VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

DATABÁZE PRO UKLÁDÁNÍ GENEALOGICKÝCH DAT

DATABASE FOR GENEALOGICAL DATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MAREK KAFONĚK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JAROSLAV ROZMAN, Ph.D**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Kafoněk Marek**
Program: Informační technologie
Název: **Databáze pro ukládání genealogických dat**
Database for Genealogical Data
Kategorie: Databáze

Zadání:

1. Nastudujte problematiku genealogie. Nastudujte relační databáze a grafovou databázi Neo4j.
2. Na základě nastudovaných znalostí navrhněte MySQL databázi pro ukládání genealogckých dat (příbuzenské a další vztahy, vazba mezi původním genealogickým záznamem a záznamem v databázi). Dále navrhněte skripty pro přenos dat z Neo4j do MySQL databáze.
3. Navrženou databázi a potřebné skripty vytvořte a proveďte překopírování dat z Neo4j do MySQL.
4. Vytvořenou databázi otestujte.

Literatura:

- Prostredníková Hana: Pokročilé zobrazování genealogických dat, bakalářská práce, Brno, FIT VUT v Brně, 2016.
- Valecký Dušan: Zobrazování genealogických dat, bakalářská práce, Brno, FIT VUT v Brně, 2017.

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání

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

Vedoucí práce: **Rozman Jaroslav, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 3. listopadu 2021

Abstrakt

Tato práce se zabývá možným převodem genealogických dat mezi grafovou a relační databází, kdy se zde navazuje na již existující data uložené v grafové databázi Neo4j. Možné převody budou sloužit k tomu, aby se s danými daty mohlo podle potřeby pracovat buď pomocí dotazovacího jazyku SQL v relační databázi, nebo pomocí jazyku Cypher v grafové databázi.

Abstract

This work deals with the possible transfer of genealogical data between graph and relational databasis, which builds on existing data stored in the graph database Neo4j. Possible transfers will be used to work with the data as needed using SQL in a relational database, or using Cypher in a chart database.

Klíčová slova

Genealogie, MySQL, Relační databáze, Neo4j, Grafová databáze

Keywords

Genealogy, MySQL, Relational database, Neo4j, Graph database

Citace

KAFONĚK, Marek. *Databáze pro ukládání genealogických dat.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jaroslav Rozman, Ph.D

Databáze pro ukládání genealogických dat

Prohlášení

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

.....
Marek Kafoněk
18. května 2022

Poděkování

Chtěl bych poděkovat mému vedoucímu panu Ing.Jaroslavovi Rozmanovi Ph.D. za ochotu a vstřícnost pří řešení bakalářské práce

Obsah

Kapitola 1

Úvod

Při nedávné zmínce jednoho z vyučujících na přednášce, ohledně vyhledávání genealogických dat, jsem začal přemýšlet, kdo vlastně byli mí předkové. Byli to venkované, bohatí hospodáři, měšťané nebo snad dokonce šlechta? Jsou mí předci spojeni pouze s životem na území naší republiky nebo mé kořeny sahají i do ciziny?

Tyto otázky se prací na téma této bakalářské práce sice nedozvím, ale rozhodně mi pomůže si nastudovat informace ohledně práce s genealogickými daty a případně mít možnost využít výslednou implementaci pro mé vyhledávání.

Samotnou práci jsem si podle bodů zadání rozdělil na 4 části. První část se věnuje studiu problematiky tématu zadané práce, kde jsem popsal potřebné informace k nastudování tak, abych ve druhé časti mohl provést návrh mého řešení. Za zmínku stojí, že už v návrhu je potřeba zohlednit, že toto téma navazuje na již existující diplomovou práci Generování rodokmenů z matričních záznamů, provedenou na fakultě informatiky, od Ing. Lucie Tušimové [12]. Další částí je samotná implementace, kdy je pomocí předchozího návrhu vytvořena MySQL databáze, a k ní skripty potřebné k přesunutí dat z grafové databáze, která byla vytvořena ve zmiňované návazné diplomové práci. Nakonec v poslední části je zmíněno testování výsledného implementovaného řešení.

Kapitola 2

Studium problematiky

K zadání mé práce bylo potřeba nastudovat problematiku témat související se zadáním. Jedná se celkem o tři problematiky a to genealogie, relační databáze MySQL a grafová databáze Neo4j. U Genealogie jsem, kromě samotného významu tohoto oboru, studoval tvorbu možných genealogických výstupů (rodokmen, rozvod, vývod) a princip získávání genealogických dat. U MySQL databáze jsem se věnoval studiu její funkcionality, tvorby ER diagramů a jejich převod do databáze a případně další práce s databází. V poslední řadě jsem se zaměřil na studium Neo4j, kde jsem, stejně jako u MySQL, musel nastudoval její funkcionalitu, princip ukládání dat, a pro mé téma důležitý, export dat.

2.1 Genealogie

Jedná se o pomocnou vědu historickou, která zkoumá vztahy mezi lidmi. První zmínky o genealogii pocházejí již ze starověku, odkud se dochovaly například rodokmeny bohů starověkého Řecka nebo příbuzenské vztahy mezi faraony v Egyptě. Každopádně pro nás zajímavější jsou pozůstatky rodopisů evropských královských rodin z raného středověku [1][7][8].

Genealogii dělíme na vědeckou a soukromou. Pro veřejnost je známější soukromá, které se také říká rodopis. Tato věda se omezuje na zájmy jednotlivců, kteří se zabývají historií své rodiny z osobních důvodů, poznání dějin, dědictví a dalších jimi podobných. Vědecká genealogie se zabývá rodinnými vztahy jedince, bez ohledu na jeho zájmy, a data z ní získaná se dále používají v rámci návazných vědeckých oborů.

2.1.1 Vyhledávání genealogických dat

K získávání genealogických dat slouží matriky, které se dají rozdělit na živé a neživé. Badatelé, jak se nazývají lidé, kteří vyhledávají genealogické záznamy, mají přístup pouze k tzv. neživým, které jsou k dispozici v archívech. Živé matriky obsahují současně žijící lidi a nikdo by k nim neměl mít přístup.

Původ slova matrika pochází z latinského slova matricula, což v překladu znamená seznamy duchovních. První zmínky o zápisu do matrik na území dnešní České republiky pochází z 16. století, ale jedná se pouze o zápisy evangelické církve. Katolická církev začala zapisovat údaje až v pozdější době. První zápisy matrik byly pouze formou seznamu, který

umožňoval duchovnímu mít přehled o osobách v jeho farnosti. Do matrik kromě duchovních zapisovali i někteří vzdělaní učitelé, kterých ale v té době nebylo tolik. V průběhu let, a s přibývajícími informacemi, docházelo k tomu, že se různé zápisy mohly lišit. Jelikož v té době nebylo přesně stanoveno jakou podobu má mít zápis do matriky, pochází odtud řada nejasností. Ať už duplicitní záznamy pro určitou osobu, zápis události do špatné matriky, nečitelné zápisy a podobně. Docházelo i k takovým případům, že farář odešel do jiné farnosti, vzal si s sebou matriku bývalé farnosti, kam poté začal zapisovat osoby z farnosti nové. Až od roku 1784 se matriky staly úředním dokumentem, který měly ve správě především farnosti, protože byly zodpovědné jak za křty, svatby, tak i za pohřby. U zapisování do matrik se stanovilo jaké informace mají dané zápisy obsahovat, aby nedocházelo k chybám z předchozích let. I přes veškeré úsilí sjednotit zapisované informace stále docházelo k chybám při zápisu, protože faráři přestali zapisovat důležité informace, které dříve většina z nich zapisovala a ztížilo se tím bádání po rodinné historii.

V roce 1870 začaly vznikat civilní matriky, které byly vedeny u okresních soudů. Zpočátku byly určeny pouze pro nevěřící lid. Později se do těchto matrik začali zapisovat i věřící lidé, kteří uzavřeli civilní sňatek.

Až od roku 1949 bylo zavedeno, že matriky připadnou státu a budou pod správou jednotlivých národních výborů. Část matrik, přibližně do roku 1875, byla předána do státních archívů a část zůstala na úřadech. I přes přesun matrik do státní správy mají své vlastní zápisy farností někteří faráři. Na počátku 20. let 21. století se v archívu nachází matriky narozených do roku 1910 a matriky oddaných a zemřelých do roku 1930 [7].

2.1.2 Matriční zápisy

Zápisy se rozdělují na tři části, kterými jsou křestní zápisy, oddací zápisy a úmrtní zápisy. V křestních zápisech se evidují záznamy o narozeném dítěti. Přesněji jeho jméno, příjmení, datum a místo narození. Kromě informací o narozeném se uvádí i oba rodiče, případně jestli se jedná o nemanželské nebo adoptované dítě. Jak už název napovídá, součástí křestních zápisů jsou i informace o samotném křtu, konkrétně se zde uvádí datum a místo křtu, jméno křtitele a jména kmotrů. Ohledně křtu se zde evidují i informace o nouzovém křtu, který se konal předčasně, a to v případě, že dítěti hrozila smrt. Dalšími možnými informacemi v těchto zápisech je náboženství narozeného, které se převážně dědí po svých rodičích, jméno porodní báby, která mohla provádět již zmiňovaný nouzový křest a další doplňující informace ohledně narozeného. Druhým typem zápisů jsou oddací zápisy. Uvádí se zde základní informace jako, datum a místo svatby, u snoubenců se evidují jejich jména, věk, bydliště, stav a rodiče. U oddacích zápisů dále nesmí chybět jméno oddávajícího a jména svědků. Dalšími možnými informacemi je náboženství prováděné svatby a ohláška, která měla informovat o konání svatby. U oddacích záznamů se také uvádí informace o případném rozvodu manželů. Poslední, a z těchto zápisů nejnovější druh, jsou úmrtní zápisy. Evidují se zde informace o jméně, věku a příčině smrti zemřelého a také místo a datum úmrtí. Ohledně pohřbu jsou zde informace o zaopatření zemřelého, datumu a místu pohřbu. V některých případech se uvádějí i svědci pohřbu, a to převážně u pohřbu mrtvorozených, kteří se uváděli jak již ve zmiňovaných křestních zápisech, tak i právě v úmrtních zápisech.

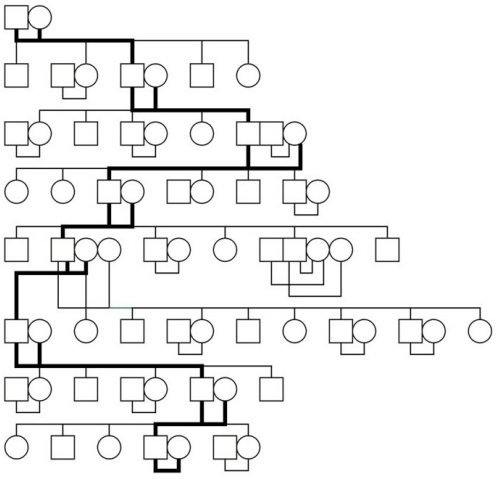
Kromě těchto tří zmiňovaných typů matričních zápisů existují i další doplňkové zápisy, které se věnují zaznamenáváním údajů majetku osob. Mezi takovéto zápisy patří zápisy v gruntovních knihách, zápisy v urbářích, rubriky katastrů a soupis poddaných.

2.1.3 Tvorba výstupů

Po bádání a získání potřebných dat je potřeba tato data prezentovat v nějaké srozumitelné podobě. Pro tento případ jsou k dispozici 3 různé typy, které nyní stručně popíšu.

Rodokmen

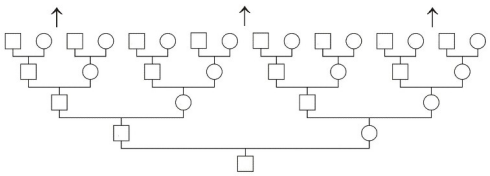
Pro širší veřejnost je nejznámějším druhem zápisu genealogických dat právě rodokmen 2.1. Díky rodokmenu se dohledávají všichni předci v mužské linii. Pro jakoukoliv další práci s genealogickými daty slouží jako odrazový můstek a jeho pomocí začíná veškeré další pátrání a tvorba ostatních druhů výstupu.



Obrázek 2.1: Schéma tvorby rodokmenu [11].

Vývod

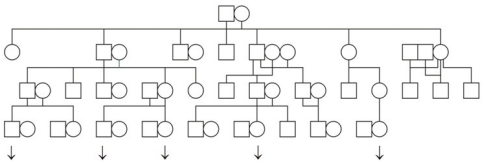
Jedná se o podobný druh jako rodokmen, rozdíl je v tom, že zde se zobrazují všichni předci výchozí osoby a to nejen mužští, ale i ženská linie. Vývod 2.2 je nejčastěji používaný tvar mezi badateli, kdy výsledkem jsou všechny linie rodinných předků. Výsledná podoba vývodu bývá zakreslena v podobě stromu.



Obrázek 2.2: Schéma tvorby vývodu [11].

Rozrod

Nejsložitější variantou pro zobrazení genealogických dat je právě rozrod 2.3. Narozdíl od předchozích variant se zde postupuje opačně, tedy od nejstaršího dohledaného předka do současnosti. Dohledávají se všichni potomci stejného příjmení, potomci synů i nemanželské děti neprovdaných dcer. V případě tvorby rozrodu je často patrné postupné vymírání rodů nebo naopak jejich postupné rozrůstání.



Obrázek 2.3: Schéma tvorby rozrodu [11].

Existuje také možnost všechny zmiňované varianty různě kombinovat. Například se může pátrat pouze po mužské linii, a k ní přidat sourozence každé generace, nebo vyhledat předky ve všech liniích a k nim zahrnout všechny jakékoli příbuzné osoby.

2.2 Relační databáze

V rámci tohoto projektu jsem studoval relační databázi MySQL vyvíjenou společností Oracle, která ji poskytuje v rámci open source software.

Základním kamenem relační databáze je tabulka, která obsahuje sloupce sloužící jako názvy entit a jednotlivé řádky, které znázorňují samotné entity. Pro práci s databází se používá dotazovací jazyk SQL, pomocí kterého můžeme například zobrazovat, vkládat, upravovat nebo mazat jednotlivé záznamy v tabulkách.

2.2.1 Tvorba relační databáze

Konceptuální modelování

Po základní analýze požadavku pro tvorbu relační databáze se vytváří ER diagram, který uvádí vztah mezi entitami v klidovém režimu. Naopak v ER diagramu se neuvádí jednotlivé operace.

Jako entitu zde chápeme objekt reálného světa, který je odlišný od ostatních objektů. Jednotlivé entity jsou součástí entitních množin, které jsou následně použity ve výsledném ER diagramu. Název entitní množiny by měl jasně specifikovat jaké entity se budou do dané množiny ukládat. Vlastnosti entit se nazývají atributy, které se rozlišují na jednoduché (atomické) nebo složené (skládající se z více atributů). Každá entita musí obsahovat jeden atribut, který je potřeba k jednoznačné identifikaci. Zde se tento identifikátor nazývá jako primární klíč v modelu ER diagramu a označuje se «PK».

V ER diagramu se mezi entitními množinami používají binární, ternární nebo unární vztahy. Nejvíce používané jsou binární a unární. Ternární vztahy jsou speciální a jedná se především o vztahy mezi třemi entitními množinami, případně i mezi více entitami. Jednotlivé vztahy by měly mít stručný, ale výstižný název. Název vztahu nemusí být v případě, když vztah vyplývá z názvů entitních množin.

