Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních věd

Grafové databáze Neo4j

Bakalářská práce

Autor: Pavel Ardolf
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Barbora Tesařová, Ph.D.

Hradec Králové duben 2016

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

……..

V Hradci Králové dne 29.4.2016 Pavel Ardolf

Poděkování:

Děkuji vedoucímu bakalářské práce Ing. Barboře Tesařové, Ph.D. za metodické vedení práce, připomínky, cenné rady a čas, který mi věnovala.

Anotace

Hlavní myšlenkou této práce je představení nerelační grafové databáze Neo4j. Nejprve jsou představeny grafy v podobě datových struktur, jejich základní algoritmy a obecná teorie grafů. Poté je práce zaměřena na NoSQL databáze, jejich druhy, principy a možnosti jejich využití. Následně se práce detailně zabývá grafovými databázemi Neo4j. Nerelační grafová databáze Neo4j je představena s využitím jazyka Cypher. Následně je v relační databázi MySQL a grafové databázi Neo4j navržen a implementován datový model. Nakonec je vybraná testovací nerelační databáze otestována s relační databází a je provedeno zhodnocení dosažených výsledků, které vyhodnotily jako vhodnou z hlediska rychlosti vykonání dotazů právě nerelační databázi Neo4j.

Annotation

Title: Neo4j graph databases

The main idea of this bachelor thesis is to introduce non-relational graph database Neo4j. First graphs are presented in the form of data structures, algorithms and their underlying general theory of graphs. Then the work is focused on NoSQL databases, their types, principles and possibilities of their use. Subsequently, the work examines in detail using graph database Neo4j. Non-relational database Neo4j graph is presented using language Cypher. Subsequently, the MySQL relational database and graph database Neo4j designed and implemented data model. Finally, the selected test non-relational databases tested with relational databases and an assessment of the results achieved, which evaluated as appropriate in terms of speed of execution of queries being non-relational database Neo4j.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Využívání grafových databází představuje jednu z nejdůležitějších změn posledních let, především v oblasti webových aplikací. S příchodem například sociálních sítí, kde je třeba ukládat velké množství dat, relační databáze nestačí. Není divu, že zájem o grafové a NoSQL databáze rapidně narůstá. Tato revoluce má významný dopad na technologie a nástroje po celém světě.

Grafem se dá reprezentovat spousta věcí a případů. Uplatnění najdou skoro všude, kde záleží na vzájemné propojenosti a vztazích mezi nějakými objekty. Grafy se používají v matematice, ekonomii, zeměpisu a mnoho dalších.

Předchůdce grafových databází nazývaný relační databáze nemusí být vždy to správné řešení pro nějaký problém. Kdybychom pro experiment vytvořili 1 000 000 generovaných záznamů, které představují osoby a každé z těch z osob měla pak vztah neboli vzájemnou propojenost k 50 dalším, měly by relační databáze značný problém. Je statisticky dokázáno, že kdybychom chtěli nad touto databází napsat dotaz, zda existuje cesta mezi dvěma osobami do hloubky 4, relační databáze by nebyla schopná tento dotaz dokončit v akceptovatelném čase. Kdybychom tento případ snížily na pouhých 1 000 osob, relační databáze by vykonala tento dotaz za 2 000 ms. Tento případ se tedy absolutně nehodí pro relační databáze. Pokud bychom tento případ implementovali do grafové databáze, v obou případech bychom zjistili, že tato databáze si s dotazem poradí do 2 ms. Mohli bychom si představit, že tento případ by se hodil na sociální sítě, kde je uživatelů mnohem více. Není proto divu, že většina sociálních sítí mají databáze založená na právě grafových databázích.

Cílem práce je seznámit se s grafovou databází Neo4j implementovanou do vybraného příkladu využití a otestovat dotazy implementované v nerelační i relační databázi.

2 Graf

Samotný graf se skládá z vrcholů a hran. Hrana vždy spojuje dva vrcholy a je buď orientovaná, nebo neorientovaná. U orientovaných hran rozlišujeme počáteční a koncový vrchol. Zde hrana vede z počátečního do koncového vrcholu. Neorientované hrany chápeme jako symetrické spojení dvou vrcholů. Může se stát, že hrana spojuje vrchol se sebou samým, pak takovou hranu nazýváme smyčkou. Všechny hrany orientované má orientovaný graf. Neorientovaný graf má všechny hrany neorientované. Také existují smíšené grafy, které mají oba druhy hran. [1]

V matematice i informatice je mnoho situací, kde lze grafy efektivně využít v prakticky motivovaných úlohách. Pomocí množiny bodů (vrcholů) a spojnic mezi některými dvojicemi bodů (hran) lze vystihnout situaci v dané problematice. [2]

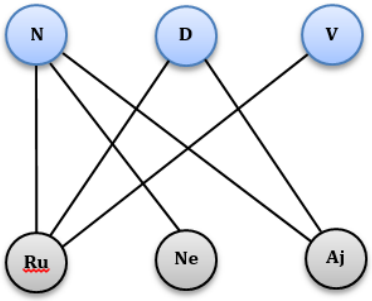
Dle matematické definice je obyčejný graf uspořádaná dvojice *(V, E)*, kde V je neprázdná množina vrcholů a *E* je množina hran, přičemž hrana *e* z *E* je dvouprvková podmnožina množiny *V*. Obyčejný graf nazýváme obyčejným grafem, jestliže množiny *V* a *E* jsou konečné. Počet prvků množiny V značíme |*V*| nebo *n* a počet prvků množiny *E* značíme |*E*| nebo *m*. [3]

Právě graf jako datová struktura je samotným základem grafových databází Neo4j.

2.1 Příklad

Na představení grafů je vhodný příklad znázorněný na obrázku 1. Jedna instituce vypsala konkurz na obsazení 3 míst pro překladatele, a to z ruštiny, němčiny a angličtiny. Přihlásili se 3 uchazeči, kteří ovládali některé z těchto jazyků. Pan Novák ovládal všechny tři požadované jazyky, pan Dvořák hovořil ruštinou a angličtinou, a pan Veselý, který mluvil jenom ruským jazykem. Tato situace se dá reprezentovat grafem, ze kterého bude jasně patrné, co graf znázorňuje. Grafové databáze Neo4j jsou pro takovéto typy úloh typické.

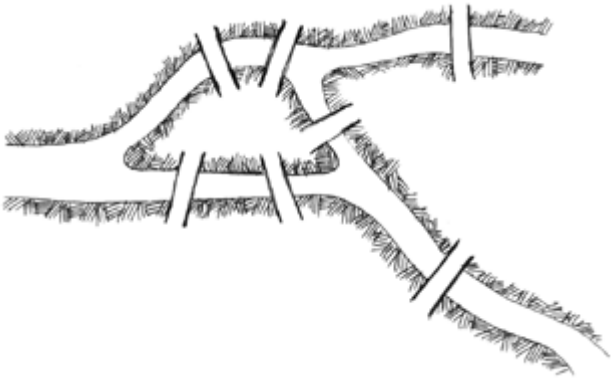
Obrázek 1, Reprezentování úlohy grafem, Zdroj: vlastní zpracování



2.2 Historie

Prvním problémem, kdy bylo třeba využít teorie grafů a je pokládán za počátek teorie grafů, byla úloha nazvaná sedm mostů města Královce. Ve městě Königsberg (česky nazvaném Královec, dnešní Kaliningrad) jsou v centru města na řece Pregel (dnes Pregolya) dva ostrovy, které v 18. století spojovalo s oběma břehy sedm mostů – viz Obrázek 2. [3]

Obrázek 2, Sedm mostů města Královce, Zdroj: [16]



Obyvatelé města při nedělní procházce často přemýšleli nad tím, zda by bylo možné projít se městem tak, že by procházku začali v jednom místě, přešli všechny mosty, přes každý právě jednou a skončili by tam, kde začali. Protože se jim stále nedařilo najít způsob, jakým by to provedli, začali se mnozí z nich domnívat, že takovou procházku ani uskutečnit nejde. Tento názor byl matematicky potvrzen až roku 1736, kdy byl uvedený problém předložen k vyřešení švýcarskému matematikovi Leonhardu Eulerovi. Tento matematik vyřešil nejen problém sedmi mostů města Královce, kde potvrdil, že není možné uskutečnit procházku výše uvedeným způsobem, ale také podal obecnou charakteristiku úloh daného typu. [3]

Poté se grafy jako takové na dlouhou dobu ocitly mimo zájem matematiků. Ke grafům se vrátili až Gustav Kirhchoff (1824-1887), Arthur Caley (1821-1895) nebo také sir William Hamilton (1805-1865). Problémem té doby bylo vyřešit úlohu čtyř barev. Hlavní otázka byla, jak na mapě obarvit státy pomocí čtyř barev tak, aby žádné dva sousední státy neměly stejnou barvu. Vyřešení tohoto problému se dostalo až v roce 1976. [16]

2.3 Podgraf grafu

Podgraf grafu je takový graf, který vznikne odebráním některých z vrcholů nebo hran z původního grafu. Při odebrání vrcholu je nutné odebrat všechny hrany vedoucí do respektive z tohoto vrcholu. Pokud byly odebrány pouze tyto hrany, nazývá se podgraf indukovaný. Pokud byly odebrány i jiné hrany a vrcholy, jedná o obecný podgraf. Faktorový podgraf grafu (někdy pouze faktor) je nazýván takový podgraf, který obsahuje všechny vrcholy původního grafu. [3]

Při vytváření dotazů pro vyhledávání se v Neo4j často používají podgrafy, kdy při definování algoritmus nepotřebujeme procházet celý graf, ale jen potřebný podgraf.

2.4 Strom

Z hlediska grafových databází Neo4j pro nás bude důležitý termín termín strom. Jedná se o souvislý graf, který neobsahuje žádnou kružnici. Po odebrání jedné z hran je porušena souvislost a přidáním jedné hrany vzniká kružnice. Strom je minimální souvislý graf na daných vrcholech. Pro stromy platí vztah, který říká, že počet hran je jen o jedna menší než počet uzlů. Les je graf, jehož všechny komponenty jsou stromy. [3]

Stromy se často v informatice používají pro ukládání dat jako tzv. datová struktura. Typickou takovou strukturou je binární vyhledávací strom. Jedná se o strom, kde z každého vrcholy vedou nejvýše tři hrany. Jedna vede „seshora“, které říkáme od „otce“ a maximálně dvě vedou „dolů“, tzn. ke dvěma „synům“. Pro každý prvek platí, že jeho levý syn je menší nebo roven otci a pravý syn je pak větší. Oproti ostatním datovým strukturám, jako je pole či spojový seznam, je binární vyhledávací strom rychlejší v nalezení prvku. Rychlejší nalezení prvku je hlavním důvode, proč se tato datová struktura používá. [3]

Obrázek 3, Příklad stromových grafů, Zdroj: [3]

2.4.1.1Problém minimální kostry grafu

Problém minimální kostry grafu patří mezi klíčové úlohy kombinatorické optimalizace. Na zrodu tohoto problému se podíleli dva významní čeští matematici. Na přelomu let 1925 - 1926 se Otakar Borůvka seznámil s pracovníkem Západomoravských elektráren Jindřichem Saxelem, který ho žádal o pomoc. Šlo o to, kdy a jak vést trasu, která měla spojovat několik desítek obcí na Moravě, aby byla co nejkratší a tím co nejúspornější. Otakar Borůvka své matematické řešení uvedl v článku „O jistém problému minimálním“. Jeho řešení však nebylo příliš průhledné, což si uvědomil další český matematik Vojtěch Jarník. Na Borůvkův článek „O jistém problému minimálním“ reagoval v článku napsaném netradiční formou dopisu se stejnojmeným názvem s podtitulkem (Z dopisu panu O. Borůvkovi). Ani jeden z uvedených matematiků patrně netušil, že se stali autory jednoho z nejznámějších optimalizačních problémů. Řešení problému minimální kostry grafu byla v obou podobách (Borůvky a Jarníka) nezávisle znovuobjevena ještě několikrát. Hlavním z důvodů byla skutečnost, že většina prací byla psána v českém jazyce a často tak matematikům z jiných zemí nedostupná. Řešení popsané Vojtěchem Jarníkem znovu nezávisle objevil J. B. Kruskal roku 1956. Třetí řešení, odlišné od předchozích dvou, podal roku 1956 opět J. B. Kruskal ve své práci „On the shortest spanning tree of a graph and the traveling salesman problém“. „Jistý problém minimální“ nazýváme v současné terminologii jako Problém minimální kostry. [3]

