bth.church_getoff as church_getoff, bth.church_getoff_where as church_getoff_where, bth.church_reenter as church_reenter,
```

```
30 bth.comment as comment, apoc.text.join(COLLECT(DISTINCT psn_pn.name),' ') as
       psn_name, psn_sn, psn_dm, psn_pr,
31 apoc.text.join(COLLECT(DISTINCT psn_oc.name),';') as psn_occupation, apoc.text.
       join(COLLECT(DISTINCT mn1.name),' ') as m1_nm,
32 apoc.text.join(COLLECT(DISTINCT mn2.name),' ') as m2_nm, apoc.text.join(COLLECT(
       DISTINCT bcn.name),' ') as birth_name,
33 apoc.text.join(COLLECT(DISTINCT bmo1.name),';') as bm_occ1, apoc.text.join(COLLECT
       (DISTINCT bmo2.name),';') as bm_occ2,
34 birth_confirm_surn, psn as psn,
35 mr1.m_when as mr1_m_when, mr1.m_where as mr1_m_where, bms1.name as mr1_surnname,
       bmdom1.name as mr1_domicile,
36 mr1.street as mr1_street, mr1.descr_num as mr1_descr_num,
37 mr2.m_when as mr2_m_when, mr2.m_where as mr2_m_where, bms2.name as mr2_surnname,
       bmdom2.name as mr2_domicile,
38 mr2.street as mr2_street, mr2.descr_num as mr2_descr_num
39 WITH owner, fin, short, af_name, signature, scan, lay, pos, baptism_date, mult,
       sex, legetimate, parents_marr_when, signs,
40 confirmation_when, confirmation_where, dead_where, church_getoff,
       church_getoff_where, church_reenter,  m1_nm, m2_nm, birth_name,
41 bm_occ1, bm_occ2, birth_confirm_surn,
42 COLLECT({rel: psn.rel, dead: psn.dead, title: psn.title, name: psn_name, surname:
       psn_sn.name, occupation:psn_occupation,
43 domicile:psn_dm.name, street: psn.street, descr_num: psn.descr_num, religion:psn.
       religion, birth_date:psn.birth_date,
44 waif:psn.waif, category: psn.category, person_relation: psn_pr.name})as person,
       mr1_m_when, mr1_m_where, mr1_surnname, mr1_domicile,
45 mr1_street, mr1_descr_num, mr2_m_when, mr2_m_where, mr2_surnname, mr2_domicile,
       mr2_street, mr2_descr_num, comment
46 RETURN DISTINCT owner, fin, short, af_name, signature, scan, lay, pos,
       baptism_date, mult, sex, legetimate, parents_marr_when,
47 signs, confirmation_when, confirmation_where, dead_where, church_getoff,
       church_getoff_where, church_reenter, birth_name as birth_confirm_name,
48 birth_confirm_surn.name as birth_confirm_surname, mr1_m_when, mr1_m_where, m1_nm,
       mr1_surnname,
49 bm_occ1 as mr1_occupation, mr1_domicile, mr1_street, mr1_descr_num, mr2_m_when,
       mr2_m_where, m2_nm, mr2_surnname,
50 bm_occ2 as mr2_occupation, mr2_domicile, mr2_street, mr2_descr_num, person,
       comment
```

Příloha F

Sedmý Cypher dotaz

```
MATCH (sn:surname{name:'Antl'})-[:person_sname__surname_id]-(psn:person)-[:
    person_birth_id__birth_id]-(bth:birth)
MATCH (bth)-[:birth_register_id__register_id]-(reg:register)
OPTIONAL MATCH (reg)-[:register_archive_id__archive_id]-(ar:archive)
OPTIONAL MATCH (reg)-[:register_fond_id__archiveFond_id]-(af:archiveFond)
OPTIONAL MATCH (psn)-[:person_domicile__domicile_id]-(dom:domicile)
OPTIONAL MATCH (psn)-[:person_person_relation__personRelation_id]-(pr:
    personRelation)
OPTIONAL MATCH (psn)-[:person_occup]-(occ:occupation)
OPTIONAL MATCH (psn)-[:person_name]-(name:name)
RETURN DISTINCT ar.short as archive, af.name as fond, reg.signature, bth.id, bth.
    scan, bth.pos, psn.rel,
       psn.title, apoc.text.join(COLLECT(DISTINCT name.name),' ') as FullName, sn.
    name as snameFull,
       dom.name as domicileFull, psn.street, psn.descr_num, psn.religion, psn.
    birth_date,
       psn.dead, psn.waif, psn.category, apoc.text.join(COLLECT(DISTINCT occ.name)
    ,';') as occupation,
       pr.name as personRelationFull, psn.widow, psn.legitimate, psn.dead_date,
    psn.work_place, psn.age
```

Příloha G

Devátý Cypher dotaz