Po entitních množinách a vztazích entitních množin se při tvorbě ER diagramu používá kardinalita, která udává vždy minimální a maximální počet dat vstupující do vztahu s jinou entitou. Možnými kardinalitami jsou 1:1 (jedna entita je ve vztahu také pouze s jednou entitou), 1:N (jedna entita je ve vztahu s více entitami) a M:N (více entit je ve vztahu také s více entitami).

V rámci ER diagramů se mohou použít dvě rozšíření. Prvním rozšířením je slabá entitní množina («weak»), která je závislá na jiné entitní množině («identif»). Slabá entitní množina používá identifikátor dominantní silné entitní množiny a atribut, který identifikuje samotnou slabou entitní množinu. Tento atribut se nazývá diskriminátor a označuje se «D». Nejčastěji je slabá entitní množina ve vztahu se silnou entitní množinou v poměru 1:N. Druhé rozšíření je generalizace/specializace, kde se jedná o vztah dvou entitních množin, kde jedna entitní množina rozšiřuje vlastnosti druhé množiny. V praxi to znamená, že pro entitní množiny je jedna generalizující entitní množina se společnými atributy a pak více specializací, které mají odlišné atributy a dědí atributy generalizující entitní množiny.

Transformace konceptuálního modelu na tabulky relační databáze

Tato část se zaměřuje na návrh tabulek relační databáze z vytvořeného konceptuálního modelu, například z ER diagramu, kdy entitní množiny jsou jednotlivé tabulky. Vztahy mezi nimi jsou znázorněny vazbami mezi tabulkami pomocí primárních a cizích klíčů.

Při návrhu databáze dochází k jejímu chybnému návrhu hlavně v případě opakujících se informací (redundancí), nemožností zobrazovat určitou hodnotu, složitou kontrolu integritních omezení a návrhu zbytečných tabulek.

Na začátek návrhu je potřeba odstranění složených a vícehodnotových atributů na jednotlivé atributy. Následně je dobré se zabývat transformací silných entitních množin, které se dají převést na jednotlivé tabulky, kdy jednotlivé atributy určují sloupce tabulky, jednotlivé řádky jsou samotné entity a každý řádek má přiřazený jeden primární klíč, tak jako primární klíč u entity v entitní množině.

U vytváření vztahů mezi tabulkami se musí zohledňovat kardinalita z ER diagramu a podle ní přidělovat tabulkám cizí klíče, které slouží právě pro vytvoření vazeb mezi tabulkami. U kardinality 1:1 je na návrháři, jaké tabulce přiřadit cizí klíč odkazující na primární klíč druhé tabulky a v případě existujícího dalšího atributu vztahové množiny, bude tento atribut přiřazen k tabulce s cizím klíčem. Narozdíl od předchozí kardinality, tak u kardinality 1:N, je dané, ke které tabulce bude vložený cizí klíč, případně další atributy vztahové množiny a to k tabulce s kardinální hodnotou N. U kardinality M:N je nutné vytvořit speciální vazební tabulku, které bude obsahovat jeden cizí klíč odkazující na primární klíč jedné tabulky a druhý cizí klíč odkazující na primární klíč druhé tabulky. Z těchto dvou cizích

klíčů vznikne složený primární klíč vazební tabulky. Do vazební tabulky se případně přidají i atributy vazební množiny. V případě existence vztahů vyššího stupně (tj. 3 a více), bude zapotřebí vždy vytvořit vazební tabulky spojující všechny tři samotné tabulky.
V případě existenci slabé entitní množiny se bude postupovat jako u entitních množin s kardinalitou 1:N, kdy silná entitní množina se transformuje beze změn a u slabě entitní množiny se transformují všechny její atributy do jedné tabulky společně s cizím klíčem, odkazujícím na primární klíč tabulky pro silnou entitní množinu.

Při transformaci generalizace/specializace lze postupovat třemi způsoby. První je, že pro generalizaci se vytvoří jedna tabulka s jejími atributy a následně se vytvoří tabulky pro jednotlivé specializace, kdy každá specializace obsahuje svoje vlastní atributy a cizí klíč odkazující na primární klíč generalizační tabulky. Druhý způsob je vytvoření tabulek podle počtů specializací, kdy každá tabulka bude obsahovat atributy generalizace a následně jednotlivé tabulky budou mít vlastní atributy jednotlivých specializací. Třetí a zároveň poslední možnost je, že budou vytvořené dvě tabulky. Jedna bude obsahovat atributy generalizace a druhá bude obsahovat cizí klíč odkazující na tabulku generalizace, potom atributy všech specializací a pomocný sloupec, který bude určovat, o kterou specializaci se jedná.

Normalizace schématu databáze

Poslední část pro splnění použitelného návrhu databáze je normalizování. Normalizace se zaměřuje na úpravu chybného návrhu databáze, a to především na redundanci[1], nemožnost reprezentovat určitou informaci a na složitou kontrolu integritních omezení. K tomuto se používají normální formy pro které platí, čím vyšší stupeň normální formy, tím kvalitnější databázový návrh. Normální formy dále definují, jaké vlastnosti musí být splněny v závislosti mezi atributy tabulky.

Při normalizaci databáze je zapotřebí zjistit funkční závislosti, mezi které patří triviální funkční závislost, plná funkční závislost a tranzitivní závislost. K dispozici máme 6 za sebou jdoucích normálních forem v následující podobě, 1. normální formu (1NF), 2. normální formu (2NF), 3. normální formu (3NF), Boyce-Coddovu normální formu (BCNF), 4. normální formu (4NF) a 5. normální formu (5NF). Každá normální forma definuje určitou vlastnost, kterou musí tabulka relační databáze mít, aby tuto formu splňovala. Kromě splnění určité vlastnosti jednotlivých normálních forem, musí být také splněny všechny předchozí normální formy. V případě, že tabulka nesplňuje některou z normálních forem, je potřeba provést bezeztrátovou dekompozici, při které rozdělíme tabulku na více tabulek bez přidání nových atributů nebo odebrání již existujících, zachováme-li závislosti jednotlivých atributů a zbavíme se případné redundance.

Pro splnění 1NF musí všechny jednoduché atributy obsahovat atomické hodnoty. Pro splnění 2NF je potřeba splnění vlastnosti, kdy každý neklíčový atribut je plně funkčně závislý na některém z kandidátních klíčů tabulky. 3NF je splněna za předpokladu, že neexistuje žádný neklíčový atribut, který je tranzitivně závislý na některém kandidátním klíči tabulky. Jelikož předchozí zmíněné normální formy se věnují závislosti neklíčových atributů, byla zavedena BCNF, která zpřísňuje 3NF a odstraňuje závislosti mezi klíčovými atributy. 4NF a 5NF se moc nepoužívají, protože už po splnění BCNF je odstraněna re-

[1] Informace, které se opakují

dundance. Pro určení výsledné normální formy databáze se vybere nejmenší maximální normální forma[6, 4, 9, 10, 3].

2.2.2 Tvorba MySQL databáze

Po studiu návrhu databáze bych se chtěl v této části věnovat studii samotné tvorby MySQL databáze.
Pro práci s MySQL databází se používají dotazy v deklarativním programovacím jazyce SQL. Programovat databázi můžeme buď dotazováním přímo v programovacím prostředí pro SQL nebo pomocí jiných jazyků a jejich příslušných knihoven pro dotazování v SQL (např. C, Python).

Prvním krokem při tvorbě databáze je její samotné vytvoření, pro které se používá dotaz `CREATE DATABASE jmeno_databaze;`, který založí prázdnou databázi, s kterou je možné dále pracovat.

K tomu abychom určili s jakou databází chceme pracovat, je dobré vždy na začátku dotazovacího skriptu použít výběr databáze pomocí `USE jmeno_databaze;`. Výběr databáze se dá používat i jinými způsoby, a to například uvedením názvu databáze před její tabulkou.

Když je vytvořená databáze a jsme k ní připojení, mohou se v ní vytvářet tabulky, a to pomocí dotazu `CREATE TABLE jmeno_tabulky;`. U vytváření tabulky je potřeba ještě definovat její sloupce, k nim datové typy a nastavit, jestli daný sloupec může nebo nemůže obsahovat hodnotu `NULL`. Při vytváření tabulky by se nemělo zapomenout nastavit primární klíč tabulky a případně přiřadit cizí klíče, které budou sloužit k propojení s jinou tabulkou. Pro rychlejší vyhledávání je dobré v tabulce přiřadit indexy, tím zaručíme, že si databázový systém zapamatuje, kde se přibližně nachází indexovaný sloupec a nemusí procházet při vyhledávání celou tabulku.

Pro případ, že je vytvořená tabulka a je potřeba upravit nebo přidat sloupec, slouží dotaz `ALTER TABLE jmeno_databaze prikaz;`. Jako `prikaz` je možné použít buď `ADD COLUMN` pro vložení nového sloupce daného datového typu na konec tabulky, nebo `DROP COLUMN` pro odstranění daného sloupce.
K odstranění tabulek z databáze se použije dotaz `DROP TABLE jmeno_tabulky;`. Pokud daná tabulka obsahuje sloupce ve formě cizího klíče, který zaručuje propojení s jinou tabulkou v databázi, tak se tato tabulka neodstraní a dotaz skončí chybovou hláškou.

Když je tabulka vytvořená, může se s ní začít pracovat, a to prováděním manipulaci s daty, pod které spadá například vkládání nových záznamů, změna existujících záznamů, zobrazení záznamů nebo mazaní záznamů.

Jednou ze zmíněných možností je vkládání nových záznamů do tabulky pomocí příkazu `INSERT INTO jmeno_tabulky (nazev_sloupce) VALUES(hodnota_sloupce);`.
Data se úspěšně vloží do tabulky pokud je přiřazená hodnota všem sloupcům, které nemají povoleno vkládat hodnotu `NULL` a zároveň nemají nastavenou defaultní hodnotu nebo při tvorbě tabulky nebyl použit příkaz `AUTO INCREMENT` pro automatické přidělování hodnot pro daný sloupec.

Pokud je potřeba opravit hodnotu sloupce existujícího záznamu tabulky, poslouží k tomu dotaz `UPDATE nazev_tabulky SET jmeno_sloupce = nova_honota;`. Součástí tohoto dotazu bývá často i dotaz `WHERE`, kde po zadání vhodné podmínky se vybere určitý záznam z tabulky.

K zobrazení hodnot vybraných sloupců z dané tabulky, které splňují určité podmínky, slouží dotaz `SELECT nazev_sloupce FROM nazev_tabulky WHERE podminka;`. Při zobrazování záznamů pomocí dotazu `SELECT` je možné použít další doplňkové dotazy, jako je například `SELECT DISTINTC ...`, který nezobrazí duplicitní záznamy. Je-li potřeba sloučit záznamy v tabulce podle daného slupce, použije se dotaz `GROUP BY` a když je potřeba seřadit záznamy podle daného sloupce, použije se dotaz `ORDER BY`. Když je potřeba zobrazit záznamy z různých tabulek, musí se propojit tyto tabulky dotazem `JOIN`. K propojení tabulek je k dispozici více `JOIN` dotazů a to například `RIGHT JOIN`, `LEFT JOIN`, `OUTER JOIN`, `INNER JOIN` a `CROSS JOIN`. Pro případ, kdy je potřeba sloučit dva rozdílné `SELECT` dotazy, použije se dotaz UNION, kdy se vybraná data zobrazí za sebou.

Jako poslední dotazy pro manipulaci s daty bych zmínil dotaz pro mazání záznamů z tabulky, `DELETE FROM nazev_tabulky WHERE podminka;`. Pomocí kterého se odstraní záznam splňující danou podmínku. Záznam tímto dotazem se ale odstraní pouze pokud neobsahuje žádná návazná data.

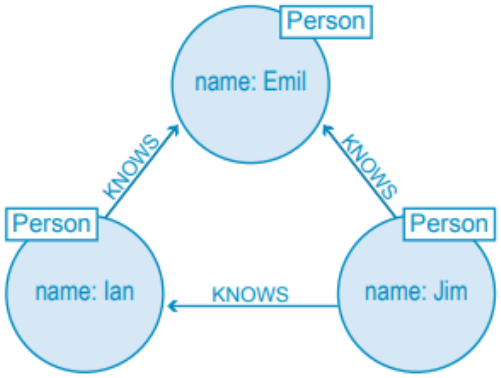
2.3 Grafová databáze

Jelikož zadání mé práce navazuje na už existující Diplomovou práci, kde výsledkem je nově vzniklá Neo4j databáze s genealogickými daty. Bylo potřeba nastudovat i princip fungování grafové databáze Neo4j, tak aby z ní mohl být prováděn převod do nově vzniklé MySQL databáze.

2.3.1 Neo4j

Neo4j je grafová databáze, která je k dispozici jako open-source. Jedná se o NoSQL nativní grafovou databázi, která je implementována v programovacím jazyce JAVA. Pro dotazování nad daty se používá primárně dotazovací jazyk Cypher, který je podobný jazyku SQL, akorát optimalizovaný pro grafy. S grafovou databází Neo4j je možně pracovat i v jiných programovacích jazycích, které mají potřebné knihovny. Kromě programovacího jazyku Java, to jsou například jazyky JavaScript a Python. Velká výhoda u grafové databáze Neo4j je v tom, že její transakce mají ACID vlastnosti a nejenom díky této vlastnosti jí používají pro své podnikání velké společnosti.

V grafové databázi Neo4j se místo tabulek používají pro reprezentaci dat uzly a jejich vlastnosti 2.4, které se mezi sebou spojují hrany a dohromady tvoří právě graf. Samotný graf této databáze je možné si představit jako když se nakreslí graf na papír. Narozdíl od relační databáze a používání různých spojovacích dotazů tabulek, které je poměrně zdlouhavé a náročné, grafová databáze se soustředí obzvlášť na vztahy a procházení mezi nimi [2, 5].