V definici se uvažuje o ohodnocení hran reálným číslem. V praxi však jedná o čísla nezáporná. Minimálních koster v daném grafu může být i více. Stačí si představit případ, kde graf, jehož všechny hrany jsou ohodnoceny stejným číslem. V takovém případě je počet minimálních koster roven počtu všech koster. [3]

2.4.1.1.1 Borůvkův algoritmus

Na počátku, nechť je každá hrana grafu *G* neobarvená, hrany mají navzájem různé ohodnocení, a nechť každý vrchol grafu *G* představuje modrý strom (uvažujeme modrý les složený z *n* modrých stromů). [3]

V každém kroku algoritmu, dokud modré hrany netvoří strom, provádíme tyto operace. Pro každý modrý strom najdeme mezi všemi hranami, jejichž jeden vrchol leží v uvažovaném stromu a druhý nikoli, hrany s nejmenším ohodnocením. Tyto hrany obarvíme modře a tak získáme modrý les. Algoritmus končí získáním modrého stromu, tedy minimální kostry grafu *G*. [4]

2.4.1.1.2 Jarníkův algoritmus

Na počátku nechť je každá hrana grafu *G* neobarvená. Zvolíme libovolný vrchol a považujeme jej za modrý strom. [3]

V každém z *(n - 1)* kroků obarvíme modře hranu s minimálním ohodnocením, jejíž jeden vrchol leží v modrém stromu a druhý v něm neleží. Existuje-li jich více, zvolíme k obarvení libovolnou z nich. Algoritmus končí jako v předchozím případě, získáním modrého stromu a tím minimální kostry grafu *G*. [4]

2.4.1.1.3 Kruskalův algoritmus

Na počátku nechť je každá hrana grafu *G* neobarvená. Provedeme uspořádání hran vzestupně podle jejich ohodnocení. Každý vrchol grafu *G* považujeme za modrý strom. [3]

V každém z m kroků rozhodneme o právě jedné hraně (v pořadí daném jejich uspořádáním), zda ji obarvíme modře či nikoli. Modře obarvíme hranu právě tehdy, když netvoří s modře obarvenými hranami kružnici (koncové vrcholy neleží v témže modrém stromu). Algoritmus ukončíme, jakmile je (n – 1) hran obarveno modře. Tyto modré hrany tvoří minimální kostru grafu G. [4]

2.5 Hledání nejkratší cesty

Jednou ze základních algoritmických úloh v Neo4j a samotných grafech je najít nejkratší cestu mezi dvěma vrcholy. Takové hledání vzniká v mnoha praktických aplikacích (hledání nejkratšího dopravního spojení). Algoritmy dosud známé

naleznou nejkratší cestu z výchozího vrcholu do všech (nebo mnoha) vrcholů. Mezi nejdůležitější algoritmy tohoto typu patří tzv. Dijkstrův algoritmus, který je implementovaný přímo implementovaný právě v Neo4j a používá se pro dotazy při hledání nejkratších cest grafu. [2]

2.5.1 Dijkstrův algoritmus

Jedná se o nejrychlejší známý algoritmus pro hledání nejkratších cest v Neo4j. Algoritmus byl vyvinut nizozemským informatikem Edsgerem Dijkstrou v roce 1959. [4]

Dijkstrův algoritmus se používá v situaci, kdy daný graf G má hrany ohodnocené kladnými reálnými čísly. Ohodnocení hrany si můžeme představovat jako její délku nebo cenu, kterou musíme zaplatit za její projití. Délka nějaké cesty v takto ohodnoceném grafu G je rovna součtu délek jejích hran a vzdálenost dvou vrcholů je pak rovna nejmenší z délek všech cest. V Neo4j probíhá ohodnocení hran defaultně. [2]

Ovšem pro hrany se záporně ohodnocenou hodnotou byl vyvinut modifikovaný Dijsktrův algoritmus, který poradí i s tímhle, ale v tomto případě pro něj platí horší časový odhad a také může dát chybné výsledky, protože nemá žádnou možnost zpracovat některý vrchol více než jednou. [1]

Tento algoritmus si uchovává všechny uzly v tzv. frontě. V každém svém kroku vybere uzel s nejvyšší prioritou (nejnižší vzdáleností od již zpracované části) a zařadí jej do fronty. Poté projde všechny zbývající uzly, které přidá do fronty a ověří, zda nejsou blíže zdroji, než byly před zařazením právě vybraného uzlu mezi zpracované. Ověřuje všechny tyto vlastnosti: [4]

Obrázek 4, Vzorec Dijkstrova algoritmu, Zdroj: [3]

$$vzdálenost_{zpracovávaný} + délkaHrany_{zpracovaný,\ potomek} < vzdálenost_{\ potomek}$$

Pokud tato nerovnost platí, tak potomkovi nastaví novou vzdálenost a označí jako předka zpracovaný uzel. Po celém tomto cyklu a samotného průchodu přes všechny potomky algoritmus vybere z fronty uzel s nejvyšší prioritou a celý postup opakuje. Algoritmus skončí v okamžiku, kdy jsou zpracovány všechny uzly grafu a fronta zůstane prázdná. [4]

2.5.2 Prohledávání do šířky

Prohledávání do šířky je jedním ze základních grafových algoritmů, který slouží k průchodu daného grafu. Jeho princip je základem například pro Jarníkův nebo Dijskrův algoritmus, který je velmi hojně využívaný v Neo4j. [4]

Při prohledávání do šířky začneme prohledávat graf z libovolného vrcholu. Tento uzel zpracujeme a při objevení jeho dosud nenalezených potomků ho uložíme do fronty a postupně zpracováváme stejným způsobem, dokud není fronta prázdná. [4]

Výsledek algoritmu je strom prohledávání do šířky obsahující všechny uzly přístupné z výchozího uzlu. Výstup je strom nejkratších cest z daného uzlu. [4]

2.5.3 Prohledávání do hloubky

Prohledávání do hloubky je algoritmus pro procházení grafu, který slouží například pro zjišťování počtu komponent, topologického uspořádání nebo detekci cyklů daného grafu. [4]

Na začátku zvolíme libovolný uzel a označíme ho jako otevřený, zpracujeme jej a zjistíme všechny dosud neobjevené potomky daného uzlu. Po návratu z rekurze uzel označíme jako uzavřený. Tímto způsobem dojdeme k průchodu všech větví grafu do hloubky. [4]

3 NoSQL

Databáze NoSQL jsou další generací databází. Toto hnutí začalo na začátku roku 2009 a rychle roste. Tento termín byl zvolen pro volně specifikovanou třídu nerelačních datových úložišť. Tyto databáze většinou nepoužívají jazyk SQL jako dotazovací jazyk. Termín NoSQL je vykládán spíše jako „not only SQL“ a v databázové komunitě je zavádějící. Do tohoto druhu se dají zahrnout i XML databáze, dokumentové databáze či objektové databáze. (Jaroslav Pokorný, 2013)
Pro databáze NoSQL jsou význačné hlavně tyto charakteristiky: nerelační, distribuované, open-source, horizontálně škálovatelné, jednoduché API, obrovské množství dat apod. Dnes podniky přijímají NoSQL databáze pro rostoucí počet případů, které jsou poháněny čtyřmi vzájemně spojujícími megatrendy ve světě informatiky: big users, big data, internet of things a cloud computing. [8]
Část NoSQL databází obvykle zjednodušuje nebo omezuje režii objevující se v relačních databázích s plnou funkčností. Data jsou organizována na logické úrovni do jistých řídkých i vícerozměrných tabulek a lze k nim přistupovat pouze pomocí primárního klíče. Data v těchto databázích mohou být rozdělena nejen horizontálně, ale i vertikálně. Lze využít replikací, což je proces kopírování a údržby databázových objektů ve více databázích. [6]

3.1 Důvody vzniku

Prvním zásadním důvodem, proč vznikly NoSQL databáze, je exponenciálně vzrůstající objem dat, které lidstvo generuje, ukládá a zpracovává. Když jsou objemy dat obrovské, relační databáze si s tím není schopná poradit. Druhým důvodem je propojenost dat. Pro vzájemnou propojenost dat se grafové databáze, implementovaných přes vrcholy navzájem spojeny hranami, dokonale hodí. Třetím důvodem je ztráta struktury. Už nelze každého jedince přidat do nějaké tabulky. Se zrodem sociálních sítí se předvídatelná struktura dat ztrácí. Čtvrtým a posledním důvodem vzniku NoSQL databází je architektura dnešních aplikací. Podle toho, jaký typ dat daná aplikace zpracovává, má každá aplikace právo si zvolit, kam data uloží a jakým způsobem.

3.1.1 Big users

Není to tak dávno, co 1000 uživatelů jedné aplikace bylo hodně a 10 000 bylo extrémní případ. Dnes téměř 3 miliardy lidí jsou připojeny k internetu a množství času tráví online (přibližně 35 miliard hodin měsíčně v roce 2014). Počet souběžně přihlášených uživatelů pořád roste. Nebylo obvyklé, že by aplikace měla miliony různých uživatelů denně a měla podporovat uživatele po celém světě 24 hodin denně, 365 dní v roce. [10]

Obrázek 5, Big users, Zdroj: [10]

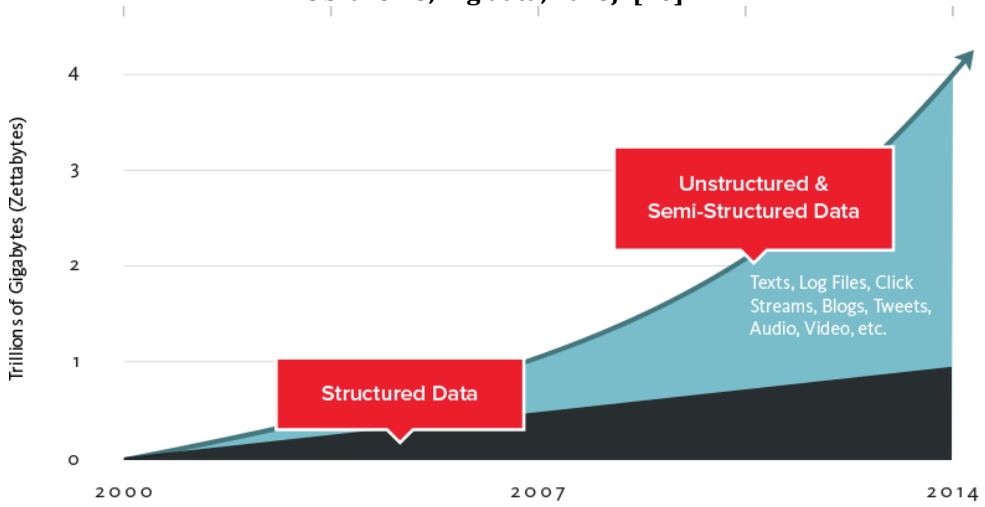
Z obrázku je vidno, že globální využití internetu roste rychle, stejně jako množství času, který každý uživatel na internetu stráví. Rostoucí využívání online aplikací má za následek rychle rostoucí počet databázových operací a potřebu jednoduššího systému škálování databází pro splnění těchto požadavků. [10]

Podpora velkého počtu souběžných uživatelů je důležitá. Nicméně požadavky na využití aplikace je těžké předpovídat, z toho vyplývá, že je také třeba předpovídat rychlý růst nebo naopak úbytek počtu souběžných uživatelů. Pro vývojáře relačních databází je proto obtížné, nebo dokonce nemožné, aby zajistili dynamickou škálovatelnost a úroveň stupnice, kterou zrovna potřebují. Toto je jeden z důvodů, proč použít právě NoSQL databázi. [10]

3.1.2 Big data

Množství dat a údajů rapidně roste. Z tohoto důvodu vývojáři začleňují nové datové typy, z nichž je většina nestrukturovaných nebo semi-struktorovaných. [10]

Obrázek 6, Big data, Zdroj: [10]



(couchbase.com, 2015)