```
MATCH (nname:normalizedName{name: 'František'})-[:
    name_norm_name_id__normalizedName_id]-(name:name)-[:person_name]-(psn)-[:
    person_birth_id__birth_id]-(bth:birth)-[:birth_register_id__register_id]-(reg:
    register)
OPTIONAL MATCH (reg)-[:register_archive_id__archive_id]-(ar:archive)
OPTIONAL MATCH (reg)-[:register_fond_id__archiveFond_id]-(af:archiveFond)
OPTIONAL MATCH (psn)-[:person_domicile__domicile_id]-(dom:domicile)
OPTIONAL MATCH (psn)-[:person_person_relation__personRelation_id]-(pr:
    personRelation)
OPTIONAL MATCH (psn)-[:person_occup]-(occ:occupation)
OPTIONAL MATCH (sn:surname)-[:person_sname__surname_id]-(psn)
RETURN DISTINCT ar.short as archive, af.name as fond, reg.signature, bth.id, bth.
    scan, bth.pos, psn.rel,
       psn.title, nname.name as normalizedName, name.name, sn.name as snameFull,
       dom.name as domicileFull, psn.street, psn.descr_num, psn.religion, psn.
    birth_date,
       psn.dead, psn.waif, psn.category, apoc.text.join(COLLECT(DISTINCT occ.name)
    ,';') as occupation,
       pr.name as personRelationFull, psn.widow, psn.legitimate, psn.dead_date,
    psn.work_place, psn.age
```

Příloha H

Příklad záznamu o narozeni


```
{
  "birth": {
    "check_req": 0,
    "owner": 11,
    "mult": 1,
    "lay": "L",
    "sex": "F",
    "scan": 10,
    "fin": 0,
    "register_id": 7819,
    "score": 2.00008667,
    "baptism_date": "1670-11-16",
    "pos": 1,
    "legitimate": "U",
    "id": 52,
    "lang": "SC"
  },
  "persons": [
    {
      "person": {
        "birth_id": 52,
        "rel": "main",
        "dead": 0,
        "id": 394,
        "religion": "none"
      },
      "names": [
        {
          "name_order": 1,
          "name": {
            "cluster_id": 11,
            "sex": "F",
            "name": "Alzbieta",
            "id": 162
          }
        }
      ]
    },
    {...},
```

```
    {...},
    {...},
  ],
  "reg": {
    "signature": "46",
    "admin_id": 39,
    "fond_id": 1,
    "scan_count": 103,
    "lang3": "NONE",
    "lang2": "CZ",
    "id": 7819,
    "type": "MAT",
    "lang1": "LAT",
    "archive_id": 7
  }
}
```

Výpis H.1: Příklad záznamu o narozeni

Příloha I

Sedmý Mongo dotaz

```
db.getCollection('birth_record').aggregate(
  [
    {
      $match: { 'persons.surname.name': 'Antl' }
    },
    { $unwind: '$persons' },
    {
      $match: { 'persons.surname.name': 'Antl' }
    },
    {
      $project: {
        id: '$bth.id',
        sname: '$persons.surname.id',
        surname: '$persons.surname.name',
        rel: '$persons.person.rel'
      }
    },
        {
      $lookup: {
        from: "archive",
        localField: "reg.archive_id",
        foreignField: "_id",
        as: "archive",
    },
    },
    {$unwind:{path: "$archive", preserveNullAndEmptyArrays: true,},},
    {
      $lookup: {
        from: "archiveFond",
        localField: "reg.fond_id",
        foreignField: "_id",
        as: "archiveFond",
    },
    },
    {$unwind:{path: "$archiveFond", preserveNullAndEmptyArrays: true,},
    },
  ]
);
```