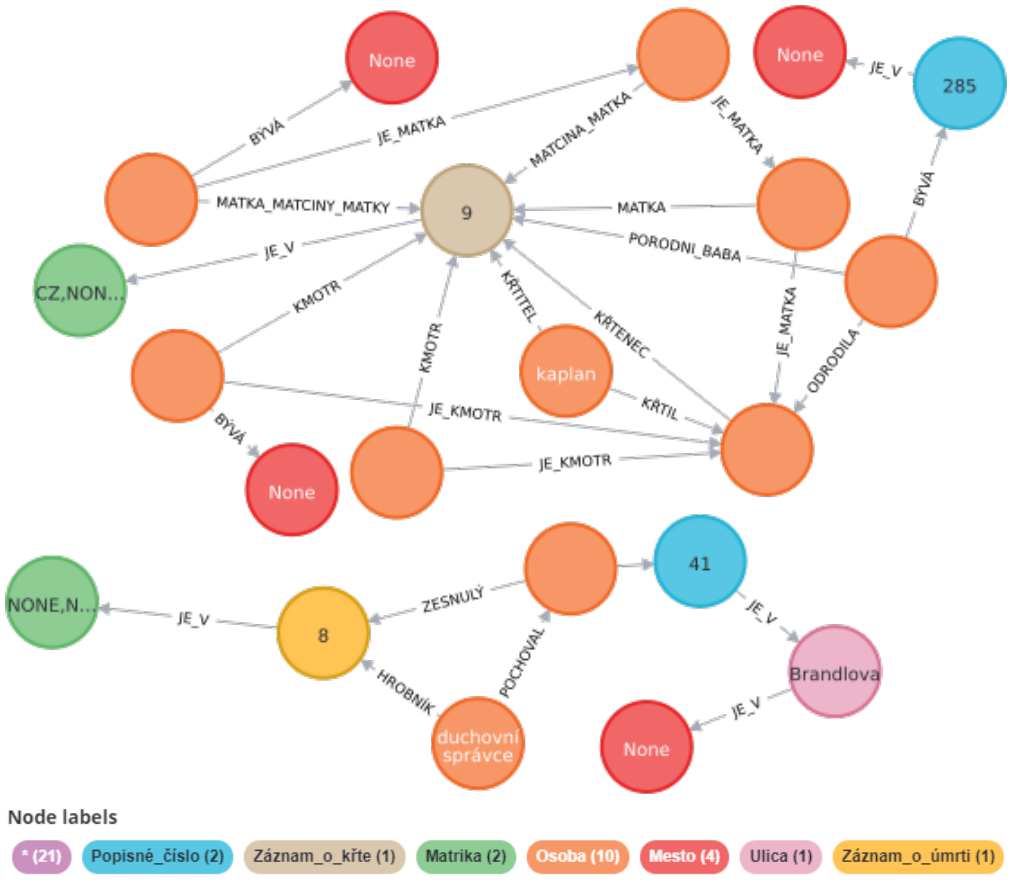


Obrázek 2.4: Ukázka grafu databáze Neo4j. Převzato z [2].

Kapitola 3

Návrh řešení

Po nastudování potřebných informací bylo potřeba v první řadě vytvořit návrh MySQL databáze pro ukládání genealogických dat, díky kterému bude moct být provedená následná implementace databáze. Návrh databáze vychází z řešení diplomové práce Lucie Tušimové z roku 2020 3.1, na kterou tato práce navazuje. Přesněji řečeno se jedná o návrh MySQL databáze, která bude odpovídat grafové databázi Neo4j z již zmiňované diplomové práce. Společně s návrhem databáze bude potřeba navrhnou případné skripty pro přesun dat mezi grafovou databází a relační databází, ale i naopak.



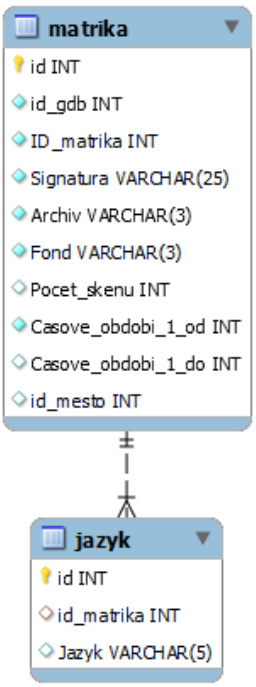
Obrázek 3.1: Příklad z původní Neo4j databáze

3.1 Návrh MySQL databáze

Pro provedení návrhu databáze pomocí konceptuálního modelování, transformace modelu na tabulky relační databáze a jejich normalizaci jsem vytvořil následující databázi. Názvy sloupců tabulek z velké části navazují na názvy vlastností uzlů grafové databáze, zbylé sloupce jsou pojmenovány tak, aby jejich název odpovídal tomu, co se v nich zaznamenává. Pro přehlednost přesouvaných dat se v každé tabulce nachází sloupec `id_gdb`, který obsahuje hodnotu jednoznačného identifikátoru původního uzlu. Samotný návrh databáze jsem pro přehlednost rozdělil na jednotlivé části, které zde představím a stručně popíši. Návrh celé databáze je součástí přílohy B.1.

3.1.1 Matrika

První část, kterou bych zde popsal, je hlavní tabulka `matrika` 3.2 sloužící k evidenci matrik. V této tabulce se budou ukládat základní data jako označení matriky signaturou, v jakém archívu se matrika nachází, fond matriky, v případě zaznamenání, kolik má daná matrika naskenovaných stran, z jakého období pochází ve formě intervalů let od, do. Do tabulky se také přenáší data `ID_matrika`, která jsou evidovaná v původní grafové databázi. Jelikož jedna matrika může obsahovat více jazykových forem, je potřeba navrhnout další pomocnou tabulku, zde tabulku `jazyk`, kam se budou tyto jazykové formy evidovat a zároveň odkazovat k tabulce `matrika`. Kromě zmíněné pomocné tabulky, pro evidenci více jazyků k jedné matrice, je potřeba navrhnout další pomocné tabulky, které budou zaručovat evidenci obcí, kde se matrika nachází. Tyto pomocné tabulky zaručí, že k jedné matrice může být evidováno více obcí. Popisu těchto tabulek se ale budu věnovat až v části 3.1.4, týkající se evidenci míst.



Obrázek 3.2: Návrh tabulek pro evidenci matriky.

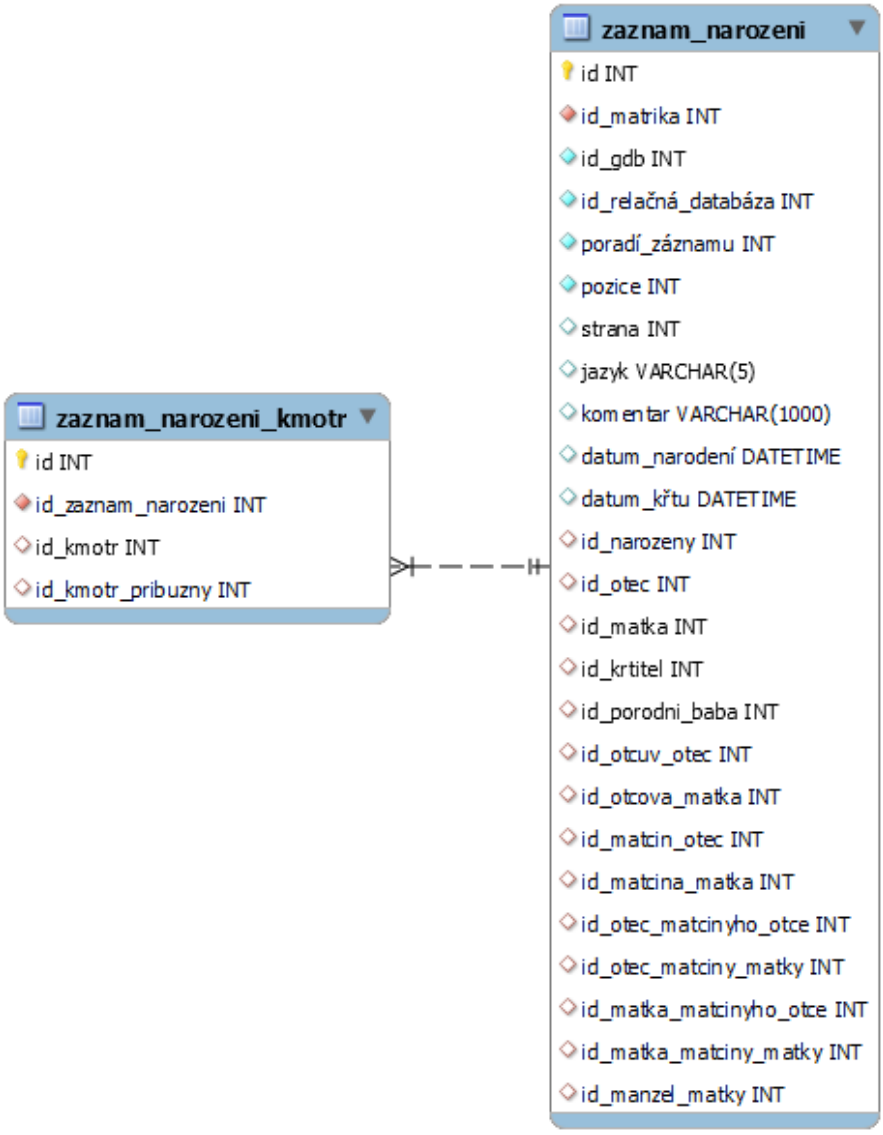
3.1.2 Matriční záznamy

Pro evidenci osob v matrikách slouží jednotlivé matriční záznamy. Tyto matriční záznamy se rozdělují do tří skupin pro evidenci narození, manželského sňatku a úmrtí. Do zde navržených tabulek se budou převádět data z uzlů `Záznam_o_křte`, `Záznam_o_svatbe` a `Záznam_o_úmrti` a jejich hrany propojení s uzly osob. Pro tyto účely se u každého druhu matričního záznamu evidují základní informace týkající se pořadí záznamu, jeho pozice a strany kde se nachází. Dále se zde eviduje jazyk zápisu a případný komentář. V matričních záznamech je potřeba také navrhnout sloupec pro evidenci jednoznačného identifikátoru k původní relační databázi, odkud se přesouvají data do grafové databáze Neo4j. Jelikož tabulky matričních záznamů navazují na tabulky matrik, je potřeba mít k dispozici sloupec, který bude formou cizího klíče odkazovat právě na návaznou tabulku matriky. Zbylá většina sloupců bude také formou cizích klíčů odkazovat na jednotlivé záznamy v tabulce `osoba` 3.1.3, která bude popsána ve zmíněné části.

Záznam narození

Tato tabulka 3.3, která slouží k evidenci nově narozených osob. Kromě již zmíněných sloupců pro základní evidenci matričních záznamů bude zde potřeba mít k dispozici sloupec pro evidenci data narození a to sloupce `datum_narodení`. Zbylé sloupce, jak už bylo zmíněno, budou formou cizího klíče odkazovat na záznamy v tabulce osob, čím se vlastně nastaví veškeré vztahy mezi osobami a matričním záznamem. Prvním zmíněným sloupcem je sloupec `id_narozeny`, který k matričnímu záznamu přiřadí narozenou osobu. Další dva sloupce se týkají rodičů, kdy pomocí sloupce `id_otec` je evidován otec narozené osoby a pomocí sloupce `id_matka` jeho matka. Po těchto poměrně jasných druhů osob k evidenci záznamu narození, je možné evidovat další návazné osoby a to křtitele v podobě `id\_krtitel`, porodní bábu v podobě `id_porodni_baba`. Dalšími sloupci pro evidenci návazných osob jsou sloupce určující prarodiče narozeného z otcovy strany a prarodiče i praprarodiče narozeného z matčiny strany. Posledním zde navrženým sloupcem je sloupec `id_manzel_matky` pro případnou evidenci nevlastního otce.

K záznamu narození se ještě eviduje kmotr a případně jeho příbuzný, ale protože může nastat situace, že k jednomu záznamu narození existuje více kmotrů a každý má svého příbuzného, je navržena pomocná tabulka `zaznam_narozeni_kmotr`.



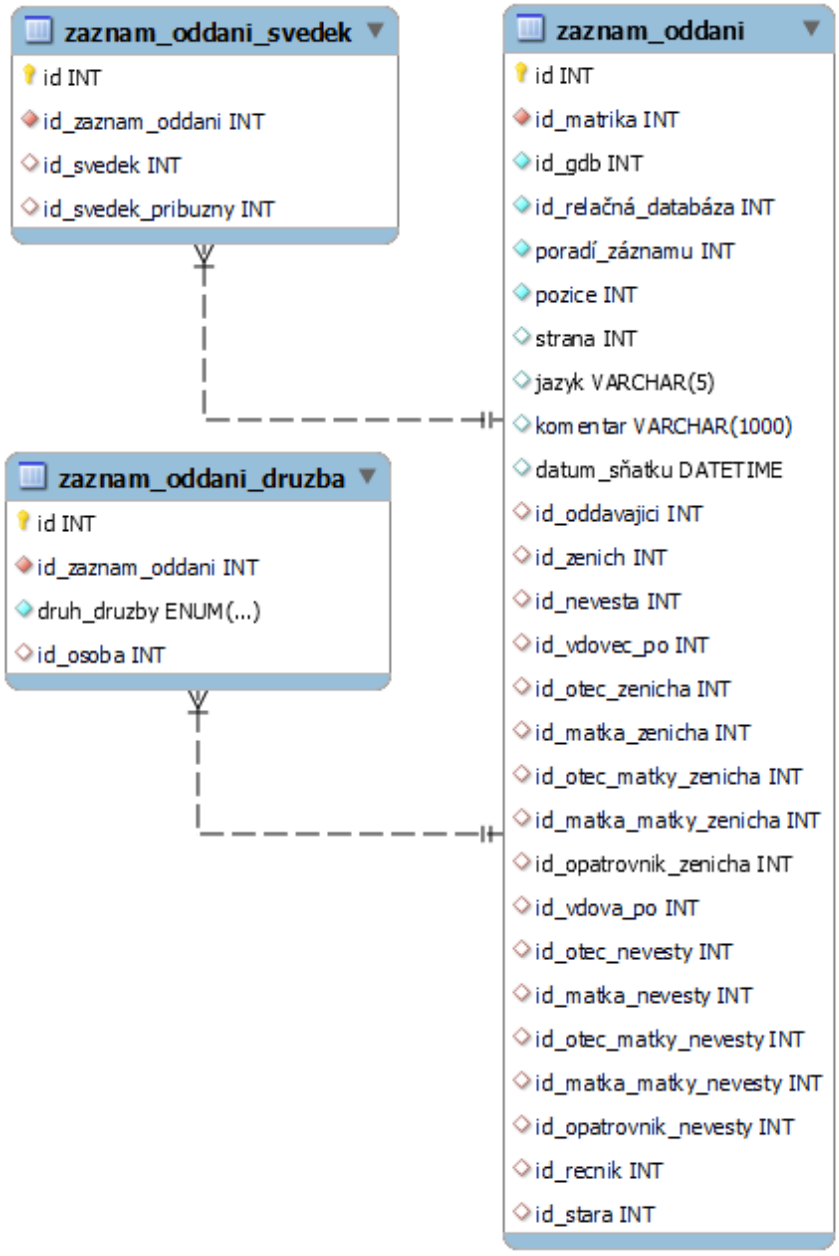
Obrázek 3.3: Návrh tabulek pro evidenci záznamů narození.

Záznam oddání

V pořadí druhá tabulka pro ukládání matričních záznamů je tabulka evidující záznamy oddání 3.4. K zaznamenání data sňatku je navrhnutý sloupec `datum_sňatku`. Hlavními účastníky jsou zde většinou ženich s nevěstou, kteří se zaznamenávají pomocí odkazů `id_zenich` a `id_nevesta`. Tak aby byli ženich s nevěstou oddáni, je potřeba mít osobu, která by měla zaručit právoplatnost manželského sňatku, pro tyto účely se zde eviduje oddávající ve sloupci `id_oddavajici`. Jak ženich, tak i nevěsta mohou být před sňatkem ovdovění. Také na tyto případy je zde myšleno a pro evidenci zesnulé manželky ženicha slouží sloupec `id_vdovec_po` a pro zesnulého manžela nevěsty sloupec `id_vdova_po`. Dalšími osobami, kteří se evidují u záznamu oddání, jsou rodinní příslušníci novomanželů, pro které jsou navrhnuty patřičné sloupce. Konkrétně se jedná o rodiče a prarodiče z matčiny strany jak

u ženicha, tak i u nevěsty. Jelikož nevěsta i ženich mohou mít přiřazeni svého opatrovníka, tak i pro tyto případy jsou navrhnuty vlastní sloupce k jejich evidenci. Pro opatrovníka ženicha sloupec `id_opatrovnik_zenich` a pro nevěstu `id_opatrovnik_nevesta`. Posledními osobami evidovanými v tabulce `zaznam_oddani` je řečník ve sloupci `id_recnik` a stará ve sloupci `id_stara`.