Explosivní růst v používání internetu a mobilních i sociálních aplikací je hnacím motorem pro tzv. big data. Výzkumná firma IDC odhaduje, že v roce 2013 byla celková velikost světových digitálních dat 4,4 zettabytů, tj. 4,4 bilionů gigabytů. Podle expertů bude tato velikost až desetkrát větší. Například Facebook zachytává osobní informace o uživateli, lokaci, společenské grafy, uživatelsky generovaný obsah a mnoho dalšího. Proto není divu, že vývojáři se snaží obohatit stávající aplikace a vytvářet nové aplikace pro rostoucí hodnoty těchto údajů. Dostupnost těchto dat rychle mění charakter komunikace, nakupování, reklamy, zábavy a řízení vztahů. Příliš nových dat jsou nestrukturované a semi-strukturované, takže vývojáři potřebují ty databáze, které tyto data mohou ukládat. Pevně definované schéma relačních databází však neumožňuje rychle začlenit nové typy dat. [10]
Zachycení a využívání velkých objemů dat vyžaduje odlišný typ databáze. NoSQL databáze poskytuje mnohem pružnější datový model, který se na určitou aplikaci mapuje lépe a zjednodušuje tak interakci mezi aplikací a databází. [10]

3.1.3 Internet of things

Objem strojově generovaných údajů je hlavním přispěvatelem k růstu velkých dat. Přes 20 miliard zařízení je v současné době napojeno na internet od tabletů, domácích spotřebičů, systémů instalovaných v automobilech, nemocnic a skladů. Tato zařízení přijímají data na životní prostředí, umístění, pohyb, teplotu, počasí a

dalších z 50 miliard senzorů. Inovativní podniky využívají internetu k vývoji nových produktů a služeb, snížení nákladů, času na trhu, zvýšení efektivity, likvidace odpadu a zvýšení spokojenosti zákazníků. Tato schopnost zvyšuje pružnost podnikání a umožňuje dynamické rozhodování. Tyto databáze vyžadují semi-strukturovaná data, což představuje problém ve světě relačních databází, protože vyžadují pevné schéma a strukturovaná data. [10]

Obrázek 7, Internet of things, Zdroj: [10]

K překonání těchto problémů se podniky spoléhají na NoSQL databáze, které umožňují souběžný přístup k datům miliony připojených zařízení a systémů, ukládají miliardy datových bodů a splňují požadavky na výkon pro kritickou infrastrukturu a provoz. [10]

3.1.4 Cloud computing

Většina nových aplikací dnes běží na veřejném, soukromém nebo hybridním cloudu. Podporuje velký počet uživatelů a používá třívrstvou architekturu internetu. [10]
V třívrstvé architektuře jsou aplikace přístupné pomocí webového prohlížeče nebo mobilních aplikací, které jsou připojeny k internetu. V cloudu směřuje příchozí provoz na scale-out vrstvě webových nebo aplikačních serverů, které zpracovávají logiku aplikace. Scale-out architektura funguje na webové/aplikační vrstvě skvěle. Pro každých 10 000 nových souběžných se jednoduše přidá další komoditní server do vrstvy webové aplikace. Pro snadnou a dynamickou škálovatelnost jsou relační databáze proto nedostačující, protože pracují na databázové vrstvě. [10]
Databáze NoSQL jsou postaveny od základu tak, aby byly distribuovány a proto jsou lepší k třívrstvé architektuře internetu. [10]

3.2 ACID

NoSQL databáze se pyšní také tím, že jsou ACID kompatibilní. ACID je zkratka čtyř cílů, o které mnozí usilují a které budou následně popsány: [9]

- **Atomicity (atomicita)**: Transakce jsou atomické (dále nerozložitelné). Pokud se část transakce nezdaří, pak je vrácena zpět. Transakce buď proběhne celá, nebo vůbec neproběhne.
- **Consistency (konzistence):** Transakce transformuje databázi z jednoho konzistentního do jiného konzistentního stavu. V průběhu transakce však nemusí být databáze konzistentní.
- **Isolation (nezávislost):** Transakce jsou navzájem nezávislé. Dílčí efekty transakce nejsou viditelné jiným transakcím. Každá transakce vidí databázi jen v konzistentním stavu.
- **Durability (trvalost):** Jednou potvrzená transakce je trvalá. Efekty potvrzené transakce jsou uloženy do databáze.

Souhrnem toho všeho je, že databáze Neo4j byla navržena tak, aby byla multifunkčním řešením databázového stylu. Sdílí vlastnosti relačních databází, které jsou shrnuty do ACID. Používá však odlišný datový model, který se dobře hodí pro hustě spojené případy užití. [9]

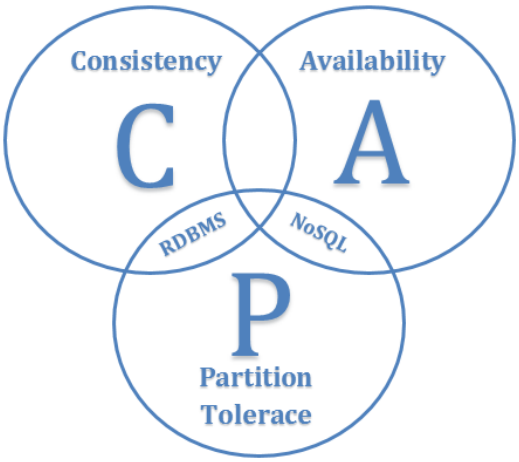
3.3 CAP Teorém

Při navrhování distribuovaného systému se používá tzv. CAP Teorém. Obsahuje tři základní vlastnosti na konzistenci, dostupnost a toleranci sítě. Také říká, že neexistuje systém, který by obsahoval všechny tyto tři vlastnosti najednou: [11]

- **Consistency (konzistence):** Všechny uzly distribuovaného systému v jednom určitém čase vidí stejná data. Jinak řečeno, všechny uzly mají shodná data.
- **Availability (dostupnost):** Každý dotaz dostane odpověď, zda byl úspěšný nebo neúspěšný

- **Partition Tolerance (odolnost proti výpadku sítě):** Pokud část sítě vypadne v důsledku přerušení spojení, systém je odolný a schopný pokračovat ve svém fungování

Obrázek 8, Cap teorém, Zdroj: [11]



U NoSQL databází lze podle Obrázku 7 dosáhnout pouze dvou vlastností ze tří uvedených. Je proto třeba při nasazování databáze zhodnotit své požadavky a rozhodnout se, zda bude lepší přístup AP nebo CP. [11]

3.4 Datové modely NoSQL

Logický datový model je nejzákladnějším klasickým přístupem k databázím. V současné době se bere v potaz rozdíl mezi konceptuálním a databázovým pohledem na data. Datové modely NoSQL databází se dají brát jako kategorie NoSQL. [6]

3.4.1 Key-value database (Klíč-hodnota)

Nejjednodušším datovým modelem NoSQL databází je úložiště zvané Key-value (klíč-hodnota). Tento datový model obsahuje pouze množinu dvojic klíč a hodnota. V relačních databázích je klíčem jméno atributu a v SQL databázích jméno sloupce. Jde o množinu pojmenovaných hodnot, kde klíč jednoznačně identifikuje hodnoty, která je typicky řetězec, ale i ukazatel, který ukazuje na danou hodnotu. [6]
Key-value databáze jsou určeny pro ukládání, vyhledávání, správu asociativních polí a datových struktur. Databáze obsahují určitý počet záznamů, které obsahují data.

Každý záznam je uložen a načten pomocí klíče, který jednoznačně identifikuje každý záznam. Z tohoto důvodu je tento typ databází používán pro rychlejší vyhledávání, ukládání a správu dat. Key-value databáze mají velice jednoduchý datový model, proto jsou masivně škálovatelné, výkonné a jsou odolné vůči chybám. [12]
Toto úložiště je velmi jednoduché svou strukturou i svými operacemi. API všech poskytovaných systémů obsahuje pouze tři operace: vložení hodnoty pro daný klíč (operace PUT), získání hodnoty pro daný klíč (operace GET) a smazání množiny klíč-hodnota (operace DELETE). Datové úložiště nezkoumá obsah vkládaných hodnot. Úlohou klientské aplikace pak je rozumět a znát klíče pro získání uložených hodnot, ovšem jsou systémy, které umožňují i jiné možnosti, jak s uloženými daty pracovat. [5]

Tabulka 1, Key-value stores

Key	**Value**
Jméno	Jan
Příjmení	Novák
Telefon	777555333

Přístup klíč-hodnota může svými přístupy připomínat jednoduché abstrakce, jakou jsou souborové systémy nebo hašovací tabulky, které jsou vhodně pro rychlé vyhledávání. Jde o rychlé a škálovatelné databáze, což by se dalo pokládat za výhody tohoto modelu, avšak jejich nevýhodou je příliš velká jednoduchost datového modelu. [6]

3.4.2 Column family database (Sloupcová)

Tento druh NoSQL databází je někdy nazýván jako key value value value databáze. Zde má každá řádek své schéma. Příkladem může být databáze Cassandra, kterou vyvinul Facebook. Jsou založené na Google Big table publikaci. Oproti relačním databázím, kde každý řádek představoval jeden záznam v tabulce, v column family databáze ukládají každý záznam do sloupce. Stejně jako key-value databáze jsou dobře škálovatelné a mají dobrý výkon. Mají však bohatší datový model než key-value databáze. [13]

Datový model těchto systémů si můžeme jednoduše představit jako tabulku, u které je možné do každého řádku volně přidávat sloupce bez nutnosti přidat je i do řádků ostatních. Takové volné nakládání se sloupci nesnižuje výkonost sloupcových databází, které bývají důkladně škálovatelné. [6]

Tabulka 2, Column family stores

RowID	Columns		
1	Jan	Novák	777555333
2	Pavel	Dvořák	728333555
3	Jaroslav	Veselý	720535353

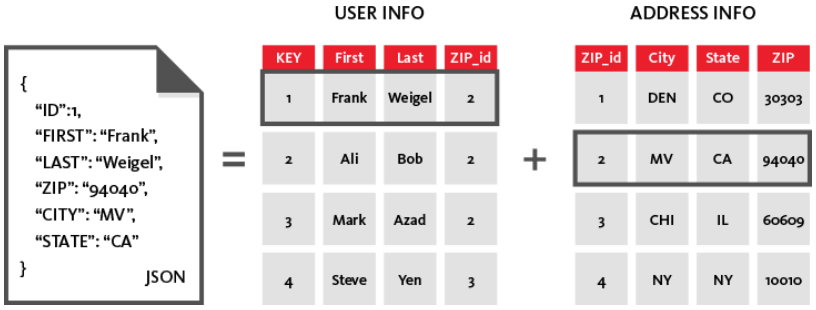
Pojmy column family, super column a column jsou v této oblasti velmi zásadní. Column family znamená to, jak jsou data uložena na disku. Všechny data v jednom column family budou uložena ve stejném souboru. Column family může obsahovat super columns a columns. Super columns obsahují další columns, ale žádné další super columns. Column je n-tice z názvu, hodnoty a časové značky. [13]

3.4.3 Dokument database (Dokumentová)

Dokumentové databáze ukládají a spravují odlišné druhy strukturovaných dokumentů, které s sebou nesou informaci o svém významu, mají tzv. samopopisný charakter. Formáty JSON, XML a jiné hierarchické struktury jsou typickým příkladem dokumentových databází. Tyto databáze umožňují přistupovat k dokumentům a vyhledávat v nich podle jejich obsahu. [5]

Příkladem může být například databáze MongoDB, u kterých je dokument sada klíčů a hodnot, kterou uložím jako celek. Tyto databáze mají zase o něco složitější datový model, takže jsou hůře škálovatelné. V této databázi můžeme dokumenty sdílet, takže můžeme pro určité lidi uložit určité typy dokumentů. Když chceme získat náhled do dat, jsou velmi důležité indexy. [14]

Obrázek 9, Dokumentová databáze, Zdroj: [26]



Tyto tři databáze mají záměrně jednodušší datový model než relační databáze, aby mohli garantovat škálovatelnost, dostupnost apod. Mají však jednu velkou nevýhodu. Nevidíme vzájemnou propojenost dat jako je to v posledním typu NoSQL databází. [14]

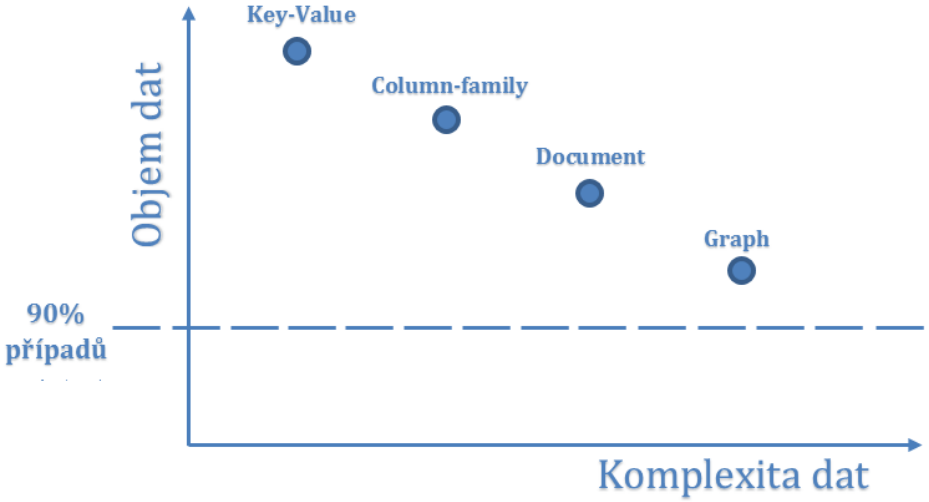
3.4.4 Graph database (Grafová)

Grafové databáze nepoužívají pevný formát tabulkového a sloupcového znázornění jako u SQL databází, ale je zde použito grafické znázornění a teorie grafů. Grafová databáze je tvořena sítí vrcholů (uzlů) a hran. Jednotlivé vrcholy zde představují danou entitu a hrany představují vazby mezi entitami. Vrcholy i hrany mají své atributy. Typem těchto datových úložišť se budeme zabývat v následujících částech práce. [15]

3.5 Porovnání datových modelů

Porovnání NoSQL databází vzhledem k jejich komplexnosti a množství dat, které dokážou pojmout, je patrno na obrázku 10. Key-value databáze jsou schopny uložit největší objem dat, nicméně tento datový model má nejmenší složitost. Na opačné straně jsou grafové databáze, jejichž složitost je největší ze čtyř uvedených typů. Ačkoli podle obrázku 10 pojmou grafové databáze nejmenší množství dat, stále se jedná o hodnoty v miliardách vrcholů a vazeb. [27]

Obrázek 10, Porovnání datových modelů NoSQL, Zdroj: [27]

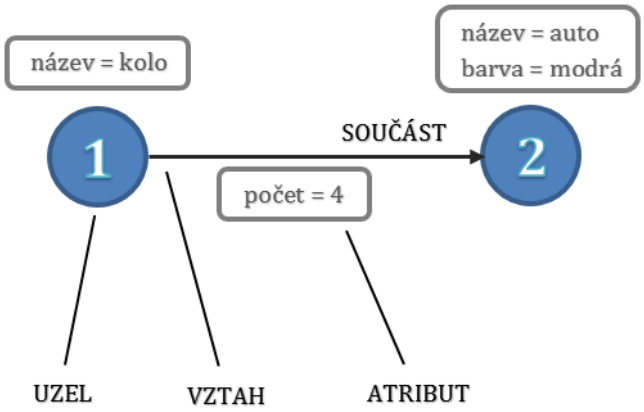


4 Grafová databáze

Grafové databáze jsou jednou z kategorií NoSQL systémů pro ukládání a uchovávání dat, který je reprezentovaný pomocí grafu. Struktura grafové databáze je složena z vrcholů (nodes, vertex) a hran (edges, relationships). Pracují tedy na obdobném principu jako síťové databázové systémy. Jejich vrcholy a hrany představují data, která jsou strukturovaná jako množiny typu klíč/hodnota. Každý vrchol má obvykle několik hran z něj vystupujících a několik hran, které do něj vstupují. Mohou se vyskytnout vrcholy, které nemají žádné vazby mezi sebou. Hrana však má vždy jeden vrchol, ze kterého vystupuje do vrcholu jiného. [7]

Na obrázku 10 je zobrazena struktura grafové databáze. Uzel může být reprezentován nějakým objektem z reálného světa (kolo, auto). Mezi dvěma uzly se nachází vztah (součást). Každý uzel nebo vztah může obsahovat vlastnost (popis objektu). Vztah nebo relace mezi dvěma vrcholy může být přímá nebo nepřímá. U přímé relace směřuje šipka od rodiče směrem k potomkovi. Přímá relace odpovídá relacím v relačních databázích, např. 1:1, 1:N, M:N. U nepřímé relace jsou šipky na obou stranách vrcholů. Říkáme, že vrcholy jsou ve vzájemném vztahu. [15]

Obrázek 11, Struktura grafové databáze, Zdroj: [15]



Tento druh databází je určen zejména k řešení úloh, které je obtížné implementovat v relačních databázích. Grafové databáze jsou vhodné pro ukládání projektů, u kterých záleží na vzájemném propojení entit (uzlů). Jsou ideální pro případy, kde je mnoho různých entit, které mezi sebou mají mnoho vazeb. V těchto případech grafové databáze nahrazují ty relační, které jsou pro tyto případy nevhodné díky nutnosti použití početně náročných operací typu JOIN. Proto grafové databáze své

uplatnění nacházejí hlavně v oblastech sociálních sítí, bioinformatice, elektronickém obchodování, telekomunikačních systémech apod. Grafové databáze jsou hojně využívány pro hledání nejkratší cesty grafu. [15]
Mezi nejpoužívanější grafové databáze patří Neo4j, Titan, Graphite, Giraph a AllegroGraph. [17]

4.1 Neo4j

Neo4j spadá do druhu NoSQL databází. Jedná se o grafový databázový systém, který byl vytvořený společností Neo Technology. Dnes patří mezi nejpopulárnější grafové databáze ve své kategorii. Neo4j je pokládána za první grafovou databázi na světě. Má největší a nejvíce aktivní společenství grafových nadšenců, kteří přispívají do Neo4j ekosystému. Poskytuje rychlý výkon pro čtení a zápis a přitom chrání integritu dat. [9]
Neo4j dnes používají stovky tisíc firem a organizací téměř ve všech průmyslových odvětvích. Případy užití jsou například správa sítí, analytický software, vědecký výzkum, směrování, organizační a projektové řízení, doporučení, sociální sítě a další. Některé vlastnosti činí Neo4j velmi populární mezi uživateli, vývojáři a administrátory. Zhmotnění vztahů v okamžiku vytvoření má za následek absenci sankcí pro složité dotazy runtime. Všechny vztahy v Neo4j jsou stejně důležité a rychlé, aby je bylo možné vytvořit a používat. [9]
Nejpoužívanější grafovou databází na světě je právě Neo4j. [17]

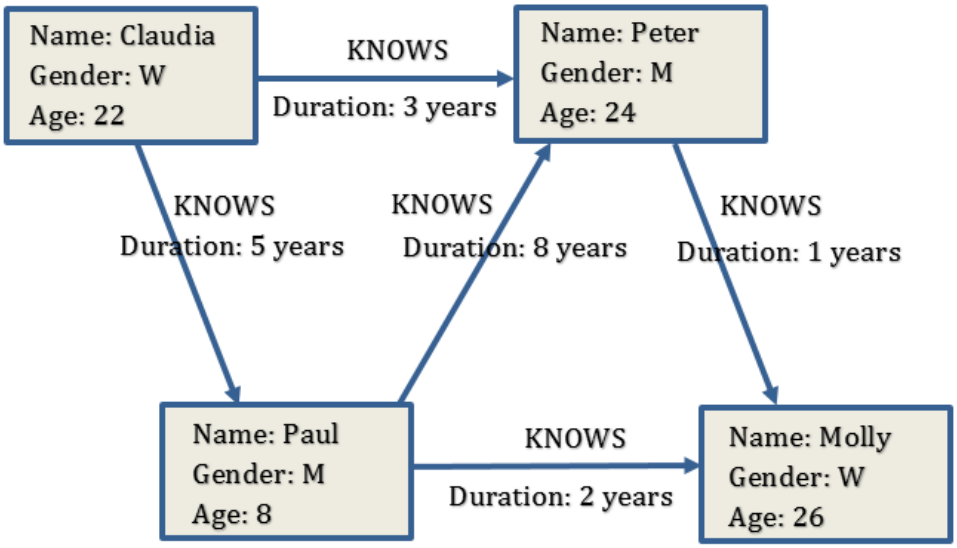
4.1.1 Historie

Grafová databáze Neo4j původně vznikla jako školní projekt několika studentů ve Švédsku. Poté vznikla společnost Neo Technology sídlící v americkém San Franciscu a ve švédském Malmo. Rozvoj začal od roku 2003, veřejně dostupná je tato databáze od roku 2007. Zdrojový kód je k dispozici na GitHub. Jedná se o open source databázi, která je implementována v Javě. První verze byla vydána společností Neo Technology v roce 2010. [9]

4.1.2 Struktura dat

Jedná se o grafovou databázi. Z toho vyplývá, že všechna data jsou ukládána do grafu, který se v případě Neo4j dělí na 4 základní části: nodes (uzly, vrcholy), relationships (vztahy/hrany), labels (štítky) a properties (vlastnosti). Vrcholy reprezentují dané entity a každá z nich může obsahovat jednu nebo více vlastností, které blíže specifikují daný vrchol. Hrana spojuje vždy dva vrcholy a mohou být popsány jednou či vlastnostmi. Každá hrana musí mít typ, prostřednictvím kterého je možné najít určité vrcholy, do kterých směřuje hrana s daným určeným typem. Vlastnosti pak popisují hrany i vrcholy a jsou definovány jako dvojice typu klíč/hodnota. Klíčem je nějaký řetězec a hodnotou může být libovolný datový typ. [9]

Obrázek 12, Struktura Neo4j, Zdroj: [15]



Obrázek 11 popisuje strukturu Neo4j databáze. Jednotlivé vrcholy zde nesou informace o lidech (Name, Gender, Age) a hrany popisují vlastnost „KNOWS“. Kdybychom chtěli najít osoby, které se znají s osobou „Peter“, začali bychom procházet graf od vrcholu s vlastností „Peter“ a hledat vrcholy, které jsou s ním spojeny hranou s vlastností „KNOWS“. Pro tento úkol lze využít více mechanismů (Cypher, Gremlin, Java API).

4.1.2.1 Indexace

Vytvořením indexu zvýšíme výkon při jakémkoli prohledávání uzlů v databázi Neo4j. Jakmile je jednou určena vlastnost uzlu, která se má indexovat, bude se s rozvojem grafu udržovat tento index aktuální. Výkon každé operace, která bude vyhledávat uzly pomocí indexovaných vlastností, se rapidně zvýší. Indexy ovšem nejsou k dispozici ihned po zadání příkazu, začne se vytvářet v pozadí běhu databáze Neo4j. Neo4j ho začne využívat při indexování, jakmile dojde k dokončení jeho tvorby. Záznam o chybách se objeví v log souboru databáze Neo4j. Další možností jak sledovat korektnost stavu indexů je pomocí API rozhraní databáze. [18]

// Vytvoření indexu na vlastnosti name všech vrcholů, které patří do labelu Person
CREATE INDEX ON :Person(name)

4.1.2.2 Omezení

Omezení se určují pomocí podmínek, které by měly určovat, jak mají data v databázi vypadat. Jakákoli změna, která by porušila předepsaný stav je zakázána. Díky omezením dokáže Neo4j pomoci uživateli udržet databázi čistší. Výhodou je, že se o všechna omezení nemusí starat aplikace. [19]

//Vytvoření omezení zajišťující, že v databázi nebude více knih se stejným ISBN
CREATE CONSTRAINT ON (book:Book) ASSERT book.isbn IS UNIQUE

4.1.3 Datové typy

Neo4j automaticky rozlišuje datové typy a sama databáze pozná, zda se jedná o řetězec, číslo nebo logickou hodnotu. Vykonává matematické, logické i porovnávací operace nad daty. Při vytváření databáze tak není nutno specifikovat, o který konkrétní datový typ se jedná. Databáze Neo4j je založena na Javě, tudíž datové typy vychází z datových typů tohoto jazyka. Speciální hodnotou je hodnota NULL, která vyjadřuje neexistenci požadovaného atributu u uzlu. [9]

4.1.4 Dotazovací jazyky

Hlavním dotazovacím jazykem databáze Neo4j je jazyk Cypher. Existují i alternativy pro dotazování v databázi. Jednou takovou alternativou je traverzovací jazyk Gremlin. [9]

- **Cypher** – jazyk primárně určený pro manipulaci s daty (dotazování, vkládání, updatování). Strukturou podobný jazyku SQL.
- **Gremlin** – jazyk určený pro traverzování grafem (procházení jednotlivých částí grafu).