K hlavní tabulce `zaznam_oddani` jsou navrženy ještě další dvě pomocné tabulky, které zaručují, že u typů osob v těchto tabulkách může být k jednomu záznamu oddání více. Jako první bych zmínil tabulku pro evidenci svědků. Jedná se o tabulku `zaznam_oddani_svedek` kam je evidován každý svědek daného sňatku, a to jak pro ženicha, tak i pro nevěstu ve sloupci `id_svedek`. Zároveň ke každému svědkovi může být evidovaný jeho příbuzný, a to ve sloupci `id_svedek_pribuzny`. Druhou pomocnou tabulkou je tabulka pro evidenci družeb a družiček. Evidence záznamů v této tabulce je navrhnutá tak, že je zde sloupec `id_osoba`, sloužící jako odkaz na osobu, která je družbou nebo družičkou daného matričního záznamu oddání. Pro rozeznání, jestli se jedná o družičku nebo družbu slouží sloupec `druh_druzby` s přednastavenými možnostmi výběru s hodnotami `'druzba'` a `'druzicka'`.



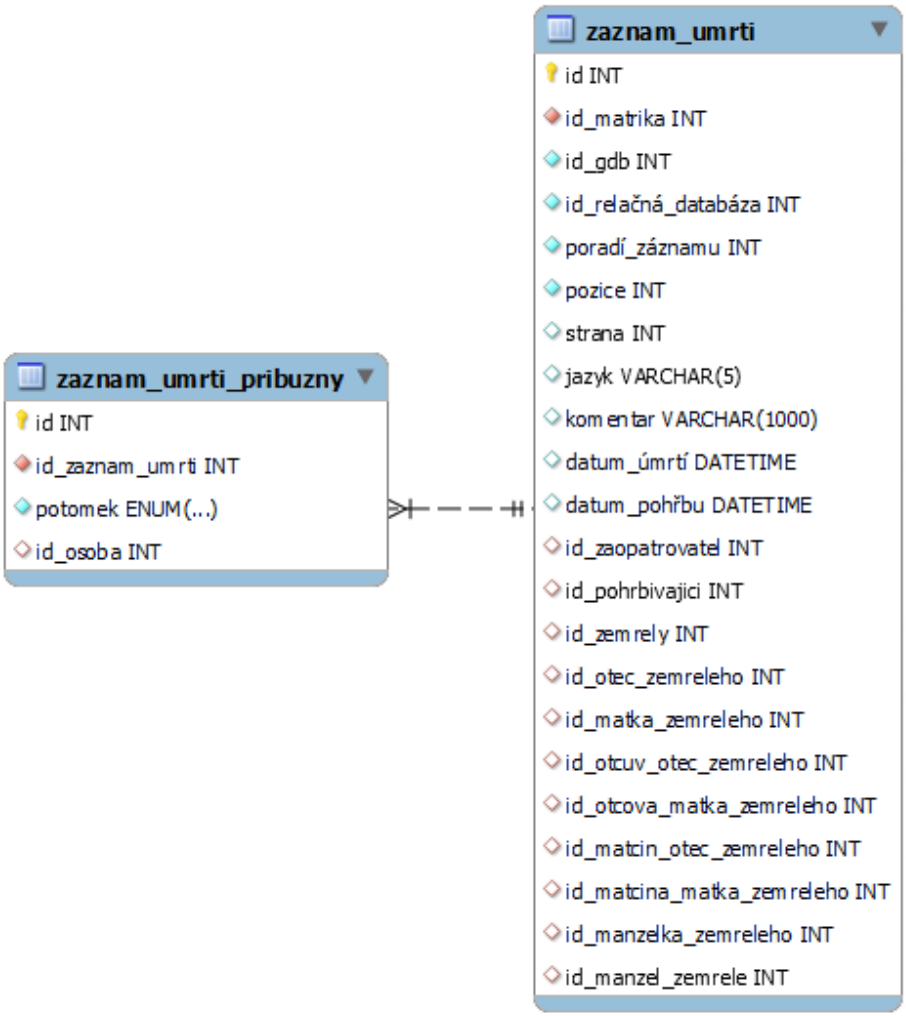
Obrázek 3.4: Návrh tabulek pro evidenci záznamů oddání.

Záznam úmrtí

Poslední ze tří tabulek matričních záznamů je návrh tabulky `zaznam_umrti` 3.5, kam se budou evidovat, jak už název napovídá, jednotlivé záznamy ohledně úmrtí osob. Hlavní osoba, která je evidována matričním záznamu úmrtí je ta, která je pochována. Tato osoba se zde zaznamenává ve sloupci `id_zemrely` odkazem formou cizího klíče na tabulku osoby. Dalšími osobami evidovanými v popisovaném návrhu matričního záznamu je osoba zaopatřovatele ve sloupci `id_zaopatrovatel` a osoba pohřbívajícího ve sloupci `id_pohrbivajici`. Jako u všech předchozích matričních záznamů, tak i zde se zaznamenává přesné datum

uskutečnění dané události, zde konkrétně data úmrtí ve sloupci `datum_umrtí`. Další návrh evidence osob k matričnímu záznamu úmrtí je také podobný jako v předchozích matričních záznamech, a to u evidence rodinných příslušníků zesnulé osoby. Konkrétně se zde evidují případní rodiče a prarodiče zesnulé osoby. Zbývajícími sloupci, navrženými v této tabulce, jsou `id_manzelka_zemreleho` a `id_manzel_zemrele` k evidenci manželky nebo manžela zesnulé osoby.

U matričních záznamů úmrtí se také evidují případní potomci zesnulého nebo jeho příbuzní. Jelikož těchto osob může být k jednomu zesnulému více, je k této tabulce matričních záznamů navrhnutá pomocná tabulka evidující tyto osoby. Evidence je zde navrhnutá tak, že do sloupce `id_osoba` se vloží odkaz na danou osobu a pomocí výběrového sloupce `potomek` se vybere z možností `'syn'`, `'dcera'` a `'pribuzny'`, podle toho o jaký vztah k zesnulé osobě se jedná.



Obrázek 3.5: Návrh tabulek pro evidenci záznamů úmrtí.

3.1.3 Osoba

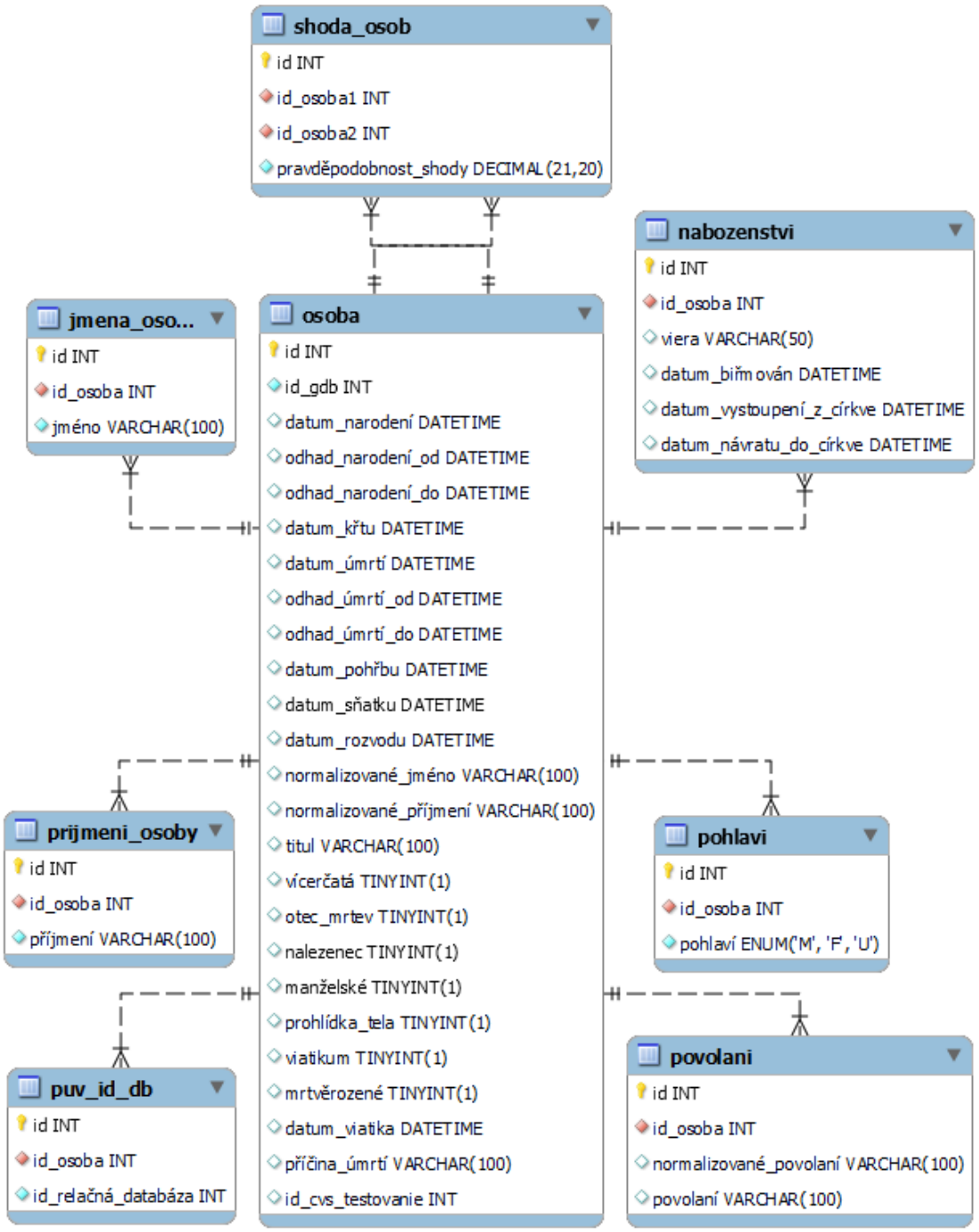
Po matrikách je potřeba mít samostatnou tabulku pro evidenci osob 3.6, která, jak už bylo zmíněno, je propojená s tabulkami všech tří tabulek matričních záznamů. U každého záznamu osoby v této tabulce jsou evidovány následující data, pro které jsou vytvořeny podobně pojmenované sloupce. Datum narození, odhad narození od a odhad narození do určující interval, ve kterém se daná osoba mohla narodit, pokud její přesné datum narození není známé, datum křtu, datum viatiky, datum sňatku, datum rozvodu, datum úmrtí a odhad úmrtí od a odhad úmrtí do, které určují v případě neznámého data úmrtí, interval ve kterém došlo k úmrtí dané osoby. Může se zdát, že sloupce pro evidenci data narození, sňatku a úmrtí jsou zde zbytečné, právě proto že se tato data evidují přímo v matričních záznamech. Není tomu tak, tyto sloupce jsou zde navrženy záměrně, a to pro případ, že k dané osobě není evidovaný nějaký ze tří matričních záznamů, ale máme k dispozici datum dané události.

Jelikož jeden uzel pro osobu v grafové databázi může zároveň obsahovat více jmen nebo příjmení, jsou součástí tabulky sloupce `normalizované_jméno` pro zaznamenávání normalizovaného jména a `normalizované_příjmení` pro zaznamenání normalizovaného příjmení, které slouží právě pro jedno jméno a příjmení, které jsou zároveň vyhodnoceny jako ty správné. Pro evidenci případných titulů je navrhnut sloupec `titul`, který bude obsahovat veškeré tituly dané osoby. V tabulce `osoba` jsou dále navrhnuty následující stavové sloupce. Sloupec `vícerčatá` určující, jestli daná osoba je součástí narozených vícerčat, sloupec `otec_mrtev`, který se eviduje v případě, má-li osoba zesnulého otce. `nalezenec` je další ze skupiny stavových sloupců, tento je konkrétně navržen pro evidenci toho, jestli se nezná původ dané osoby a byla nalezená. Stavový sloupec `manželské` je navržen pro evidenci, jestli se jedná nebo nejedná o nemanželské dítě. Dalšími sloupci jsou `prohlídka_tela`, určující, jestli u dané osoby byla provedena prohlídka těla, `viatikum` určující, jestli u dané osoby bylo viatikum provedeno a poslední stavový sloupec `mrtvěrozené` určující, jesli se jedná o mrtvorozené dítě.

V poslední řadě je u osoby evidována příčina úmrtí a jednoznačný identifikátor testovacího csv souboru související s testováním v původní grafové databázi Neo4j. K osobám je potřeba dále evidovat data ohledně pohlaví, náboženství, zmiňovaných více jmen nebo příjmení, jednoznačný identifikátor z původní relační databáze, povolání a případnou pravděpodobnost shody s jinou osobou. K těmto zmíněným datům je zapotřebí vytvořit další pomocné návazné tabulky, kdy všechny tyto pomocné tabulky jsou navrhnuty tak, že pro každou osobu může být evidováno více záznamů z jedné tabulky. K tomuto bude sloužit sloupec `id_osoba`, která bude formou cizího klíče odkazovat na nadřazenou tabulku osob.

Jména, příjmení a původní ID osob

Jak už jsem zmiňoval v části pro popis nadřazené tabulky osoby, tak pro jeden uzel osoby v grafové databázi může existovat více záznamů v původní relační databázi, tím pádem může být součástí jednoho uzlu více jmen. Právě tato jména se zaznamenávají do tabulky `jmena_osoby` 3.6. Na podobném principu jsou navrhnuty ještě další dvě tabulky, a to tabulka `prijmeni_osoby` 3.6 pro evidenci příjmení osoby a tabulka `puv_id_db` 3.6 pro evidenci jednoznačných identifikátorů osoby z původní relační databáze.



Obrázek 3.6: Návrh tabulky osoby a k ní návazných tabulek.

Pohlaví osob

Další doplňující tabulkou je tabulka `pohlavi` 3.6. Pro výběr pohlaví je zde navrhnutý výběrový sloupec s možnostmi výběru mužského, ženského nebo jiného pohlaví.

Náboženství

Součástí pomocných tabulek k tabulce osoby je tabulka `nabozenstvi` 3.6, která je navržená pro evidenci názvu náboženství a případných datů s ním spojených. Mezi zmíněnými daty je datum biřmování, datum vystoupení z církve a datum návratu do církve.

Povolání osob

Další pomocná tabulka, rozšiřující data k záznamům osob, je tabulka `povolani` 3.6, obsahující navrhnutý sloupec `povolani` evidující originální název povolání ze kterého nemusí jít vždy poznat, o přesně které povolání se jedná, a to kvůli tomu, že je povolání zapsáno cizím jazykem nebo se jedná o nějaké historické povolání. Aby bylo povolání zapsáno i ve srozumitelné podobě, je zde navrhnutý sloupec `normalizované_povolaní` pro evidenci normalizovaného názvu povolání.

Shoda osob

Poslední ze skupiny pomocných tabulek k nadřazené tabulky osob je tabulka `shoda_osob` 3.6. Na rozdíl od všech předchozích tabulek obsahuje dva sloupce odkazující na záznamy z tabulky osob, konkrétně se jedná o sloupce `id_osoba1` a `id_osoba2`. Dalším sloupcem této tabulky je `pravděpodobnost_shody` obsahující desetinné číslo s pravděpodobnostní shodou mezi odkázanými osobami.