4.1.4.1 Cypher

Jedná se o deklarativní dotazovací jazyk, který využívá Neo4j. Tento jazyk dělá správu grafové databáze srozumitelnější a pochopitelnější pro všechny uživatele. Důležitou vlastnosti tohoto jazyka ten ta, že jde o jazyk deklarativní. Deklarativní jazyk umožní uvést, co hledáte, deklarovat vzor, který byste chtěli vyvolat. Umožňuje větší srozumitelnost údajů, které píšete, což je důležité, protože uživatelé mají tendenci číst své databázové dotazy častěji než je psát. Také umožňuje optimalizaci dotazu, o kterém byste v imperativním přístupu nikdy nevěděli. Dotazovací jazyk Cypher je pro Neo4j velmi důležitý. Stává se tak dostupným a srozumitelným pro větší spektrum uživatelů. Ani většina vývojářů nestojí o to, aby musela spravovat data jazykem, který se jim nelíbí. Jazyk Cypher umožňuje jednodušší nástroj pro správu aplikace. Cypher je inspirován SQL jazykem. Klíčová slova jako např. ORDER BY a WHERE jsou příkladem. [9]
Použití jazyka Cypher můžeme inspirovat na příkladu z Obrázku 11, kde bylo úkolem najít osoby, které mají nějaký vztah s osobou „Peter“ (jsou s ním spojeny hranou).

```
//užití jazyka Cypher na příkladu
START n = node(*)
MATCH n-[:KNOWS]->person
WHERE person.name = 'Peter'
RETURN person.name, person.gender, person.age
```

Příkaz START říká, jaké vrcholy se mají projít a znak „*“ označuje všechny vrcholy. Příkaz MATCH určuje identifikaci hran, kterými má daný příkaz projít. WHERE označuje podmínku. Poslední příkaz RETURN určuje data, která mají být výstupem tohoto dotazu.

4.1.4.2 Gremlin

Gremlin je jazyk pro procházení grafů, který je založený na jazyku Groovy. Jedná se o imperativní dotazovací jazyk, u kterého je nutné přesně definovat postup, jakým se má daný dotaz vykonat. Jeho výhodou je nezávislost na databázi Neo4j. Dotazy jsou poslané od klienta na server a výstup je vrácený jako množina uzlů a hran. Tento způsob dotazování udržuje konzistentnost v celém API. [9]

//užití jazyka Gremlin na příkladu
g.V.has(„name“, „Peter“).out(„KNOWS“)

Tímto příkazem zobrazíme množinu vrcholů, do kterých vede hrana s vlastností „KNOWS“ a vede buď k vrcholu, nebo od vrcholu s vlastností „Peter“. Pomocí příkazu „has“ specifikujeme, jakou podmínku má mít daný dotaz. Pomocí příkazu „out“ vypíšeme informace o vrcholech jako výstup dotazu.

4.1.5 Grafové algoritmy

Když potřebujeme najít nejkratší cestu z jednoho vrcholu do druhého, používá knihovna Neo4j grafové algoritmy, které se v těchto situacích používají. Používanými algoritmy jsou: [21]

- shortest path – pro nalezení nejkratší cesty mezi dvěma uzly s maximálně patnácti hranami
- all paths – pro nalezení všech cest mezi dvěma uzly
- all simple paths – pro nalezení nejjednodušších cest mezi dvěma uzly
- Dijkstra – vhodný pro databázi s cost parametry, popsaný v první kapitole
- A* - pro nalezení nejlevnější (nejnižší možné náklady) cesty mezi dvěma uzly

Když není definováno jinak, defaultně je vždy použitý algoritmus nejkratší cesty (shortest paths). [20]

4.1.6 Systémové požadavky

Protože paměť omezuje velikost grafové databáze a disk rychlost a čtení a zápisu, je nutné znát minimální a doporučené systémové požadavky: [22]

- **Procesor** – minimálně Intel Core i3, doporučeno Intel Core i7
- **Operační paměť** – minimálně 2GB, doporučeno 16-32GB
- **Pevný disk** – minimálně 10GB SATA, doporučeno SSD w/SATA
- **Operační systém** – HP UX, Linux, Windows, Max OS (nutnost Javy a architektury x86)

4.1.7 Licence

Neo4j je software, který je k dispozici ve dvou různých licencích. K dispozici je Community Edition pod licencí GPLv3 (General Public License version 3) pro nekomerční použití, která je zdarma. Enterprise Edition je verze pro komerční využití pod licencí APGLv3 (Affero General Public License), která zdarma není. [23]

Tabulka 3, Přehled licencí Neo4j

Verze	Licence	Popis
Community Edition	GPLv3	Základní databáze, plná podpora ACID
Enterprise Edition	AGPLv3	Pokročilý monitoring, online zálohování, klastrování, rozsah, dostupnost

4.1.8 Využití

Databázi Neo4j je používána zejména sociálními sítěmi, ale IT společnostmi, Identity a Access Managementem atd. Mezi sociální sítě využívající Neo4j patří portály LinkedIn, GameSys, Meetic, LiveStation, Care nebo Snap Interactive. Neo4j najde své využití také ve velkých společnostech jako je HP nebo Cisco. [24]

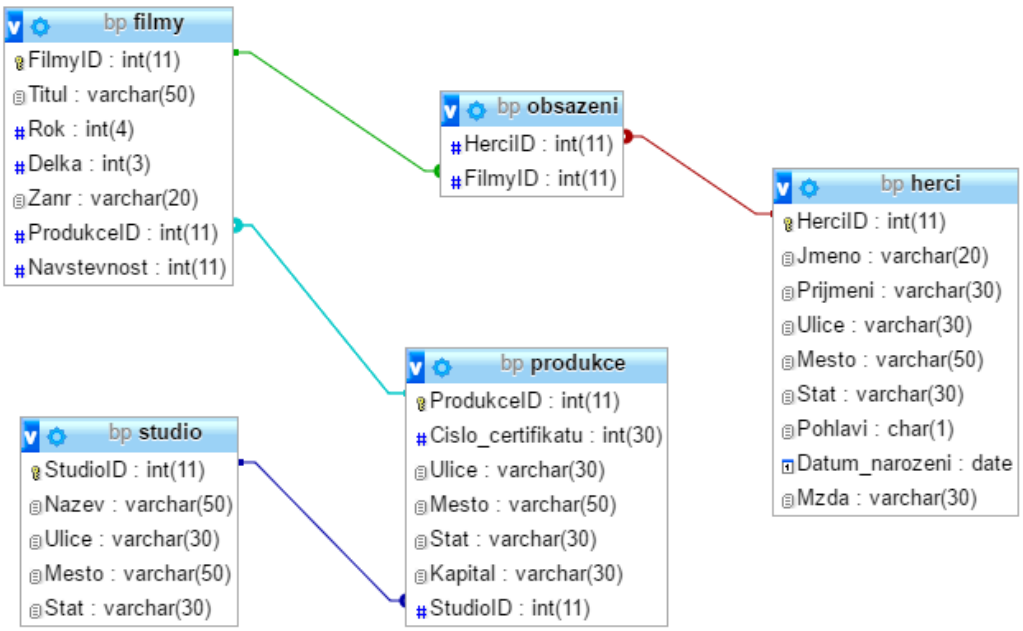
5 Praktická část

Praktická část této práce pojednává o rozdílech mezi relační a grafovou databází ve struktuře dat, rychlosti dotazování při velkém množství dat a porovnávání výhod a nevýhod obou těchto databází. Byl popsán datový model relační databáze a generátorem bylo vygenerováno velké množství dat. Následně byla provedena transformace vybraného příkladu relační databáze implementované v MySQL do grafové databáze implementované do Neo4j. Po transformaci do grafové databáze proběhla testování v obou zmíněných databázích formou spouštění dotazů nad takto velkým množstvím vygenerovaných dat a bylo provedeno šetření, které ukázalo, kdy je více vhodné daný databázový software použít a kdy méně.

5.1 Datový model v MySQL

Testovací databází je zvolena jednoduchá databáze filmů, která by po transformaci měla vystihovat rozdíly mezi relační databází MySQL a grafovou databází Neo4j.

Obrázek 13, Datový model, Zdroj: vlastní zpracování



5.1.1 Popis vybraných tabulek

Herci

Tabulka herců uchovává základní informace o herci. V tabulce se zaznamenává jednoznačný identifikátor herce, který je zároveň primárním klíčem této tabulky, jméno herce, jeho příjmení, bydliště, pohlaví, datum narození a mzda, jakou je ohodnocen hraním ve filmech.

Obsazení

Tabulka obsazení by vůbec nemusela být uvedena v datovém modelu, kdy tabulky herci a filmy mezi sebou neměly vazbu M:N. Z tohoto důvodu musí vzniknout vazební tabulka, která tvoří vazbu mezi dvěma tabulkami a umožňuje několika záznamům z jedné tabulky přiřadit několik záznamů z tabulky druhé. Tabulka proto obsahuje primární klíč z tabulky herci a z tabulky filmy.

Filmy

V tabulce filmy se uchovávají informace v podobě identifikátoru, který je primárním klíčem, titulu filmu, roku natočení filmu, délce filmu v minutách, jeho žánru a návštěvnost. Tabulka obsahuje také cizí klíč z tabulky produkce, který definuje vztah mezi těmito dvěma tabulkami.

Produkce

Tabulka obsahuje ID produkce, číslo certifikátu, které každá produkce musí mít, informace o umístění produkčního štábu, základní kapitál a ID studia jako cizí klíč.

Studio

V tabulce studio obsahuje informace o natáčecích studiích, jako jsou ID studia, název a sídlo samotného studia.

5.2 Generování testovacích dat

Protože by bylo velmi náročné a zdlouhavé plnit databázi postupně skutečnými daty, bylo využito generátoru. Vygenerování samotných testovacích dat je nutné provést před samotnou transformací do grafové databáze Neo4j, jelikož při vytváření struktury grafové databáze budou tato data vyžadována.

5.2.1 DbForge Data Generator for MySQL

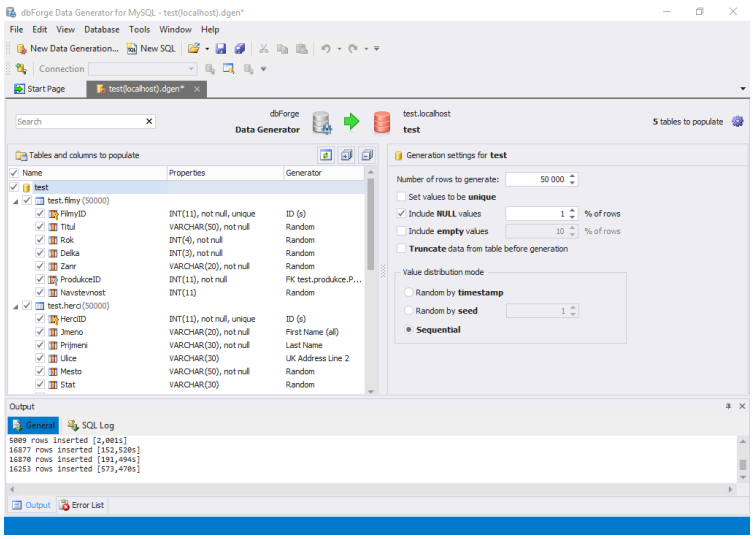
Tento nástroj je určený pro generování testovacích dat nad MySQL databází. Aplikaci je možné stáhnout jako trial verzi na 30 dní zdarma, poté je nutné zakoupení licence. Podporuje všechny verze mysql a existuje verze i pro SQL Server. Poskytuje mnoho užitečných funkcí, např: [25]

- Generování smysluplných dat z nabídky
- Generování dat do SQL nebo CSV souboru
- Generování dat přímo do databáze
- Vytváření vlastních generovaných dat
- Podpora všech verzí MySQL
- Podpora pro SQL Server

5.2.1.1 Postup generování dat

Před samotným generováním dat je třeba na lokálním úložišti vytvořit databázi, kterou generovanými daty budeme chtít naplnit. K tomu je potřeba nainstalovat si databázi MySQL, např. Xampp verze 3.2.2. Po vytvoření samotné databáze je třeba naistalovat dbForge Data Generator for MySQL a spustit. Po spuštění je třeba vytvořit nové připojení, kde budeme dotázáni na adresu hosta, port a jméno samotné databáze. Pokud bude spojení úspěšně navázané, aplikace zobrazí strukturu databáze, ke které jsme se připojili.