3.1.4 Evidence místa

Poslední skupinou navrhnutých tabulek jsou tabulky pro evidenci místa. Mezi skupinu těchto tabulek patří tabulky pro evidenci názvů ulic, čísel popisných, měst a další pomocná a vazební tabulka místa.

Město

Jako první bych představil návrh tabulky `mesto` B.1, kde se kromě originálního názvu města ve sloupci `nazov_mesta` eviduje i jeho normalizovaný název. Dalším navrhnutým sloupcem je `is_mesto`, obsahující identifikátor záznamu města z původní grafové databáze.
K této tabulce bylo potřeba navrhnout pomocnou tabulku `mesto_gps` B.1 pro evidenci GPS souřadnic. Kromě sloupce `id_mesto`, který formou cizího klíče zaručuje propojení s nadřazenou tabulkou měst, je zde sloupec `gps_souradnice` evidující číselnou hodnotu dané souřadnice a sloupec `poradi`, zaručující pořadí zaevidované GPS souřadnice v souřadném systému GPS souřadnic.

Ulice

Jako další bych představil návrh tabulky `ulice` B.1, kde se eviduje samotný název ulice a do sloupce `id_ulice` identifikátor ulice z grafové databáze. V tabulce je dále navržený sloupec `id_mesto`, který je formou cizího klíče a zajistí propojeni s tabulkou `mesto`.

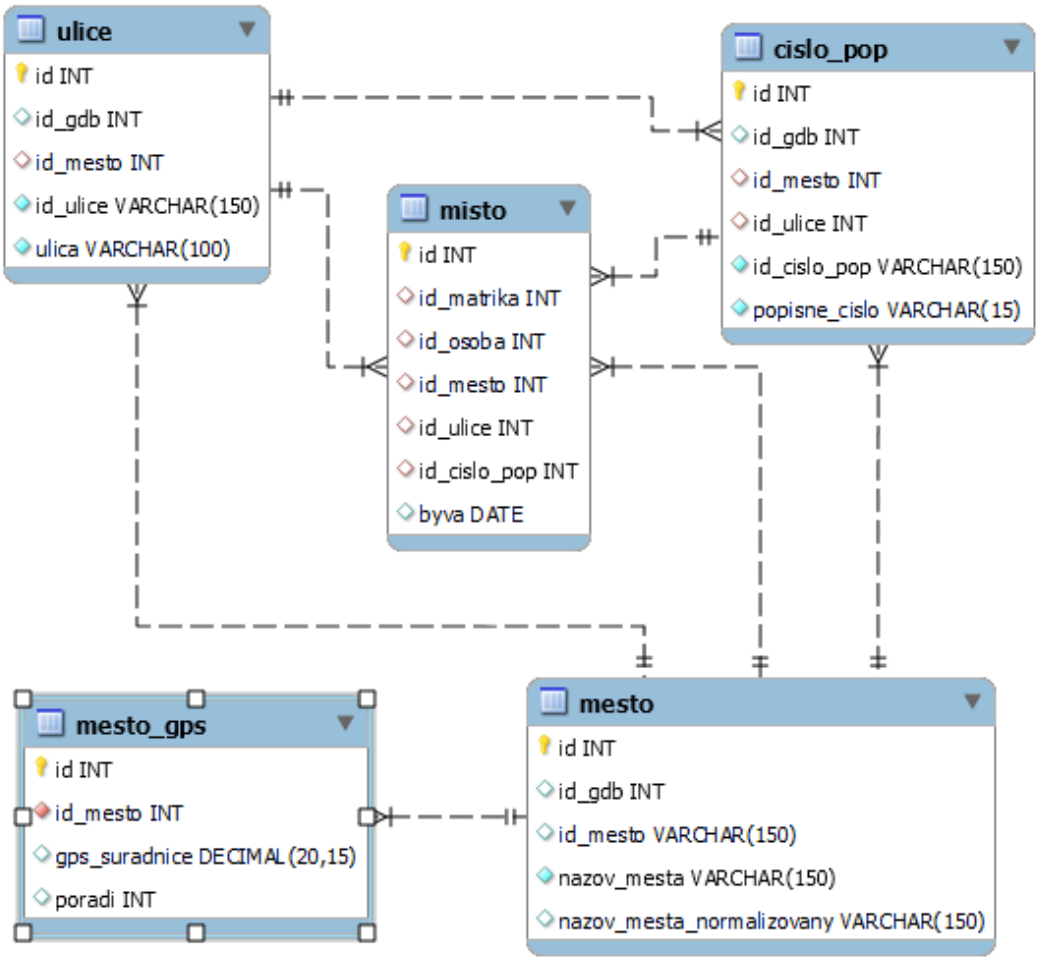
Číslo popisné

Ve stylu všech předchozích tabulek této části je i návrh tabulky `cislo_pop` B.1, kde se eviduje číslo popisné a do sloupce `id_cislo_pop` identifikátor čísla popisného z grafové databáze. Stejně jako v předchozí tabulce ulic, tak i zde je navržený sloupec `id_mesto`,

který je formou cizího klíče a zajistí propojeni s tabulkou `mesto`. Podobně je navrhnutý sloupec `id_ulice`, který zajistí propojení s tabulkou ulic.

Místo

Poslední tabulkou v této části je vazební tabulka `misto` B.1, která slouží k propojení tabulek z této části míst 3.1.4 a tabulek matrik 3.1.1 a tabulek osob 3.1.3. Pro propojení s tabulkami měst slouží sloupec `id_mesto`, s tabulkami ulic sloupec `id_ulice`, s tabulkami čísel popisných sloupcem `id_cislo_pop`, s tabulkami osob sloupec `id_osoba` a s tabulkami matrik sloupec `id_matrika`. Posledním sloupcem je `byva`, kam se eviduje datum, které se zaznamenává při propojení tabulek míst s tabulkou osoby.



Obrázek 3.7: Návrh tabulky místa.

3.2 Návrh skriptů pro převod

Po vytvoření návrhu relační databáze, která bude sloužit ke kopii již existujících genealogických dat z grafové databáze, bude potřeba navrhnout samotný princip přenosu dat, a to jak z grafové databáze do relační, tak i obráceně. Právě možnému přenosu dat mezi těmito databázemi bych se chtěl v této části věnovat a navrhnout možné řešení.

3.2.1 Převod z Neo4j do MySQL

Jako první bych představil popis návrhu převádění dat z grafové databáze Neo4j do relační MySQL databáze 3.8. Pro zajištění samotného převodu bude v první řadě potřeba se připojit k existující Neo4j databázi a připojit se k nově založené MySQL databázi i s vytvořenými tabulkami, které byly navrženy v předchozí části. Po úspěšném připojení k obou databázím přijde na řadu dotazování se a následný převod získaných dat. Dotazováním bude potřeba postupně procházet jednotlivé uzly každého typu a následně jejich data přenášet do konkrétních tabulek vytvořené relační databáze.

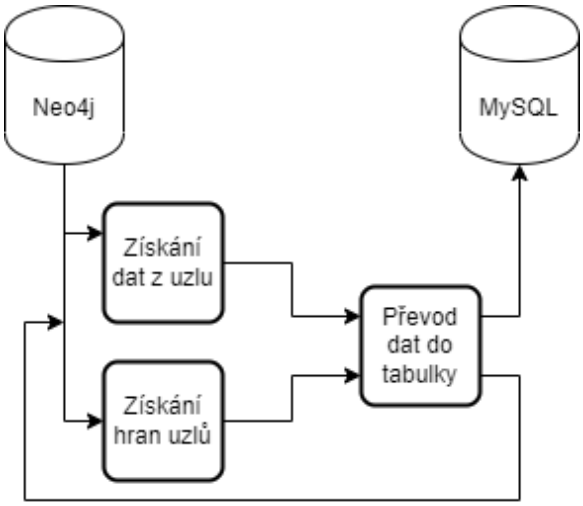
Začne se postupným dohledáváním dat z uzlů měst, ulic a čísel popisných. Tato data ze zmíněných uzlů se přesunou do patřičných tabulek v relační databázi. Data měst do tabulky `mesto`, ulice do tabulky `ulice` a číslo popisné do tabulky `cislo_pop`.

Po dohledání a přesunutí všech míst se přejde na dotazování k uzlům matrik, odkud se budou přenášet data do tabulky `matriky`. Při každém zaznamenání matriky se zároveň zaeviduje i místo, přesněji se zde jedná pouze o název obce, případně více obcí. Tato evidence místa se provede způsobem, že se v případě neexistujícího názvu obce založí nový záznam s daným názvem obce v tabulce `mesto`. Následně se propojí obec s matrikou pomocí vazební tabulky `misto`, kde se vyplní odkaz na záznam daného města a dané matriky, zbylé sloupce odkazující na jiné tabulky zůstanou v tomto případě prázdné. V případě, že obec, kde se daná matrika nachází, už existuje, nevytváří se další duplicitní záznam s tímto názvem obce, ale pouze se vytvoří záznam do vazební tabulky, která propojí dané záznamy obce s matrikou.

Po přesunu dat z uzlů týkajících se matrik se přejde k přesouvání dat ohledně jednotlivých osob. U osob se začne dotazováním k jejich uzlům a následný přesun dat do jejich tabulky `osoba`. Jelikož uzel osoby může obsahovat více jmen, příjmení nebo id z původní relační databáze, provede se jejich přesun do navržených pomocných tabulek `jmena_osoby`, `prijmeni_osoby` nebo `puv_id_db`, které jsou návazné k tabulce `osoba`. Dalšími záznamy jsou data ohledně pohlaví, povolání a náboženství, budou přesouvány do pomocných tabulek `pohlavi`, `povolani` a `nabozenstvi`. Každý uzel osoby je hranou propojen s místem pobývání, které může být znázorněno městem, ulicí nebo i číslem popisným. K propojení těchto záznamů slouží vazební tabulka `mista`, kam se odkáže na místo pobývání osoby. V případě, že se jedná o ulici, vloží se odkaz do sloupce `id_ulice`, pokud se jedná o číslo popisné, vloží se odkaz do sloupce `id_cis_pop`, a pokud se jedná o název obce, vloží se odkaz do sloupce `id_mesto`. Společně s jedním vyplněným sloupcem odkazující na nějaký druh adresy je potřeba do tabulky doplnit i odkaz na danou osobu do sloupce `id_osoba`, čím se zajistí konečné propojení.

V této fázi přesunu, kdy je hotový přesun dat týkajících se matrik, dat všech osob, míst nalezení matrik a pobývání osob z grafové databáze, je potřeba převést data z matričních záznamů, které zaručují vztahy mezi jednotlivými osobami uložených v určitých matrikách. Jako první se provede přenos dat matričních záznamů narození. Jedná se o přiřazení matriky a osob k matričnímu záznamu, které jsou součástí daného záznamu a přenesení přesného data narození. Podobným způsobem se přenesou matriční záznamy oddání a matriční záznamy úmrtí, s tím rozdílem, že u oddání se eviduje pouze přesné datum oddání a u úmrtí přesné datum úmrtí.

Ve většině zmíněných případech se po dohledávání dotazy v grafové databázi, přenáší záznamy do relační databáze tak, že se vytvoří nový záznam. Jsou ale i případy, kdy záznam už existuje, jen je potřeba k danému záznamu přidat nebo upravit nějaká data. Pro tyto účely je potřeba si ukládat do proměnných určité identifikátory, pomocí kterých můžeme zaručit správně provedenou úpravu záznamů.



Obrázek 3.8: Návrh přenosu z Neo4j do MySQL databáze.

3.2.2 Převod z MySQL do Neo4j

Tato část není přímo součástí zadání mé bakalářské práce, ale po konzultaci s vedoucím mé práce jsme se domluvili, že ji tam přidám.

Princip tohoto převodu bude podobný jako u předchozího převodu 3.9. Stejně jako v předchozí části, tak i zde bude potřeba být v první fázi úspěšně připojený k obou databázím, mezi kterými má dojít k přesunu dat. Na rozdíl od předchozí části se zde dohledávají data pomocí dotazů v MySQL databázi a ukládají pomocí dotazů Cypher do Neo4j databáze.

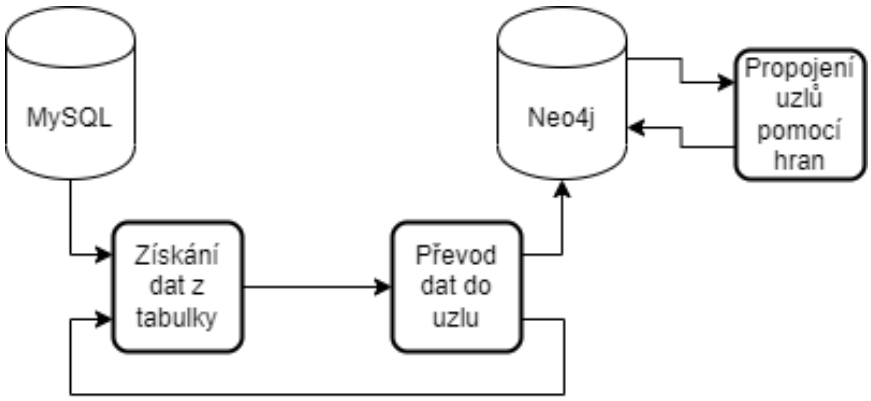
U tohoto převodu se začne převodem dat matrik, kdy se dotazem nad tabulkou `matrika` získají potřebná data, která se postupně přenesou do jednotlivých uzlů typu `Matrika` v grafové databázi.

Jakmile jsou převedené všechny záznamy matrik, začnou se postupně převádět data osob, a to z tabulky `osoba` do uzlů typu `Osoba`. Součástí tohoto převodu dat osob se převedou veškerá data z tabulky a zbylé vlastnosti se doplní až později.

Ještě než dojde k převodu samotných matričních záznamů, je potřeba převést data týkající se míst určení. Konkrétně se jedná o převod záznamů z tabulek `mesto`, `ulice` a `cislo_pop` do uzlů typů `Mesto`, `Ulica` a `Popisné_číslo`. S převodem samotných záznamů MySQL databáze bude potřeba ještě zaručit převod existujících vazeb mezi zmíněnými tabulkami místa určení a také převod vazeb mezi záznamy tabulek míst určení a záznamy v tabulce `osoba`.

Nyní se dostáváme k zásadnímu převodu matričních záznamů. Zde se pro všechny tři typy matričních záznamů provede prvně převod samozného záznamu dané tabulky do patřičného typu uzlu v grafové databázi. Poté dojde k převodu vazeb mezi matričním záznamem a osobami a matričním záznamem a matrikou, čím dostaneme výsledný vzhled schematu uzlů grafové databáze.

Jelikož ještě nejsou převedena data z pomocných tabulek k tabulce `matrika` a `osoba`. Provede se tento převod nyní



Obrázek 3.9: Návrh přenosu z MySQL do Neo4j databáze.

Kapitola 4

Implementace a testování

Po teoretické části návrhu bych zde chtěl představit i praktickou část samotné implementace. Jedná se o skripty sloužící pro práci jak s Neo4j, tak i s MySQL databází. Pro možnou práci s oběma databázemi zároveň jsou tyto skripty implementovány v programovacím jazyce Python. Přesněji se jedná o čtyři různé skripty `dictionary.py` obsahující slovníky pomocí kterých dochází k propojování názvů vlastností a hran uzlů s názvy sloupců tabulek, `mysqlDatabase.py` zahrnující obsluhu MySQL databáze, `graphDatabase.py` pro obsluhu grafové databáze Neo4j a `main.py`, který slouží pro propojení předchozích komunikací a tím zajišťující přesun dat mezi nimi.

Společně s těmito programy je potřeba mít ještě k dispozici soubor obsahující přihlašovací údaje k obou databázovým serverům. Tento soubor by měl obsahovat název HOSTNAME, číslo PORTU, uživatelské jméno, heslo a název databáze jak k MySQL, tak i k Neo4j serveru. V případě, je-li potřeba se připojit ke vzdálenému MySQL serveru jsou potřeba i přihlašovací údaje pro připojení se k SSH tunelu. V tomto případě musí externí soubor obsahovat i název HOSTNAME, PORT, uživatelské jméno a heslo pro SSH připojení. V následujících podčástech bych chtěl podrobněji představit popis implementace jednotlivých zmiňovaných programů.

4.1 Implementace MySQL

Hlavní částí této práce bylo vytvořit samotnou MySQL databázi a především k tomuto slouží program `mysqlDatabase.py`. V tomto programu je importována knihovna `pymysql`, která slouží pro možnost připojení se k MySQL serveru a implementaci MySQL dotazů v jazyce Python a knihovna `sshtunnel` pro zajištění SSH tunelu k případnému připojení ke vzdálenému MySQL serveru. Tento program je implementován pomocí vytvořené třídy `MysqlDB`.

Na začátku tohoto programu dojde ke čtení z externího souboru obsahující přihlašovací údaje. Tyto přihlašovací údaje se rozdělí do slovníku odkud je možné je následně používat a přihlásit se pomocí nich k databázovému serveru.

Jak už bylo zmíněno, tak pro případné připojení ke vzdálenému MySQL serveru je potřeba se nejdříve připojit k SSH tunelu, k tomuto se použije sekvence následujících příkazů:

```
tunnel = SSHTunnelForwarder((ssh_hostname, ssh_port),
                            ssh_username=ssh_username,
                            ssh_password=ssh_password,
                            remote_bind_address=(sql_hostname, sql_port)
                            )
tunnel.start()
```

Po úspěšném připojení k SSH tunelu nebo v případě připojování se k lokálnímu serveru a tím pádem nevyužití SSH tunelu dojde pomocí příkazu

`pymysql.connect(hostname, port, user, password, database)`

k připojení k databázi MySQL serveru a následně dojde k vytvoření kurzoru pomocí příkazu `cursor = connection.cursor()`, díky kterému jsou prováděny veškeré SQL dotazy.

Program následně obsahuje funkce, které pomocí MySQL dotazů mažou a vytváří jednotlivé tabulky. Zmiňované mazání tabulek se provádí pokaždé před vytvářením tabulek, tak aby se v případě existence mohli tabulky korektně znovu založit. Protože se rovnou při tvorbě tabulek vytváří propojení tabulek pomocí cizích klíčů, musí se tabulky vytvářet v určitém pořadí, a to nejdříve tabulky `mesto`, `mesto_gps`, `ulice` a `cislo_pop`.

Po vytvoření tabulek sloužících k evidenci místa se vytváří tabulka `osoba`, která eviduje informace o osobách z genealogické databáze matrik. Jakmile je vytvořená tabulka `osoba`, přejde se na tvorbu tabulek, které jsou na tuto tabulku návazné. Jedná se o tabulku `povolani`, přiřazující povolání k dané osobě, tabulka `nabozenstvi`, evidující data týkající se náboženské víry, tabulka `pohlavi` určující pohlaví osoby. Jelikož jeden uzel osoby v databázi Neo4j může obsahovat více jmen, příjmení a id odkazující na původní záznam z relační databáze matrik, vytvoří se další tři tabulky pro evidenci právě těchto záznamů. Konkrétně tabulky `jmena_osoby`, `prijmeni_osoby` a `puv_id_db`. Poslední vytvořenou pomocnou tabulkou k tabulce `osoba`, je tabulka `shoda_osob`, kam se zaznamenává pravděpodobnostní shoda mezi osobami.

Jakmile jsou založené tabulky pro ukládání dat k osobám, začnou se zakládat tabulky týkající se samotných matrik. V první řádě se vytvoří samotná tabulka `matrika`, kam se budou ukládat data k jednotlivým matrikám. Následně se založí pomocná tabulka `jazyk` evidující jazyk zápisů k dané matrice. Právě teď se při zakládaní tabulek nacházím v bodě, kdy máme založené tabulky pro evidenci míst, osob a jejich pomocných tabulek a matrik a její pomocné tabulky. To znamená, že je možné založit vazební tabulku `misto` sloužící k propojení tabulek míst s matrikou nebo osobou.

Jako poslední se vytvoří jedny z nejdůležitějších tabulek, a to tabulky pro jednotlivé záznamy matrik, které zaručují nejen propojení osob s matričními záznamy, ale i nastavení vazeb mezi jednotlivými osobami. Jedná se o tabulky matričních záznamů `zaznam_narozeni`, `zaznam_oddani` a `zaznam_umrti` a k nim pomocných tabulek `zaznam_narozeni_kmotr`, `zaznam_oddani_svedek`, `zaznam_oddani_druzba` a `zaznam_umrti_pribuzny`.

Po funkci zajištující tvorbu tabulek se v tomto programu nachází funkce zajišťující volání MySQL dotazů. Jako první je funkce pro vytváření záznamů v MySQL tabulce, která má čtyři vstupní argumenty, a to příznak ignore, název tabulky, seznam s názvy sloupců a seznam s daty nově vytvořeného záznamu v tabulce. První argument této funkce v podobě pravdivostního příznaku slouží k tomu, jestli se bude volat dotaz `INSERT INTO` nebo `INSERT IGNORE INTO`, který zamezí vkládání duplicitních záznamů. Znamená to, že při volání funkce

```
insert_table_all(False, table_name, (column1, column2, column3),
                 (value1, value2, value3))
```

se vytvoří SQL dotaz

```
INSERT INTO table_name (column1, column2, column3)
VALUES(value1, value2, value3),
```

který v dané MySQL tabulce založí nový záznam.

Další funkce pro tvorbu MySQL dotazů slouží pro výpis dat z tabulky, přesněji se jedná o tvorbu SELECT dotazu. Zde se funkce volá s argumenty název tabulky, seznam s názvy sloupců a případně řetězec obsahující podmínku hledaných záznamů, hodnota hledané podmínky, název tabulky k možnému propojení, název sloupce, pomocí kterého dojde k propojení, jedné tabulky a druhé tabulky a typ propojení. Podmínka se zohledňuje pouze pokud je zadaná, to stejné platí i pro případ propojení tabulek. Uvedl bych zde jednoduchý případ v případě využití všech vstupních argumentů funkce.

```
select_table(table1, [table1.column1, table1.column2, table2.column3],
             table1.column2, value, table2, column2, column1)
```

Po zavolání funkce v této podobě se vygeneruje následující SQL dotaz.

```
SELECT table_name1.column1, table_name1.column2, table_name2.column3
FROM table_name1
JOIN table_name2 ON table_name2.column2 = table_name1.column1
WHERE table_name1.column2 = value,
```

Naopak tato funkce může sloužit i jen pro zobrazení hodnot určitých sloupců pro všechny záznamy jedné tabulky a v tom případě by se volala následující fuknce.

```
select_table(table_name, (column1, column2))
```

Následně se pomocí této funkce vygeneruje SQL dotaz.

```
SELECT column1, column2
FROM table_name
```

Jelikož v některých případech není možné poskládat potřebný `SELECT` dotaz. Jsou zde implementované další složitější dotazy, které se využívají zejména při testování a také naopak i jeden jednodušší dotaz, který slouží pouze pro výpis dané hodnoty, zde se konkrétně jedná o `only_select(column)`, kde právě argument column je daná hodnota. Výsledkem je vygenerovaný jednoduchý SQL dotaz `SELECT column`.

Kromě zakládání nových záznamů do tabulek a výpisu z nich je potřeba ještě mít k dispozici funkci pro úpravu záznamů v tabulkách. Pro tento účel slouží funkce `update_table`

obsahující podobné argumenty jako zmíněná funkce `insert_table`, s tím rozdílem, že argumenty pro podmínku jsou zde povinné a argumenty k propojení více tabulek zde nejsou k dispozici. Zmínil bych zde jednoduchou ukázku volání této funkce

```
update_table(table, (column1, column2), (value1, value2), column3, value3)
```

Výsledkem této funkce bude vyvolání následujícího SQL dotazu.

```
UPDATE table_name1
SET column1 = value1, column2 = value2
WHERE column3 = value3,
```

Mezi poslední funkce této třídy patří funkce sloužící k provedení SQL dotazu a funkce pro odpojení kurzoru a ukončení spojení s MySQL databází.

Kromě posledních zmíněných funkcí se při úspěšném ukončení ostatních funkcí provede stručný zápis výsledku do textového souboru. Zároveň u všech těchto funkcí dochází k zachycování výjimek a v případě zachycení výjimky se do textového souboru zaznamená chybová hláška, uzavře se spojení s textovým souborem a ukončí se spojení pomocí poslední zmíněné funkce a následně se zobrazí chybová hláška i v terminálu, kde došlo ke spuštění.

4.2 Implementace Neo4j

Po seznámení se s programem obsluhující MySQL databázi, je potřeba ještě popsat funkčnost programu pro komunikaci s grafovou databází Neo4j. Jsou zde importovány následující knihovny. Knihovna `neo4j`, díky které se umožní práce s Neo4j databází, knihovna `datetime` pro zpřístupnění funkcí pro práci s daty a je zde importovaný program `dictionary.py` 4.3. Také tento program je implementovaný pomocí vytvořené třídy, a to třídy `GraphDatabase`.

Stejně jako u předchozího programu, tak i zde dojde na začátku k otevření souboru s přihlašovacími údaji, které využije následující funkce pro připojení ke grafové databázi Neo4j.

```
neo4j.GraphDatabase.driver(uri=uri,
                           auth=(username, password),
                           database=database)
```

Argument `uri` u zmíněné funkce připojení je URI adresa ke grafové databázi. Tato URI adresa je poskládána ještě před zavoláním funkce, a to z HOSTNAME a čísla portu z externího souboru s přihlašovacími údaji určené pro Neo4j. `Auth` slouží pro zadání přihlašovacích údajů a database pro výběr databáze, ke které se chceme připojit. Tento program dále obsahuje funkce pro manipulaci s daty grafové databáze Neo4j.

První ze zmíněných funkcí pro manipulaci dat je funkce sloužící pro zobrazení všech vlastností daného uzlu.

```
get_all_record(node_name)
```

Výsledkem této funkce bude vyvolání následujícího cypher dotazu.

```
MATCH (r:node_name) RETURN id(r), r
```

Zbylé funkce zobrazující data z grafové databáze jsou podobného principu, proto bych zde podrobněji zmínil pouze jednu a zbylé jen stručně popsal. Podrobně bych popsal funkci `get_person_bind`, která obsahuje 3 vstupní argumenty, a to argument `node` pro zadání názvu uzlu z grafové databáze a argumenty `id_person` a `r` sloužící jako vyhledávací podmínky dotazu. Výsledkem této funkce je výpis jednoznačného identifikátoru zadaného uzlu, který v určitém vztahu s daným uzlem osoby. Uvedu zde příklad volání této funkce.

```
get_person_bind(node_name, value1, value2)
```

A následně výsledek vygenerovaného cypher dotazu.

```
MATCH (o:Osoba)-[r]->(z:node_name)
WHERE ID(o) = value1 and type(r) = 'value2'
RETURN ID(z)
```

Jak už jsem zmínil, další funkce pro výpis dat z grafové databáze si jsou hodně podobné. Mezi tyto funkce patří `get_record_bind(node, id_record)`, kde vstupními argumenty je název uzlu a hledaná hodnota tohoto uzlu, ke kterému vede hrana z jiného uzlu `Osoba`. Tato funkce následně vrací hodnoty všech vlastností uzlu `Osoba` a typ a datum hrany, kterou jsou tyto uzly spojeny.

K získání jednoznačného identifikátoru uzlu `Matrika` slouží další implementovaná funkce `get_registry_bind(node, id_record)`, kde argument `node` určuje název uzlu který je propojený hranou typu `JE_V` s uzlem `Matrika` a druhý argument `id_record` značí hodnotu jednoznačného identifikátoru uzlu daného argumentem `node`.

Podobná funkce `get_lives_bind(node, id_person)` se používá pro získání jednoznačného uzlu zadaným argumentem `node`, ke kterému vede hrana typu `BÝVÁ` z uzlu `Osoba` a právě druhý vstupní argument udává jednoznačný identifikátor uzlu `Osoba`.

Předposlední funkce, pro výpis dat z grafové databáze, je funkce `get_persons_bind()`, díky které se získají identifikátory obou uzlů, které jsou zde uzly `Osoba` a jsou propojeny hranou typu `POTENCIONALNÍ_ZHODA`, právě vlastnost tohoto uzlu `pravděpodobnost_shody` je třetí získaná hodnota z této funkce.

Jako poslední funkce spadající do skupiny získávání dat z grafové databáze je funkce `get_bind(node1, node2, id_a)`. První dva argumenty této funkce značí první a druhý uzel, kteréjsou spolu propojeny hranou a argument `id_a` obsahuje hodnotu jednoznačného identifikátoru prvního uzlu. Výsledkem této funkce je hodnota jednoznačného identifikátoru druhého uzlu zadaného vstupním argumentem.

Od funkcí pomocí kterých se získávají data z grafové databáze Neo4j jsme se dostali k funkcím zapisující do grafové databáze. Nachází se zde funkce pro tvorbu uzlů se zápisem jejich vlastností a funkce pro tvorbu hran s případným zápisem jejich vlastností.

Začal bych u funkcí pro tvorbu uzlů, které vždy obsahují argument `properties` a `values`. Druhý argument obsahuje hodnoty ve formě matice, z tohoto důvodu se vždy na začátku každé funkce začne pomocí cyklu procházet touto maticí a poté je k dispozici pro každý krok cyklu pouze jedno pole hodnot. Následně se z prvního argumentu a získaného pole

hodnot vytvoří slovník, díky kterému se poté zapisují hodnoty k určitým vlastnostem podle klíčů tohoto slovníku. Kromě všech vlastností uzlů, které jsou součástí původní databáze Neo4j, jsou součástí nové Neo4j databáze dvě další vlastnosti a to `id_gdb` určující jednoznačný identifikátor uzlu z původní Neo4j databáze a `id_mysql` obsahující jednoznačný identifikátor záznamu z tabulky MySQl databáze.

První z funkcí pro zakládání uzlů je `create_registry_nodes(properties, values)`, pomocí které se zakládají nové uzly `Matrika` se všemi vlastnostmi, tak jako v původní Neo4j databázi a to i se zadanými hodnotami. Podobná funkce je, která slouží pro zakládání uzlu `Osoba` a jeho vlastností je funkce `create_persons_nodes(properties, values)`. Dalšími funkcemi pro zakládání uzlů a jejich vlastností je funkce pro zakládání uzlů určující místo `create_place_nodes(node, properties, values)`, kde se argumentem `node` určuje, jestli se jedná o tvorbu uzlu `Mesto`, `Ulica` a `Popisné_číslo`. K zakládání jednotlivých uzlů matričních záznamů slouží funkce `create_record_nodes(node, properties, values)`, kdy argument `node` určuje, o jaký druh matričního záznamu se jedná. Poslední funkce týkající se zápisu do uzlů je funkce `add_node_property(node, properties, values)`, která do uzlu zadaného pomocí argumentu `node` přidá do vlastnosti z druhé hodnoty seznamu `properties`, který je zadán jako argument funkce jeho hodnotu z argument `values`.