Obrázek 14, Ukázka prostředí db Forge Data Generator for MySQL, Zdroj: vlastní zpracování



Po načtení tabulek budeme moci vybrat, který typ dat se má do jednotlivých sloupců vygenerovat. Pro jednoznačné identifikátory lze nastavit, od kterého čísla mají začít a kde přesně skončit. Kdyby byl sloupec některé tabulky pojmenovaný FirstName, tento nástroj by sám rozpoznal a ke sloupci přiřadil, že typ generovaných v tomto sloupci bude křesní jméno.
Před samotným generováním je třeba nastavit, kolik dat chceme do každé tabulky vygenerovat, kolik procent null sloupců má být generátorem vyplněno, zda má probíhat generování sekvenčně (u ID zvolíme) a zda chceme generování do souboru nebo přímo do databáze.
Rychlost generování závisí na počtu generovaných dat a zařízení, na kterém generování probíhá. Generování sto tisíců řádků trvalo v průměru 2 a půl hodiny.

5.3 Datový model v Neo4j

Úkolem bude provést transformaci relační databáze implementované v MySQL do grafové databáze Neo4j.

5.3.1 Mapování z relační databáze

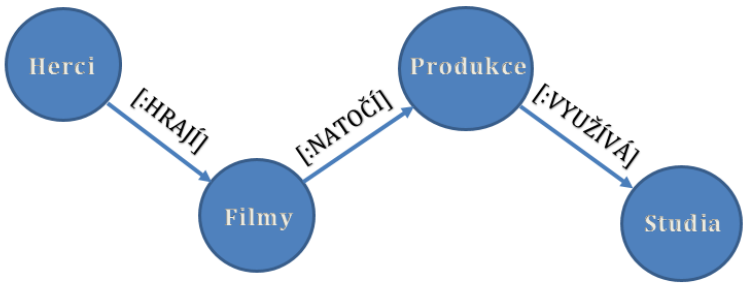
Datový model v relačních databázích je reprezentován tabulkami a data jsou ukládána do sloupců a řádků tabulek. Pro převedení do grafové struktury je nutné definovat podmínky, za jakých bude tato transformace možná.
Každý záznam v tabulce v relačních databázích je v grafovém modelu reprezentován jako vrchol. Název tabulky je označen jako souhrnné označení pro skupinu vrcholů a jednotlivé sloupce tabulky jsou v Neo4j definovány jako vlastnosti, které blíže popisují daný vrchol. Vlastnosti popisující daný vrchol budou uloženy jako dvojice klíč/hodnota. Klíčem hodnoty je vždy název vlastnosti odpovídající názvu sloupce v tabulce relační databáze a hodnotou jsou data primitivních datových typů (varchar, inrteger, date apod.), která odpovídají řádku v datovém modelu relační databáze. Mezi tabulkami, které v relační databázi představují relace, v grafovém schématu jsou reprezentovány pojmenovanými hranami, kde název přesně vystihuje její význam. V relačních databázích rozlišujeme tyto typy relací: [28]

- **1:1** – tato relace v relačních databázích znamená, že jednomu záznamu v jedné tabulce přiřazuje jeden záznam z tabulky druhé. V grafových databázích odpovídá tento vztah pojmenované hraně.
- **1:N** – v relačních databázích říká, že jednomu záznamu z jedné tabulky lze přiřadit více záznamů z jiné tabulky. Tuto relaci lze v grafové struktuře reprezentovat orientovanou hranou směřující z jednoho vrcholu ke druhému.
- **M:N** – v tomto případě se v relačních databázích vytváří vazební tabulka (v našem případě tabulka obsazeni), která tvoří relace M:N mezi dvěma tabulkami. Relací reprezentujeme skutečnost, že několika záznamům z jedné tabulky lze přiřadit více záznamům z druhé tabulky. V grafové databázi lze tento problém vyřešit dvojím způsobem. Můžeme využít neorientované hrany, která bude jako vlastnosti obsahovat data z vazební tabulky. Tomuto problému se můžeme vyhnout již před samotnou transformací do grafové databáze. Při exportování jedné tabulky lze sloučit tabulku s vazební tabulkou a poté při vytváření hran porovnávat námi exportovaná data.

5.3.2 Schéma grafového modelu

Na obrázku 15 je zobrazené schéma datového modelu grafové databáze, která bude implementována do Neo4j s využitím jazyka Cypher. Graf je transformací datového modelu relační databáze. Graf je tvořený počátečním vrcholem Herci, který bývá také často označován jako kořenový, ze kterého jsou postupně vytvářeny další vrcholy.

Obrázek 15, Návrh grafové databáze, Zdroj: vlastní zpracování



5.3.3 Přípravy pro implementaci grafové databáze

Data jsou vygenerována a relační databáze implementována v MySQL. Nyní je nutné provést SQL dotaz na vyexportování a sloučení tabulek herci a obsazeni do CSV souboru. Poté vyexportujeme každou tabulku do jednotlivých CSV souborů, což je možné provést v uživatelském rozhraní Xampp.

```
//Export vazební tabulky obsazeni a tabulky herci do CSV souboru
SELECT 'HerciID', 'Jmeno', 'Prijmeni', 'Ulice', 'Mesto', 'Stat', 'Pohlavi',
'Datum_narozeni', 'Mzda', 'HerciID', 'FilmyID' UNION ALL
SELECT * FROM herci
INNER JOIN obsazeni ON obsazeni.HerciID = herci.HerciID
INTO OUTFILE 'C:/Users/probookPA/Desktop/CSV_BP/herci.csv'
FIELDS TERMINATED BY ',' ENCLOSED BY '"' ESCAPED BY '\\' LINES TERMINATED
BY '\n'
```

5.4 Implementace datového modelu v Neo4j

Vytvoření grafové databáze je nejdůležitější částí celého testování, aby bylo s čím porovnávat datový model relační databáze.

5.4.1 Použité technologie

Při implementaci datového modelu v grafové databázi Neo4j byly použity tyto technologie:

- **Neo4j Community Edition 2.3.2** – využití jazyka Cypher
- **Xampp Control Panel 3.2.2** – využití databáze MySQL

5.4.2 Popis implementace

Před samotnou implementací se musíme ujistit, že máme data z MySQL databáze vyexportovaná v CSV souborech. Pokud jsme tak učinili, můžeme začít s importováním CSV souborů prostřednictvím jazyka Cypher. Příkazem načteme CSV soubor na disku počítače a postupně z něj budeme číst. Je nutné, aby soubor CSV

obsahoval hlavičku, tzn. názvy jednotlivých sloupců. Import by jinak takto nebyl možný.

//Vytvoření vrcholů a jejich vlastností, přiřazených do labelu Herci
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM
"file:///C:/Users/probookPA/Desktop/CSV_BP/herci.csv" AS row
MERGE (h:Herci {HerciID: row.HerciID, Jmeno: row.Jmeno, Prijmeni: row.Prijmeni, Mesto: row.Mesto, Pohlavi: row.Pohlavi, Datum_narozeni: row.Datum_narozeni, Mzda: row.Mzda})

//Vytvoření vrcholů a k nim přiřazených vlastností patřící do labelu Filmy
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM
"file:///C:/Users/proobokPA/Desktop/CSV_BP/filmy.csv" AS row
MERGE (f:Filmy {FilmyID: row.FilmyID, Titul: row.Titul, Rok: row.Rok, Delka: row.Delka, Zanr: row.Zanr, Navstevnost: row.Navstevnost})

//Vytvoření vrcholů a k nim přiřazených vlastností patřící do labelu Produkce
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM
"file:///C:/Users/probookPA/Desktop/CSV_BP/produkce.csv" AS row
MERGE (p:Produkce {ProdukceID: row.ProdukceID, Cislo_certifikatu: row.Cislo_certifikatu, Mesto: row.Mesto, Stat: row.Stat, Kapital: row.Kapital})

//Vytvoření vrcholů a vlastností, které budou přiřazeny do labelu Studia
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM
"file:///C:/Users/probookPA/Desktop/CSV_BP/studio.csv" AS row
MERGE (s:Studia {StudioID: row.StudioID, Nazev: row.Nazev, Ulice: row.Ulice, Mesto: row.Mesto, Stat: row.Stat})

Před samotným vytvářením hran (vztahů mezi jednotlivými vrcholy) je dobré udělat indexy na právě vytvořených vrcholech. Zajistíme tak rychlejší vyhledávání při vytváření hran.

```
//Vytvoření indexů
CREATE INDEX ON :Herci(HerciID)
CREATE INDEX ON :Filmy(FilmyID)
CREATE INDEX ON :Produkce(ProdukceID)
CREATE INDEX ON :Studia(StudiaID)
```

U vytváření hran je stejně jako v případě vytváření vrcholů nezbytné, aby CSV soubor na prvním řádku obsahoval hlavičku, tzn. názvy jednotlivých sloupců. Pro tyto účely využijeme stejný soubor jako při generování vrcholů. Při vytváření příkazu je nutné, aby zde byly porovnávány ID těch vrcholů, které chceme hranou spojit. Na závěr dotazu je třeba umístit název hrany a určit, zda má být hrana orientovaná. Pokud ano, je nezbytné určit odkud kam. Podle datového modelu grafové databáze bude v prvním případě hrana orientovaná od vrcholů Herci k vrcholům Filmy.

```
//Vytvoření hran [:HRAJE]
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM
"file:///C:/Users/probookPA/Desktop/CSV_BP/herci.csv" AS row
MATCH (h:Herci {HerciID: row.HerciID})
MATCH (f:Filmy {FilmyID: row.FilmyID})
MERGE (h)-[:HRAJE]->(f)
```

```
//Vytvoření hran [:NATOCI]
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM
"file:///C:/Users/probookPA/Desktop/CSV_BP/filmy.csv" AS row
MATCH (f:Filmy {FilmyID: row.FilmyID})
```

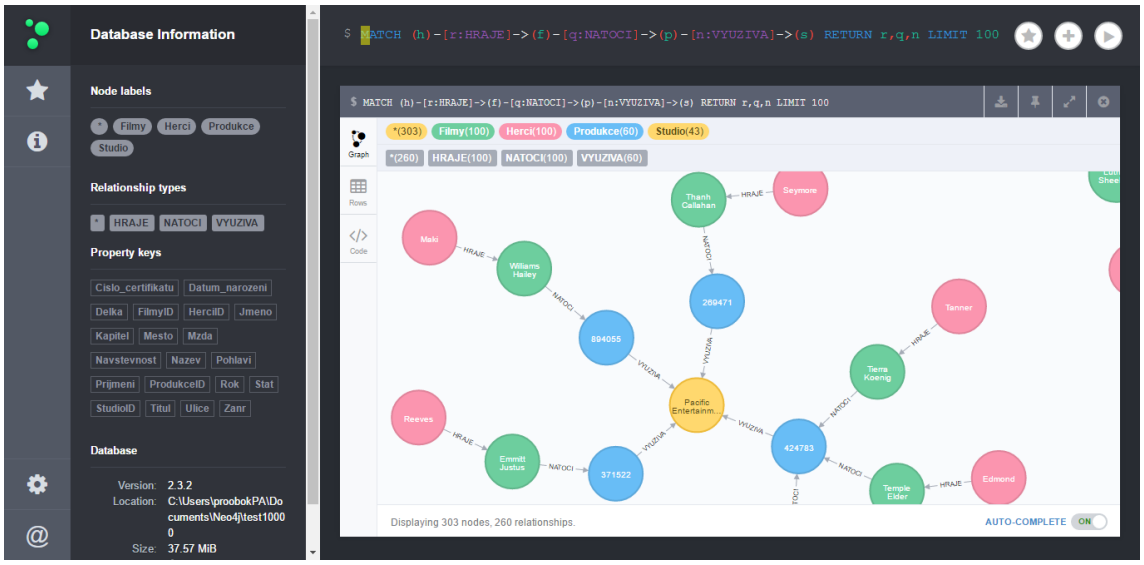
MATCH (p:Produkce {ProdukceID: row.ProdukceID})
MERGE (f)-[:NATOCI]->(p)

//Vytvoření hran [:VYUZIVA]
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM
"file:///C:/Users/probookPA/Desktop/CSV_BP/produkce.csv" AS row
MATCH (p:Produkce {ProdukceID: row.ProdukceID})
MATCH (s:Studio {StudioID: row.StudioID})
MERGE (p)-[:VYUZIVA]->(s)

5.4.3 Popis prostředí Neo4j

Každá nová verze prostředí Neo4j s sebou přináší něco nového. Datový model naší databáze ve verzi Neo4j 2.3.2 je možno vidět na obrázku 16.

Obrázek 16, Popis prostředí Neo4j, Zdroj: vlastní zpracování



Dotazy nad grafovou databází Neo4j v jazyce Cypher se píší do příkazového řádku nahoře. Pokud je dotaz takový, že dovolí výsledek zobrazit pomocí grafu, bude to vypadat jako na obrázku 16. Výsledky dotazu lze zobrazit i pomocí tabulky ve formě řádků a sloupců jako v SQL jazyce. Každý dotaz lze přidat do kategorie oblíbených dotazů. Samotný graf jako výsledek dotazu si lze upravovat podle libosti. Je možné

změnit barvy jednotlivých vrcholů, jejich velikost, která z vlastností vrcholu má být použita jako titulek vrcholu atd. Pokud chceme výsledek dotazu zobrazit jako graf, je dobré dotaz omezit na nějaký počet vrcholů klíčovým slovem LIMIT, protože pokud máme databázi o velkém množství dat, tak trvá celkem dlouho jeho samotné vykreslování. Tyto věci jsou závislé na množství dat a fyzickém stroji, na kterém je databáze spuštěna. Graf je možné vyexportovat do CSV, PNG, SVG nebo JSON souboru.

V panelu na levé straně je možné vidět labely (skupiny) jednotlivých vrcholů, vztahů i vlastností. Neo4j umožňuje zjednodušit práci v prostředí v podobě předpřipravených tlačítek na vytvoření vrcholu, vztahu apod., nicméně znalost jazyka Cypher je pro toto prostředí klíčové. Graf je možné pojmenovat a nastavit vzhled prostředí (Normal, Dark, Outline). V nastavení je možné změnit hodnotu pole Max Execution Time, Max Frames, Max History, Max Neighbors, Max Rows, Max Raw Size a Refresh Interval.

K dispozici je zobrazení příkladů databází. Je zde možné zobrazit známou Northwind databázi, která je hojně využívána v obchodním odvětví. Neo4j manuál je v prostředí též volně dostupný.

5.5 Testování

V této kapitole proběhne samotné testování relační a grafové databáze. Cílem bylo zjistit, který z testovacích systémů je rychlejší na provádění dotazů. Testování patří mezi nejdůležitější části této práce. Má pomoci rozhodnout, který databázový systém je vhodnější z hlediska rychlosti vykonání dotazů. Dotaz v grafové databázi Neo4j by měl být vykonán rychleji než dotaz v relační databázi MySQL.

5.5.1 Konfigurace počítače

Samotný proces testování probíhal na lokálním úložišti. V průběhu testování bylo třeba změnit nastavení databázových serverů a z hlediska rychlejšího a snazšího přístupu bylo zvoleno toto úložiště. Testování proběhlo na notebooku s touto hardwarovou konfigurací:

- **Procesor** – Intel Core i5-3230M CPU, 2.60 GHz
- **Paměť** – 8 GB
- **Disk** – 1 TB
- **Operační systém** – Windows 10 Professional
- **Typ systému** – 64bitový operační systém

5.5.2 Rychlost vykonání dotazů

Testována byla právě rychlost vykonaných dotazů. Byly sestaveny tři dotazy a ty byly postupně vykonány nad relační databází MySQL a poté nad grafovou databází Neo4j. U každého z těchto dotazů je detailně popsáno, co dělá, syntaxe obou dotazovacích jazyků (SQL, Cypher) a v poslední řadě je uvedena tabulka, kde jsou hodnoty naměřených rychlostí dotazů pro relační databázi MySQL a grafovou databázi Neo4j.

Cestování bylo provedeno tak, že každý dotaz byl vykonán na datovém modelu pro různý počet záznamů, nejprve pro 10 000 záznamů, následně pro 50 000 záznamů, poté pro 100 000 záznamů a v poslední řadě pro 500 000 záznamů v každé tabulce relační databáze. Dotaz byl vykonán pětkrát pro různý počet záznamů.

Před samotným testováním bylo možné určit, jakým způsobem měřit čas vykonání dotazu. První možností bylo vybrat hodnotu času vykonání dotazu, tzv. executing time, která představuje čas, za který se vykoná určitý dotaz, ačkoli bez zobrazení jeho výsledku. Druhou volbou byl čas vykonání dotazu a následné zobrazení výsledků, tzv. fetch time, který představuje tento čas. Při tomto testování byla vybrána hodnota, která představuje vykonání dotazu i následné zobrazení výsledků, tedy fetch time.

Dotazy v relační databázi byly nejprve prováděny pouze s indexy cizích klíčů a s definovanými primárními klíči. Nicméně naměřené výsledné hodnoty byly značně veliké, proto bylo nutné vytvořit podpůrnou indexaci všech atributů u klíčového slova WHERE u každého ze tří testovacích dotazů. Z tabulek níže je vidno, že tento krok zapříčinil nesrovnatelně rychlejší dotazování nad relační databází MySQL.

Jako nástroj pro toto testování v relační databázi byl použit phpMyAdmin, který se pro práci s databází MySQL standardně používá. V grafové databázi Neo4j byl pro testování rychlosti vykonání dotazu zvolen nástroj Web admin.

5.5.2.1 Dotaz č. 1

Dotaz vybere titul a rok natočení filmů, ve kterých hrál herec s příjmením Smith. Výsledky budou seřazeny podle titulu filmu.

MySQL

SELECT *filmy.FilmyID, filmy.Titul, filmy.Rok*

FROM *filmy*

INNER JOIN *obsazeni* ***ON***

obsazeni.FilmyID = filmy.FilmyID

INNER JOIN *herci* ***ON***

herci.HerciID = obsazeni.HerciID

WHERE *herci.Prijmeni = "Smith"*

ORDER BY *filmy.Titul*

Tabulka 4, Výsledky měření prvního dotazu v MySQL bez indexace sloupce u příkazu WHERE

	Jednotlivé kroky (s)					(s)
Počet záznamů	1	2	3	4	5	Průměr
10 000	0,361	0,300	0,294	0,330	0,318	0,321
50 000	1,917	1,882	1,837	1,845	1,751	1,846
100 000	120,124	112,250	118,875	109,125	114,453	114,965
500 000	x	x	x	x	x	x

//Vytvoření indexu u příkazu WHERE

CREATE INDEX Prijmeni ON herci (Prijmeni)

Tabulka 5, Výsledky měření prvního dotazu v MySQL s indexací sloupce u příkazu WHERE

	Jednotlivé kroky (s)					(s)
Počet záznamů	1	2	3	4	5	Průměr
10 000	0,091	0,082	0,118	0,093	0,084	0,094
50 000	0,430	0,455	0,481	0,478	0,389	0,447
100 000	1,212	1,008	1,145	0,982	1,142	1,098
500 000	2,659	3,012	2,845	2,621	2,780	2,783

Neo4j

START *h = node(*)*

MATCH *(f:Filmy)<-[:HRAJE]-(h:Herci)*

WHERE *h.Prijmeni = "Smith"*

RETURN *f.FilmyID, f.Titul, f.Rok*

ORDER BY *f.Titul*

Tabulka 6, Výsledky měření prvního dotazu v Neo4j

	Jednotlivé kroky (s)					(s)
Počet záznamů	1	2	3	4	5	Průměr
10 000	0,090	0,086	0,094	0,129	0,081	0,096
50 000	0,292	0,258	0,308	0,240	0,212	0,262
100 000	0,420	0,435	0,475	0,401	0,408	0,428
500 000	0,650	0,689	0,654	0,664	0,643	0,660

5.5.2.2 Dotaz č. 2

Dotaz vybere údaje o hercích s příjmením Smith, kteří hráli ve filmech spolupracujících se studiem Smart Media Group a zobrazí požadované údaje. Výsledek tohoto dotazu bude seřazen podle názvu filmu.

MySQL

SELECT *herci.HerciID, herci.Jmeno, herci.Prijmeni, herci.Datum_narozeni, filmy.FilmyID, filmy.Titul, filmy.Rok*

FROM *filmy*

INNER JOIN *obsazeni* ***ON***

obsazeni.FilmyID = filmy.FilmyID

INNER JOIN *herci* ***ON***

herci.HerciID = obsazeni.HerciID

INNER JOIN *produkce* ***ON***

produkce.ProdukceID = filmy.ProdukceID

INNER JOIN *studio* ***ON***

studio.StudioID = produkce.StudioID

WHERE *herci.Prijmeni = "Smith" and studio.Nazev = "Smart Media Group"*

ORDER BY *filmy.Titul*

Tabulka 7, Výsledky měření druhého dotazu v MySQL bez indexace sloupců u příkazu WHERE

	Jednotlivé kroky (s)					(s)
Počet záznamů	1	2	3	4	5	Průměr
10 000	0,786	0,802	0,794	0,810	0,800	0,798
50 000	3,104	3,048	2,941	3,042	2,955	3,018
100 000	300,548	282,450	293,640	294,550	289,974	292,232
500 000	x	x	x	x	x	x

//Vytvoření indexů u příkazu WHERE

CREATE INDEX Prijmeni ON herci (Prijmeni)

CREATE INDEX Nazev ON studio (Nazev)

Tabulka 8, Výsledky měření druhého dotazu v MySQL s indexací sloupců u příkazu WHERE

	Jednotlivé kroky (s)					(s)
Počet záznamů	1	2	3	4	5	Průměr
10 000	0,120	0,118	0,109	0,129	0,105	0,116
50 000	0,628	0,589	0,603	0,622	0,655	0,619
100 000	1,548	1,490	1,660	1,550	1,483	1,546
500 000	4,892	4,576	5,311	5,020	4,984	4,957

Neo4j

START *h = node (*)*

MATCH *(h:Herci) -[:HRAJE]->(f:Filmy)-[:NATACI]->(p:Produkce)-[:VYUZIVA]->(s:Studio)*

WHERE *h.Prijmeni = "Smith" and s.Nazev = "Smart Media Group"*

RETURN *h.HerciID, h.Jmeno, h.Prijmeni, h.Datum_narozeni, f.FilmyID, f.Titul, f.Rok*

ORDER BY *f.Titul*

Tabulka 9, Výsledky měření druhého dotazu v Neo4j

	Jednotlivé kroky (s)					(s)
Počet záznamů	1	2	3	4	5	Průměr
10 000	0,065	0,049	0,058	0,052	0,049	0,055
50 000	0,242	0,212	0,350	0,196	0,259	0,252
100 000	0,493	0,485	0,510	0,472	0,498	0,491
500 000	0,754	0,770	0,697	0,683	0,712	0,723

5.5.2.3 Dotaz č. 3

Dotaz vybere základní údaje o produkcích a studiích, kde název studia je Federal Systems Inc. a číslo certifikátu produkce je menší než 1000.

MySQL

SELECT *produkce.ProdukceID, produkce.Cislo_certifikatu, studio.StudioID,*
studio.Nazev
FROM *produkce*
INNER JOIN *studio* ***ON***
studio.StudioID = produkce.StudioID
WHERE *studio.Nazev ="Federal Systems Inc." and produkce.Cislo_certifikatu < 1000*

Tabulka 10, Výsledky měření třetího dotazu v MySQL bez indexace sloupců u příkazu WHERE

	Jednotlivé kroky (s)					(s)
Počet záznamů	1	2	3	4	5	Průměr
10 000	0,1998	0,1782	0,1668	0,2003	0,1811	0,1852
50 000	1,0015	0,9842	0,9979	0,9450	0,9680	0,9793
100 000	39,6537	40,1687	37,4852	37,1258	37,4520	38,3771
500 000	x	x	x	x	x	x

//Vytvoření indexu u příkazu WHERE
CREATE INDEX Nazev ON studio (Nazev)
CREATE INDEX Cislo_certifikatu ON produkce (Cislo_certifikatu)

Tabulka 11, Výsledky měření třetího dotazu v MySQL s indexací sloupců u příkazu WHERE

	Jednotlivé kroky (s)					(s)
Počet záznamů	1	2	3	4	5	Průměr
10 000	0,085	0,073	0,064	0,094	0,080	0,079
50 000	0,350	0,299	0,310	0,285	0,330	0,315
100 000	0,780	0,812	0,764	0,750	0,809	0,783
500 000	1,620	1,688	1,587	1,610	1,599	1,621

Neo4j

START *p = node(*)*

MATCH *(p:Produkce)-[:VYUZIVA]->(s:Studio)*

WHERE *s.Nazev = "Federal Systems Inc." and p.Cislo_certifikatu < "1000"*

RETURN *p.ProdukceID, p.Cislo_certifikatu, s.StudioID, s.Nazev*

Tabulka 12, Výsledky měření třetího dotazu v Neo4j

	Jednotlivé kroky (s)					(s)
Počet záznamů	1	2	3	4	5	Průměr
10 000	0,064	0,062	0,058	0,063	0,064	0,062
50 000	0,209	0,270	0,264	0,314	0,264	0,264
100 000	0,440	0,394	0,412	0,388	0,364	0,400
500 000	0,618	0,581	0,610	0,628	0,575	0,602

5.6 Zhodnocení výsledků

Byl otestován požadavek na datový model relační databáze MySQL a nerelační grafové databáze Neo4j. Hlavním požadavkem byla rychlost vykonávání dotazů. Testování rychlosti vykonání dotazů probíhalo tak, že každý z dotazů byl opakovaně vykonán nad různým počtem záznamů. Celkový počet záznamů v relační databázi je shrnut v tabulce 13 a celkový počet jednotlivých entit v grafové databázi je vyobrazen v tabulce 14.