Druhou skupinou funkcí sloužící k zakládání záznamů v grafové databázi jsou funkce vytvářející hrany mezi jednotlivými uzly. Mezi tyto funkce patří například vytváření hran `JE_V` pomocí `create_place_rel(node, properties, values)` a to mezi uzly míst `Mesto`, `Ulica` a `Popisné_číslo`, kdy výsledkem je jedna z následujících možností vytvořené hrany

```
Ulica-[r:JE_V]->Mesto,
Popisn_slo-[r:JE_V]->Ulica,
Popisn_slo-[r:JE_V]->Mesto.
```

K založení hrany `BÝVÁ` mezi uzlem `Osoba` a jedním ze tří typů uzlu určující místo slouží funkce `create_address_bind(rel_names, rel_values)`, zároveň se i zapíše vlastnost `date` této hrany, obsahující datum, kdy osoba pobývala na daném místě. K založení hran mezi uzly `Záznam_o_křet`, `Záznam_o_svatbe` nebo `Záznam_o_úmrti` a uzlem `Osoba` je vytvořena funkce `create_record_bind(node, rel_names, rel_values, rel_property=None)`, která zároveň do vytvořené hrany vloží vlastnost `date`, určující datum konání dané události. Tyto zmíněné hrany se tvoří v cyklu tak, aby se každý jeden matriční záznam zároveň propojil se všemi osobami, které do něj patří. Před zahájením tohoto cyklu se ale ještě propojí matriční záznam hranou `JE_V` k uzlu `Matrika`.

Jelikož kromě spojení mezi osobou a matričním záznamem je potřeba mít hrany i mezi jednotlivými uzly `Osoba`, bylo potřeba implementovat další funkci, a to prvně `create_persons_bind(rel_names, rel_values)`, kde se pouze vytvoří cyklus pro zobrazování jednotlivých polí hodnot, pomocí kterých se ještě s názvy sloupců vytvoří slovník a následně k samotnému vytváření hran mezi jednotlivými osobami se zavolá další funkce `create_persons_bind1(bind_dict, dictionary, rel=None)`. Poslední zmíněná funkce se volá celkem pětkrát, kdy první argument `bind_dict` odkazuje na jednotlivé slovníky z programu `dictionary.py` 4.3, druhý argument `dictionary` obsahuje vždy slovník s hodnotami pro vkládání vytvořený v rodičovské funkci a poslední argument `rel` obsahuje případný název hrany mezi uzly. Poslední funkcí, která slouží k tvorbě záznamů v grafové databázi, konkrétně k tvorbě hran `POTENCIONALNÍ_ZHODA` mezi uzly `Osoba`, je funkce

`create_person_probability(rel_names, rel_values)`, která zároveň vytvoří i vlastnost `pravděpodobnost_shody` dané hrany.

Tak aby se mohly všechny Neo4j dotazy vygenerované v předchozích funkcí provést, je potřeba mít implementovanou následující funkci `run_graph_query()`. Tato funkce v podstatě pouze provede vygenerovaný dotaz a v případě chyby ukončí program zachycenou chybou.

4.3 Pomocný slovník

Tato část je implementovaná pro jednodušší práci při určování názvu hran pro jednotlivé sloupce MySQL databáze.

Slovník `table_col` udává název hrany mezi osobou a matričním záznamem v grafové databázi pro jednotlivé sloupce z tabulky matričních záznamů v relační databázi. Slovník `is_in` určuje pro které spojení sloupců tabulek `mesto`, `ulice` a `cislo_pop` z Mysql databáze se vytváří hrana `JE_V`, slovník `lives` určuje propojení sloupců z tabulek zmíněných u předchozího slovníku a tabulkou `osoba` hranou `BÝVÁ`. Zbylé slovníky slouží ke zjištění hran mezi sloupci tabulek `osoba` a to tak, že slovník `father` udává název hrany pro všechny jeho prvky `JE_OTEC`, slovník `mother` hranu s názvem `JE_MATKA`, slovník `married` název hrany `JSOU_MANŽELÉ`, slovník `related` název hrany `JE_PRÍBUZNÝ` a u posledního nezmíněného slovníku `other` je název hrany určen klíčem jednotlivých záznamů slovníku.

4.4 Převod dat

V předchozích částech implementace jsme si představili zvlášť komunikaci s MySQL databází a Neo4j databází. V této části bych chtěl představit propojení zmiňovaných databází a zajištění převodu dat z Neo4j databáze do MySQL databáze a naopak. V první řadě se otevře pro zápis textový soubor určený k případnému logování běhu přenosu dat. Dále je potřeba se připojit jak k MySQL databázi, tak i k Neo4j databázi. Po úspěšném připojení se zavolá samotná funkce `main()`, díky které se zahájí samotný přenos dat mezi databázemi. V případě, chceme-li přesouvat data z Neo4j databáze do MySQL, je potřeba zavolat funkci `main(neo4j_to_mysql=True)`, v opačném případě `main(neo4j_to_mysql=False)`.

4.4.1 Neo4j do Mysql

Než dojde k samotnému převodu dat mezi databázemi, je potřeba založit tabulky v relační databázi MySQL, k tomu se využije funkce `create_all_table()`, která je součástí programu `mysqlDatabase.py` 4.1, poté dojde k samotnému přenosu dat a to v následujícím pořadí.

Jako první se provede převod dat z uzlů `Mesto` do tabulky `mesto` a to z důvodu, že se jedná o data, které nenavazují na žádná jiná data. Na města přímo navazují ulice a nakonec na ulice s městy navazují čísla popisná, to znamená, že nejdřív dojde k převodu dat z uzlů `Ulica` do tabulky `ulice` a následné propojení s tabulkou `mesto`. Jak už bylo zmíněno, další převod proběhne z uzlů `Popisná_čísla` do tabulky `cislo_pop`, kdy se při převodu doplní i případné vazby s předchozími tabulkami určující nějaké konkrétní místo.

Další částí převodu se týká dat matrik, kde se převádí data z uzlů `Matrika` do tabulky `matrika`. Kromě vytvoření záznamů ve zmiňované tabulce k evidenci matrik je nutné převést data ohledně jazyků a obcí výskytu do příslušných tabulek a propojit s hlavní matriční tabulkou.

Nyní, po převodu dat matrik, dojde postupně k převodu dat osob a všech tří druhů matričních záznamů a právě tyto jednotlivé převody bych zde popsal podrobněji.

Jako první se provede převod dat z uzlů `Osoba` do tabulky `osoba`. Princip je takový, že se nejdříve pomocí funkce z třídy `graphDatabase` 4.2 získají veškerá data ze všech uzlů `Osoba`. Z těchto získaných dat se pomocí vytvořených funkcí, zajišťujících potřebnou konverzi, začnou postupně, pomocí funkce z třídy `MysqlDB` 4.1, vytvářet jednotlivé záznamy v tabulce `osoba`. Po každém vytvoření záznamu se do určené proměnné uloží jeho jednoznačný identifikátor, který bude sloužit k propojení záznamů z pomocných tabulek osoby s tabulkou `osoba`. Začne se s převodem dat do pomocných tabulek `pohlavi`, `jmena_osoby`, `prijmeni_osoby`, `puv_id_db`, `nabozenstvi` a `povolaní` a to pomocí vytvořené funkce `fill_exp_table`, která zajistí správné vytvoření všech potřebných záznamů v zadané tabulce. Další na řadu přijde volání funkce `person_lives`, která zaručí přenesení propojení hranou z grafové databáze mezi uzly `Osoba` a `Mesto`, `Ulica` nebo `Popisné_číslo`. Získaná data těchto propojení se zaznamenají do vazební tabulky `misto`, která zaručuje propojení tabulky `osoba` s tabulkami `mesto`, `ulice` a `cislo_pop`. Po vytvoření převodu všech uzlů typu `Osoba` do tabulek relační databáze sloužící k evidenci osob, dojde k přenesení hran z grafové databáze týkající se pravděpodobnostní shody osob do pomocné tabulky `shoda_osob`, návazné k hlavní tabulce `osoba`.

Dostáváme se k nejdůležitější fázi převodu, a to k převodu samotných matričních záznamů a jejich příslušných hran. Na začátku převodu u všech tří druhů matričních záznamů se získají hodnoty vlastností pro všechny uzly patřící k danému druhu matričního záznamu. Provede se konverze hodnot určitých vlastností, tak aby se tyto hodnoty mohli vkládat do tabulek relační databáze. Následně se pomocí funkcí z třídy `graphDatabase` 4.2 a `MysqlDB` 4.1 založí záznam pro daný matriční záznam i s propojením k nadřazené tabulky `matrika`. Jakmile je vytvořený záznam se základními hodnotami, začnou se přidávat jednotlivé hodnoty pro sloupce, které slouží k vytvoření vazeb mezi matričním zázname a tabulkou `osoba`. Společně při vytváření těchto vazeb se zaeviduje i datum uskutečnění dané události. Jelikož některých typů osob návazných na daný matriční záznam může být více, vytváří se do pomocných tabulek k danému matričnímu záznamu. Jako poslední věc se převedou příbuzné osoby, a to v pomocné tabulce `zaznam_narozeni_kmotr` příbuzný pro kmotra a v pomocné tabulce `zaznam_oddani_svedek` příbuzný pro svědka.

4.4.2 Mysql do Neo4j

Nad rámec zadání byl implementovaný i převod dat z navrhnuté MySQL databáze do původní Neo4j databáze. K provedení tohoto převodu se znovu použijí obě dříve zmíněné třídy `graphDatabase` 4.2 i `MysqlDB` 4.1. Funkcionalita tohoto převodu je poměrně jednoduchá a to, že se vždy z MySQL databáze pomocí určitého `SELECT` dotazu zobrazí potřebná data, která se pomocí Cypher dotazů převedou do Neo4j databáze.

Na rozdíl od předchozího převodu, se zde začíná s převodem dat matrik, a to již zmíněným způsobem. Pomocí již implementované funkce se poskládá Cypher dotaz k vytvoření uzlu `Matrika` i se všemi vlastnostmi, ale prozatím se vyplní pouze ty jejich hodnoty, které je možné získat také z už implementované funkce pro získávání dat z Mysql databáze, zde z tabulky `matrika`.

Jakmile jsou založené všechny uzly matrik, přijde řada na tvorbu uzlů osob a to podobným způsobem jako právě předešlé uzly matrik.

Nyní jsou vytvořené všechny uzly typu `Matrika` a `Osoba` a může se začít s tvorbou uzlů k určení místa. Pro všechny typy uzlů patřící do této skupiny probíhá převod ta, že se nejdříve založí všechny uzly i se zadanými vlastnostmi. Následně dojde k případnému propojení daných uzlů s uzly typu `Osoba` hranou `BÝVÁ` a v případě existence se k hraně přiřadí i její vlastnost `date`. Poté u uzlů typu `Mesto` dojde k doplnění vlastnosti `gps_suradnice` z pomocné MySQL tabulky `mesto_gps`. U zbylých typů uzlů `Ulica` a `Popisné_číslo` k doplňování uzlů nedochází, ale dochází zde k vytváření hran `JE_V` propojující buď uzly typu `Ulica` s `Mesto` nebo `Popisné_číslo` s `Mesto` a nebo `Popisné_číslo` s `Ulica`.

Po vytvoření všech předchozích základních typů uzlů, bude potřeba založit uzly jednotlivých matričních záznamů, následné jejich propojení pomocí hran s osobami a poté navazující propojení hran mezi osobami, které jsou spolu ve vztahu. U všech tří druhů matričních záznamů se zmíněný převod provádí stejným způsobem pomocí příslušných implementovaných funkcí jak ze třídy `graphDatabase` 4.2, tak i ze třídy `MysqlDB` 4.1. Následně u každého typu matričních záznamů dojde k přesunu dat z pomocných tabulek, konkrétně doplnění kmotrů a jejich příbuzných u typy uzlu `Záznam_o_křte` a `Osoba`, svědků, jejich příbuzných, družby a družiček u typu uzlu `Záznam_o_svatbe` a `Osoba` a potomků a příbuzných u typu uzlu `Záznam_o_úmrti` a `Osoba`. U všech těchto převodů z pomocných tabulek matričních záznamů se provede i doplnění hran mezi danými uzly typu `Osoba`.

V další fázi převodu proběhne doplnění chybějících vlastností a to doplnění vlastnosti `Obce` a `Jazyky` u uzlů `Matrika` a doplnění vlastností týkajících se povolání, náboženství, vlastnosti s názvem `pohlaví`, `jméno`, `příjmení` a `id_relačná_databáza` u uzlů `Osoba`.

Nyní už je převod skoro dokončený, zbývá jen převést hrany mezi jednotlivými osobami, konkrétně doplnění hran `POTENCIONALNÍ_ZHODA` a doplnění její případné vlastnosti s názvem `pravděpodobnost_shody`.

Tímto popsaným postupem je možné z vytvořené MySQL databáze, vytvořené pomocí předešlého převodu, vytvořit kopii původní Neo4j databáze a je možné si tímto způsobem otestovat správnost nově vytvořené relační databáze. Každopádně tento převod není primárně určený pro testování, k testování jsou vytvořené samostatné skripty, o kterých více v další části 4.5.

4.5 Testování

Pro výsledné naimplementované řešení jsem společně s vedoucím mé práce vymyslel následující testování. Otestovat rychlost převodu a to jak z Neo4j databáze do MySQL databáze, tak i naopak a poté tyto výsledky porovnat a zhodnotit. Hlavně ale bude potřeba otesto-

vat správnost samotného převodu, a to nejen kontrola počtu převedených záznamů, ale i kontrola jejich hodnot. Jelikož hlavním úkolem mé práce byl převod z Neo4j do MySQL, testoval jsem správnost pouze tohoto převodu a také, jak už jsem zmiňoval, i zpětný převod se může použít jako forma testování. Všechny testování jsem prováděl s daty původní Neo4j databáze, kde se nachází 11374 uzlů osmi různých typů (`Matrika`, `Osoba`, `Mesto`, `Ulica`, `Popisné_číslo`, `Záznam_o_křte`, `Záznam_o_svatbe` a `Záznam_o_úmrti`) a 23696 hran, které navzájem tyto uzly propojují.

4.5.1 Rychlost převodu

Co se týká testování rychlosti převodu, zde jsem si nasimuloval celkem 40 převodů. 10 z původní Neo4j do nově navržené MySQL databáze, a 10 převodů z navržené MySQL databáze do nové Neo4j databáze. Toto testování jsem prováděl jak pro lokální databázi MySQL, tak i pro vzdálenou MySQL databázi na zpřístupněném školním serveru Radegast a databázi Neo4j jsem použil pouze lokálně, protože vzdálená databáze nebyla zprovozněna. U každého nasimulovaného převodu jsem si zaznamenával jeho čas a z těchto časů jsem si následně vypočítal průměrnou dobu převodu, zaznamenal nejdelší a nejkratší dobu trvání převodu a výsledky zapsal do tabulky 4.1.