Tabulka 13, Celkový počet záznamů v relační databázi MySQL

Počet záznamů v každé tabulce	10 000	50 000	100 000	500 000
Celkový počet záznamů	50 000	250 000	500 000	2 500 000

Tabulka 14, Celkový počet entit v grafové databázi Neo4j

Počet záznamů v každé tabulce	10 000	50 000	100 000	500 000
Celkový počet vrcholů	40 000	200 000	400 000	2 000 000
Celkový počet hran	30 000	150 000	300 000	1 500 000
Celkový počet vlastností	240 000	1 200 000	2 400 000	12 000 000

U obou databázových systémů byla provedena optimalizace prostřednictvím indexů z důvodu rychlejšího dotazování. V relační databázi MySQL byly definovány primární klíče a vytvořeny indexy na cizích klíčích. Testování ukázalo, že relační databáze si s dotazováním nad takto implementovaným datových modelem u počtu 500 000 záznamů nedokáže poradit. Hodnoty testování u 100 000 záznamů byly také velmi vysoké, proto bylo třeba učinit kroky, které zrychlí dotazování nad relační databází. Proto byly vytvořeny indexy u sloupců tabulek, které byly testovány jako podmínky u klíčového slova WHERE. Dotazování nad takovouto strukturou relační databáze ukázalo o mnoho přijatelnější výsledky, nežli tomu bylo v předchozím případě. U grafové databáze Neo4j byly vytvořeny indexy pro všechny vrcholy, které hrály zásadní roli při vytváření hran mezi vrcholy. Indexování by mělo sloužit zejména pro rychlejší vykonávání dotazů nad databázemi.

Z výsledků naměřených hodnot dotazů je vidět, že vykonání dotazů v grafové databázi Neo4j je rychlejší než v případě relační databáze MySQL. Rozdíly jsou vidět až při větším počtu záznamů, tedy 100 000 a více. Tyto rozdíly se výrazněji projevují u dotazů, kde je potřeba projít více tabulek, tedy při použití klíčového slova JOIN v relačních databázích. U dotazu č. 2 byl tentýž dotaz v Neo4j téměř 7x rychlejší než dotaz spuštěný nad relační databází MySQL. Čím více dat z jiných entit je třeba projít, tím delší je fetch time v relační databázi. Detailní pohled na naměřené hodnoty testování vykonaných dotazů v konkrétním databázovém systému pro 500 000 záznamů každé tabulky v relační databázi je popsán v tabulce 15.

Tabulka 15, Zrychlení všech dotazů pro 500 000 záznamů v každé tabulce

	MySQL (s)	Neo4j (s)	Zrychlení
Dotaz č. 1	2,783	0,660	4,22x
Dotaz č. 2	4,957	0,723	6,86x
Dotaz č. 3	1,621	0,620	2,61x

U grafové databáze Neo4j je i proto snadnější rozšíření datového modelu než u relační databáze MySQL. V Neo4j stačí vytvořit nový vrchol, hranou ho připojit k jinému vrcholu a případně definovat vlastnosti daného vrcholu nebo hrany. Tato změna je však v relačních databázích často komplikovanější.

Z dosažených výsledků je patrné, že v rychlosti vykonávání dotazů je vhodnější použít grafovou databázi Neo4j. Podle mého názoru se Neo4j jeví jako vhodnější použití datového úložiště v mnoha případech užití. Grafová databáze Neo4j má velké uplatnění se na trhu databází a bude mít velkou budoucnost.

6 Závěr

V práci byly představeny základní koncepty teorie grafů, které jsou stěžejní a zásadní pro další problematiku této práce. Poté byly představeny NoSQL databáze, které se v dnešní době hojně využívají v sociálních sítích i jiných oblastech. Jedním z typů NoSQL jsou grafové databáze, které byly v této práci také popsány. Hlavním tématem však byla grafová databáze Neo4j. Byly vysvětleny základní charakteristiky této databáze, syntaxe jazyků, použití algoritmů apod.

Byl vytvořen datový model, který byl implementován do relační databáze MySQL a nerelační grafové databáze Neo4j. Obě databáze byly testovány dotazováním se nad velkým množstvím dat. Hlavním parametrem testování byla rychlost vykonávání dotazů nad oběma databázemi. Podle zhodnocení dosažených výsledků (viz předchozí kapitola) byla zvolena jako vhodnější databáze grafová databáze Neo4j, která dosáhla lepších výsledků v testování než relační databáze MySQL.

Tato práce byla velice zajímavá. Vyzkoušel jsem si práci s relativně novou a mladou technologií, která se řadí do skupiny NoSQL databází. V současné době se NoSQL databáze dále rozvíjejí a do budoucna se mohou stát hojně využívanou databází v mnoha případech a ohledech. Hlavním důvodem je flexibilní datové schéma, na které relační databáze nestačí, a stále zvyšující se objem dat.

7 Seznam použité literatury

[1] DEMEL, Jiří. Grafy a jejich aplikace. 1. vydání. Praha 5: nakladatelství Akademie věd České republiky, 2002, 257 stran, ISBN 80-200-0990-6

[2] MATOUŠEK, Jiří, NEŠETŘIL, Jaroslav. Kapitoly z diskrétní matematiky. 2. vydání. Praha: nakladatelství Karolinum, 2000, 377 stran, ISBN 80-246-0084-6

[3] MILKOVÁ, Eva. Teorie grafů a grafové algoritmy. 1. vydání. Hradec Králové: nakladatelství Gaudeamus, 2013, 200 stran, ISBN 978-80-7435-267-6

[4] KOLÁŘ, Josef. Teoretická informatika. 2. vydání. Praha: Česká informatická společnost, 2004, 205s, ISBN 80-900853-8-5

[5] HOLUBOVÁ, Irena, KOSEK, Jiří, MINAŘÍK, Karel, NOVÁK, David. Big Data a NoSQL databáze. 1. vydání. Praha: Grada Publishing, a.s., 2015, 288 stran, ISBN 978-80-247-5938-8

[6] POKORNÝ, Jaroslav, VALENTA, Michal. Databázové systémy. 1. vydání. Praha: nakladatelství ČVUT, 2013, 274 stran, ISBN 978-80-01-05212-9

[7] ROBINSON, Ian, WEBBER, Jim, EIFREM, Emil. Graph Databases. 2. vydání. O'Reilly Media, 2013, 224 stran, ISBN 978-1-4493-5626-2. Dostupné také z: http://graphdatabases.com/.

[8] NOSQL-DATABASE, NoSQL, [online]. 2011, [cit. 2016-21-1]. Dostupné z www: http://nosql-database.org/

[9] VAN BRUGGEN, Rik. Learning Neo4j. 1.vydání. Packt publishing, 2014, 222 stran, ISBN ISBN 978-1-84951-716-4. Dostupné také z: http://neo4j.com/book-learning-neo4j/.

[10] COUCHBASE, Why NoSQL? , [online]. 2015, [cit. 2015-31-11]. Dostupné z www: http://www.couchbase.com/nosql-resources/what-is-no-sql

[11] WORDPRESS, NoSQL for ERP?, [online]. 2012, [cit. 2016-23-1]. Dostupné z www: https://smist08.wordpress.com/tag/cap-theorem/

[12] AYENDE, That No SQL Thing – Key/Value stores, [online]. 2010, [cit. 2016-21-1]. Dostupné z www: https://ayende.com/blog/4449/that-no-sql-thing-key-value-stores

[13] AYENDE, Column (Family) Databases, [online]. 2010, [cit. 2015-31-11]. Dostupné z www: http://ayende.com/blog/4500/that-no-sql-thing-column-family-databases

[14] MONGODB, Document databases, [online]. 2015, [cit. 2015-31-11]. Dostupné z www: https://www.mongodb.com/document-databases

[15] INFOQ, Graph Databases, NoSQL and Neo4j, [online]. 2010, [cit. 2016-22-3]. Dostupné z www: http://www.infoq.com/articles/graph-nosql-neo4j

[16] VOHO, Teorie grafů, [online]. 2008, [cit. 2015-31-11]. Dostupné z www: htto://voho.cz/wiki/graf/

[17] DB-ENGINES, DB-Engines Ranking, [online]. 2013, [cit. 2016-22-3]. Dostupné z www: http://db-engines.com/en/ranking

[18] NEO4J, Indexes, [online]. 2016, [cit. 2016-23-3]. Dostupné z www: http://neo4j.com/docs/stable/query-schema-index.html

[19] NEO4J, Constraints, [online]. 2016, [cit. 2016-23-3]. Dostupné z www: http://neo4j.com/docs/stable/query-constraints.html

[20] NEO4J, Graph Algorithms, [online]. 2016, [cit. 2016-23-3]. Dostupné z www: http://neo4j.com/docs/stable/rest-api-graph-algos.html

[21] NEO4J, Graph Algorithms, [online]. 2016, [cit. 2016-23-3]. Dostupné z www: http://neo4j.com/docs/stable/graph-algo.html

[22] NEO4J, Systém Requirements, [online]. 2016, [cit. 2016-23-3]. Dostupné z www: http://neo4j.com/docs/stable/deployment-requirements.html#_software

[23] NEO4J, About Neo4j Licenses, [online]. 2016, [cit. 2016-23-3]. Dostupné z www: http://neo4j.com/licensing/

[24] NEO4J, Custumers, [online]. 2016, [cit. 2016-23-3]. Dostupné z www: http://neo4j.com/customers/

[25] DEVART, DbForge Data Generator for MySQL Features, [online]. 2016, [cit. 2016-2-4]. Dostupné z www: https://www.devart.com/dbforge/mysql/data-generator/features.html

[26] LINKEDIN, A Comprehensice Analysis – NoSQL vs RDBMS, [online]. 2015, [cit. 2-4-2016]. Dostupné z www: https://www.linkedin.com/pulse/comprehensive-analysis-nosql-vs-rdbms-rassul-fazelat

[27] 3PILLARGLOBAL, Selection criteria for NoSQL databases part III, [online]. 2015, [cit. 2-4-2016]. Dostupné z www: http://www.3pillarglobal.com/insights/selection-criteria-for-nosql-database

[28] NEO4J, From relational to neo4j, [online]. 2015, [cit. 2-4-2016]. Dostupné z www: http://neo4j.com/developer/graph-db-vs-rdbms/

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor: **Pavel Ardolf**

Studium: I1301394

Studijní program: B1802 Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Název bakalářské práce: **Grafové databáze Neo4j**

Název bakalářské práce AJ: Graph database Neo4j

Cíl, metody, literatura, předpoklady:

Cílem práce je seznámit se s grafovou databází Neo4j implementovanou do vybraného příkladu využití a porovnat algoritmy pro vyhledávání. 1. Úvod 2. Teoretická část a. NoSQL b. Graf c. Grafová databáze 3. Praktická část v Neo4j 4. Závěr 5. Zdroje

MILKOVÁ, Eva. Teorie grafů a grafové algoritmy. 1.vydání. Hradec Králové: nakladatelství Gaudeamus, 2013, 200 stran, ISBN 978-80-7435-267-6 ROBINSON, Ian, WEBBER, Jim, EIFREM, Emil. Graph Databases. 2.vydání. O'Reilly Media, 2013, 224 stran, ISBN 978-1-4493-5626-2. Dostupné také z: http://graphdatabases.com/. VAN BRUGGEN, Rik. Learning Neo4j. 1.vydání. Packt publishing, 2014, 222 stran, ISBN ISBN 978-1-84951-716-4. Dostupné také z: http://neo4j.com/book-learning-neo4j/.

Garantující pracoviště: Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu

Vedoucí práce: Ing. Barbora Tesařová, Ph.D.

Oponent: Ing. Andrea Vokálová

Datum zadání závěrečné práce: 14.1.2015