Typ převodu	Počet převodů	Doba trvání převodu [s]		
		Minimální	Maximální	Průměrná
Neo4j -> Lokální MySQL	10	2110	2235	2165
Lokální MySQL -> Neo4j	10	772	799	783
Neo4j -> Vzdálený MySQL	10	3144	4083	3716
Vzdálený MySQL -> Neo4j	10	773	801	788

Tabulka 4.1: Rychlost převodu dat mezi databázemi

Z naměřených dat 4.1 jde jednoznačně vidět, že převod z MySQL databáze do Neo4j je časově méně náročný, a to jak při převodu s lokálním nebo vzdáleným databázovým serverem MySQL. Dokonce u tohoto převodu nemá typ připojení k MySQL serveru žádný vliv. Protože se při převodu z MySQL databáze do Neo4j databáze používá pouze SQL dotaz `SELECT`, pro získání dat k přenesení a poté se zbytek dělá v grafové databázi nám dokazuje, že Neo4j databáze zaručuje rychlejší manipulaci s daty, tedy alespoň dotazy při ukládání dat, než databáze relační. Naopak u převodu z Neo4j do MySQL se kromě vyhledávacích dotazů obou dotazovacích jazyků používají hlavně SQL dotazy `INSERT` a `UPDATE`, které z velké části způsobují pomalejší převod dat.

4.5.2 Správnost převodu

Předchozí testování slouží pouze pro informaci, jak přibližně dlouho daný převod trvá. Hlavním testováním je kontrola správnosti převodu. Toto testování se provádí pomocí vygenerovaného csv souboru z původní Neo4j databáze a testovacího skriptu. Výsledky testování se zapíší do externího souboru určeného k zapisování průběhu

Testování začíná tak, že se zkontroluje správnost převedení všech uzlů do jednotlivých záznamů v tabulkách navržené relační databáze. Jedná se o kontrolu mezi následujícími typy uzlů a k nim příslušnou tabulkou, a to jak kontrolou počtu převedených záznamů,

tak i správnost převedených hodnot záznamů. Typ uzlu `Osoba` s tabulkou `osoba`, `Matrika` s tabulkou `matrika`, `Mesto` s tabulkou `mesto`, `Ulica` s tabulkou `ulice`, `Popisné_číslo` s tabulkou `cislo_pop`, `Záznam_o_křte` s tabulkou `zaznam_narozeni`, `Záznam_o_svatbe` s tabulkou `zaznam_oddani` a `Záznam_o_úmrti` s tabulkou `zaznam_umrti`.

Po kontrole správnosti převodu u hlavních tabulek se přejde na kontrolu pomocných tabulek `povolani`, `pohlavi`, `nabozenstvi`, `jmena_osoby`, `prijmeni_osoby` a `puv_id_db` s uzlem typu `Osoba`, tabulky `mesto_gps` s uzlem `Mesto` a tabulek `jazyk`, `misto` a `mesto` s uzlem `Matrika`. U těchto kontrol se postupně prochází testovací csv soubor a pomocí sloupce `id_gdb`, který obsahuje jednoznačný identifikátor na původní záznam v Neo4j databázi, se najde daný záznam z csv souboru a otestuje se správnost převedených hodnot.

Poslední fáze tstování se týká kontroly správného převodu hran. Princip je zde takový, že se z testovacího csv souboru vyberou záznamy hran, kde se nachází informace o jaký typ hrany se jedná a odkud kam tato hrana směřuje. Následně se každá tato hrana projde a zjišťuje se, jestli se přenesla správně a vznikla v MySQl databází správná vazba.

Jak už jsem zmínil na konci předchozí části 4.4.2, tak k otestování může posloužit i zpětně vytvořený převod. Jelikož tento převod nebyl součástí zadání a byl dodělán v průběhu práce, nepoužil jsem ho k testování a pro testování správnosti převodu jsem použil pouze testování pomocí csv souboru.

Kapitola 5

Závěr

Hlavním úkolem mé bakalářské práce bylo navrhnout MySQL databázi a skripty pro převod genealogických dat z grafové databáze Neo4j a následně tyto návrhy implementovat a otestovat.

Otestováním jsem zjistil, že by implementace v případě, že se k převodu použijí správná data z Neo4j databáze obsahující genealogická data, měla být funkční a měla splňovat potřeby proč k této implementaci došlo. Po převodu máme k dispozici stejná data jako jsou v původní grafové databázi a je možné nad nimi pracovat pomocí SQL dotazovacích skriptů. V případě nějaké úpravy nebo vytvoření nových záznamů v nově vytvořené MySQL databázi je možné použít zpětný převod z mySQL do Neo4j, který byl po domluvě s vedoucím mé práce vytvořen nad rámec zadání. Tímto zpětným převodem dostaneme novou Neo4j databázi i s případnými upravenými záznamy a využít veškeré výhody grafové databáze. Jednoduše řečeno, tato práce slouží k tomu aby se s genealogickými daty mohlo pracovat jak v relační, tak i grafové databázi a to protože některé věci jdou u jednoho typu databáze řešit jednodušeji a efektivněji než u druhého typu, nebo dokonce některé věci v jedné databázi nejdou a v druhé ano.

Věřím, že tato má práce bude v případě potřeby sloužit, tak jak má a bude tak zajištěna lepší kompatibilita při vyhledávání genealogických dat mezi grafovou databází Neo4j a relační databázi MySQL.

Literatura

[1] *Genealogie aneb rodopis*. 2015. [Online; accessed 28.12.2020]. Dostupné z: http://www.genealogie.cz/genealogie/.

[2] Bryce Merkl Sasaki, R. H. *Graph Databases for Beginners*. Neo4j, 2018. https://go.neo4j.com/rs/710-RRC-335/images/Graph_Databases_for_Beginners.pdf.

[3] Corporation, O. *MySQL 8.0 Reference Manual*. Oracle Corporation, 2021.

[4] Harrington, J. L. *Relational Database Design and Implementation*. Todd Green, 2016. ISBN 978-0-12-804399-8.

[5] Ian Robinson, E. E. *Graph Databases, 2nd Edition*. 2. vyd. Beijing: O'Reilly Media, Inc, červen 2015. ISBN 978-1-49-193089-2.

[6] J., D. C. *An introduction to database systems. 8th Edition*. 8. vyd. Boston: Pearson/Addison Wesley, 2004. ISBN 978-0-32-119784-9.

[7] Lednická, B. *Sestavte si rodokmen pátráme po svých předcích*. Grada, 2012. ISBN 978-80-247-4069-0.

[8] Peterka, J. *Cesta k rodinným kořenům aneb Praktická příručka občanské genealogie*. Libri, 2006. ISBN 80-7277-307-0.

[9] Silberschatz Abraham, S. S. *Database system concepts, 6th Edition*. 6. vyd. New York: McGraw-Hill Higher Education, 2011. ISBN 978-0-07-352332-3.

[10] Skřivan, J. *Databáze a jazyk SQL*. 2000. [Online; accessed 20.12.2021]. Dostupné z: https://www.interval.cz/clanky/databaze-a-jazyk-sql/.

[11] Sperát, P. I. *Rodokmeny - Rodokmeny a nakladatelství*. 2019. [Online; accessed 28.12.2020]. Dostupné z: https://www.sperat.cz/rodokmeny/.

[12] Tušimová, L. *Generování rodokmenů z matričních záznamů*. Brno, CZ, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/22818/.

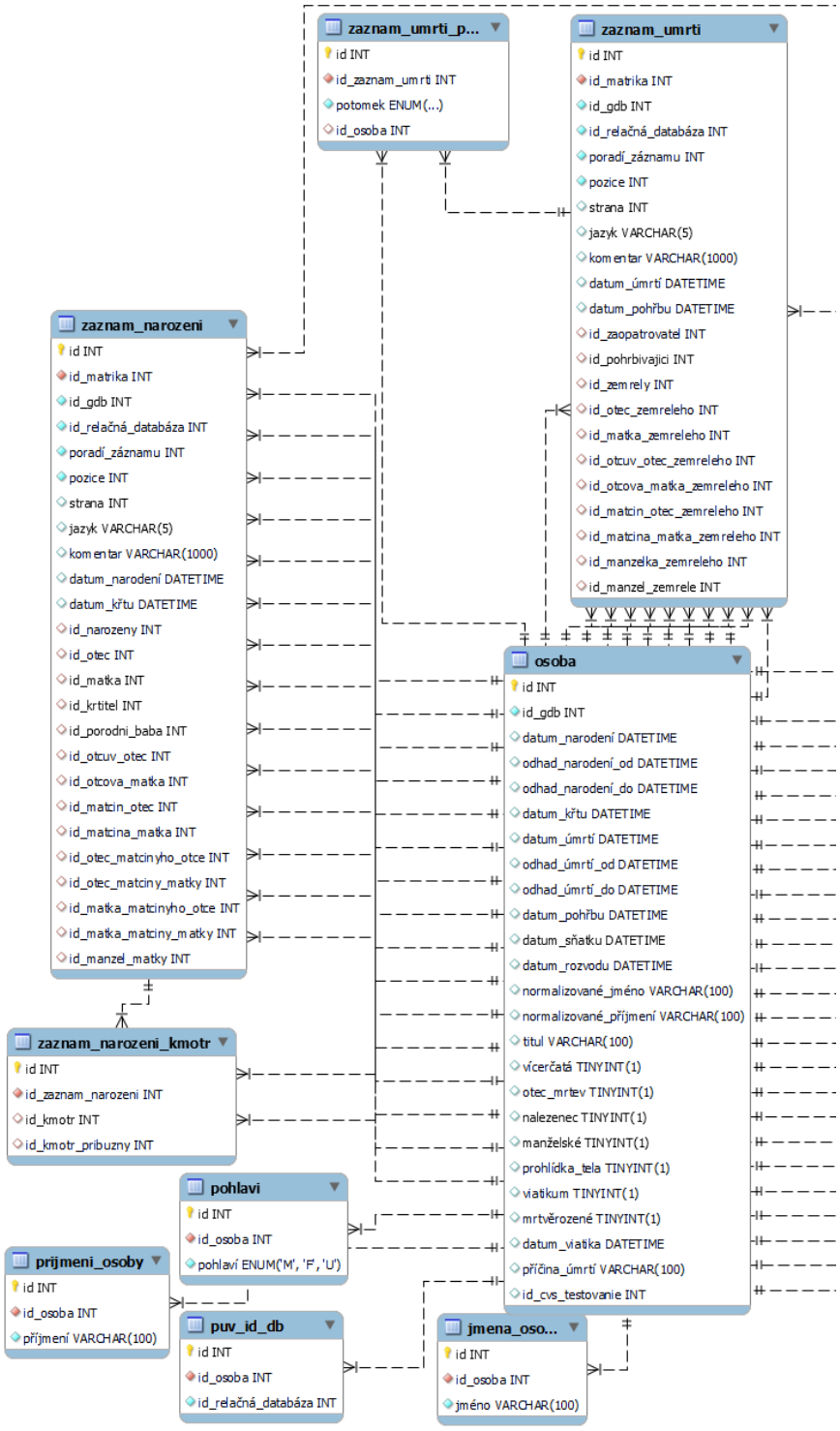
Příloha A

Obsah přiloženého paměťového média

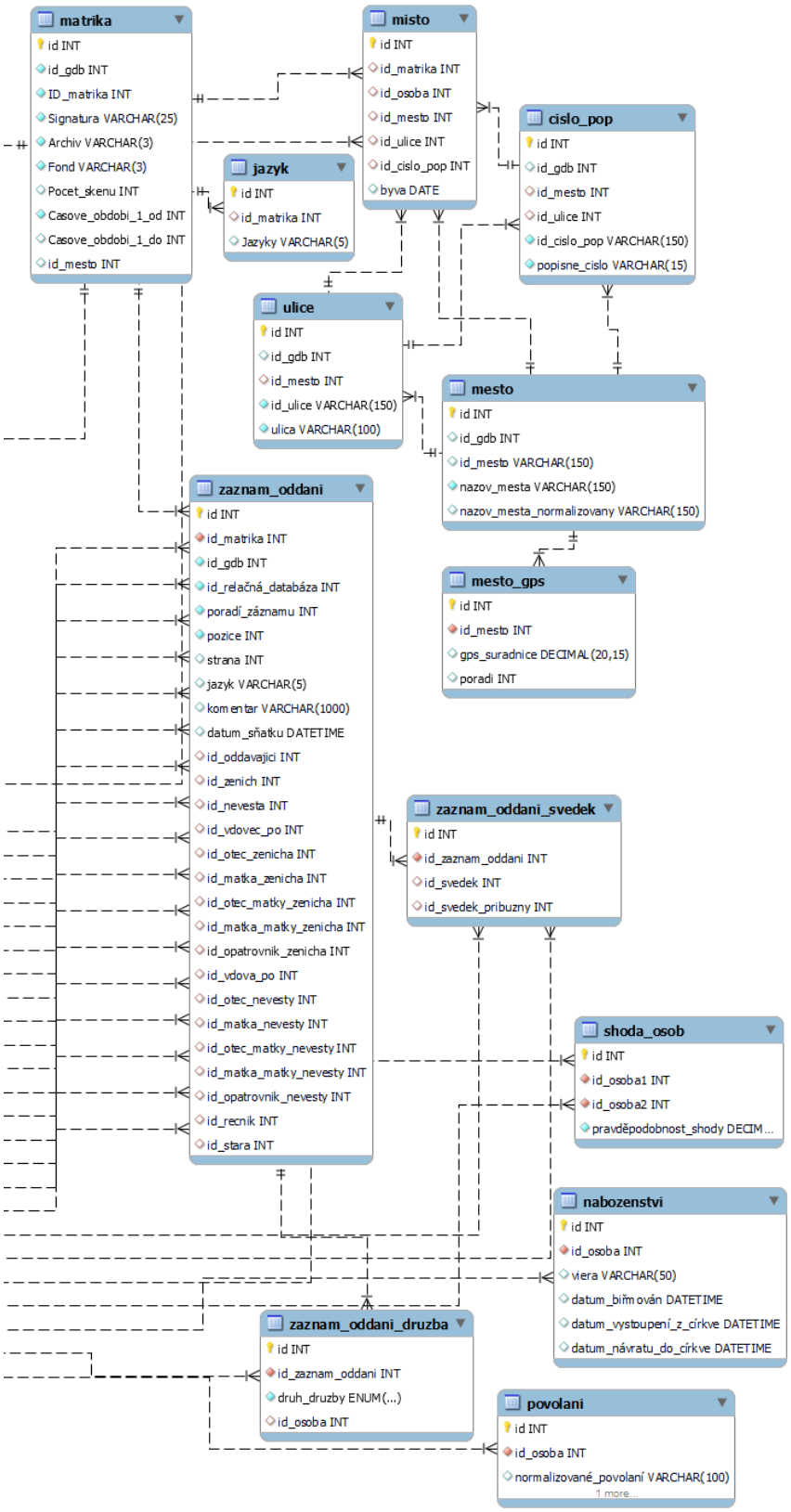
- Importní json soubor
- Testovací csv soubor
- Zdrojové kódy
- Návod k použití
- Zdrojové soubory LaTeX
- PDF technické zprávy
- Obrázek návrhu relační databáze

Příloha B

Návrh genealogické databáze



Obrázek B.1: Návrh MySQL databaze 1/2.



Obrázek B.2: Návrh MySQL databaze 2/2.