Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Modelování v grafové nerelační databázi

Plzeň 2013 Milan Bečvář

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 16. května 2013

Milan Bečvář

Poděkování

Děkuji vedoucímu diplomové práce panu Ing. Romanovi Moučkovi, Ph.D. za vstřícný přístup, užitečné rady a připomínky při zpracování mé diplomové práce.

Abstract

This master thesis looks into non-relational databases focused on graph databases and choice of the most suitable of them. After that, an analysis of EEG/ERP data model is made and suitable subset is chosen. Then a data model is designed and implemented in the selected non-relational database. Finally, the selected non-relational database is tested together with the relational one and the attained results are evaluated. Neo4j is evaluated as the suitable non-relational database from the viewpoint of flexibility and speed of executing queries. That is why it was recommended for deployment at EEG/ERP portal.

Tato diplomová práce se zabývá nerelačními databázemi se zaměřením na grafové databáze a následným výběrem nejvhodnější z nich. Posléze je provedena analýza datového modelu EEG/ERP portálu a výběr vhodné podmnožiny. Následně je ve vybrané nerelační databázi navržen a implementován datový model. Nakonec je vybraná nerelační databáze otestována s databází relační a zhodnotí se dosažené výsledky, které vyhodnotily jako vhodnou nerelační databázi z hlediska flexibility a rychlosti vykonání dotazů právě Neo4j, a ta byla doporučena pro nasazení v rámci EEG/ERP portálu.

Obsah

1 Úvod

V praxi mohou nastat situace, kdy nám již přestane vyhovovat stávající datový model relační databáze. Důvodem může být pevně stanovené datové schéma, které je v případě relačních databází tvořeno tabulkami nebo velký objem uchovávaných dat. Z tohoto důvodu je vhodnější zvolit nerelační databáze. Nerelační neboli NoSQL databáze mohou ukládat data do různých struktur (graf, xml dokument, klíč / hodnota, apod.) a jsou schopné pracovat s obrovským množství dat (zejména Big data[1]). V současné době se využívají hlavně v oblasti cloud computingu, sociálních sítí, apod.

Náplní této diplomové práce je prostudovat možnosti, které nabízejí nerelační databáze a prorovnat je s databázemi relačními. Zaměřit se na databáze umožňující přímo ukládat grafové struktury (dále jen grafové databáze) a vybrat jednu z nich pro návrh a implementaci podmnožiny datového schématu EEG/ERP portálu. Takto vytvořenou grafovou databázi otestovat s databází relační a zhodnotit dosažené výsledky. Na základě zhodnocení dosažených výsledků rozhodnout, která databáze je vhodnější a proč.

Práce je členěna do několika kapitol. První kapitola se věnuje nerelačním databázím, např. možností dotazování, škálování a použití v praxi. Dále je uveden přehled základních typů nerelačních databází a následné zaměření na grafové databáze, ze kterých jsou vybrány tři zástupci a u každého z nich je uvedena základní charakteristika. Druhá kapitola se zabývá výběrem vhodné grafové databáze a detailní seznámení se ní. Další čtyři kapitoly jsou věnovány analýze datového schématu EEG/ERP portálu a výběrem vhodné podmnožiny. Součástí analýzy je generování testovacích dat do datového schématu EEG/ERP portálu. Na základě této analýzy se v další kapitole navrhne datový model grafové databáze a poté naimplementuje. Poslední se zabývá samotným otestováním relační databáze s databází nerelační. Testují se dva hlavní požadavky - flexibilita datového modelu a rychlost vykonání dotazů. Na konci této kapitoly jsou zhodnoceny dosažené výsledky, podle nichž se rozhodne, která z testovaných databází je vhodnější.

Dodatek k této diplomové práci tvoří kapitoly příloh, které obsahují uživatelskou dokumentaci popisující celý postup testování, dále schéma grafové databáze včetně rozšíření datového modelu a popis obsahu přiloženého CD.

[1]Big data - označuje soubory dat, jejichž velikost je mimo schopnosti zachycovat, spravovat a zpracovávat data běžně používanými softwarovými prostředky v rozumném čase.

2 NoSQL databáze

Pojem NoSQL označuje velmi početnou skupinu nerelačních databází. Tyto databáze na rozdíl od relačních databázé nepoužívají dotazovací jazyk SQL. Neznamená to tedy, že by všechna datová úložiště spadající do této skupiny odmítala SQL, jak by mohl název NoSQL naznačovat. Ve skutečnosti pojem NoSQL bývá databázovou komunitou vysvětlován jako „no only SQL" (v překladu „nejen SQL").

Především se jedná o alternativní přístup ukládání dat do databáze jiným způsobem než jakým ho ukládají databáze relační, který může být v některých případech vhodnější. Z tohoto důvodu nemusíme používat jen zaběhlé relační databázové systémy, ale v určitých případech, kdy se nehodí relační (tabulkový) model databáze, využít nerelační databáze. Přesnější definice je takováto: **"NoSQL databáze je software pro persistenci dat, který je alternativou ke klasickým relačním databázím".**[6]

2.1 Charakteristika a motivace

- Vysoká škálovatelnost (vertikálně, horizontálně)
- Snadná podpora replikací a distribuce dat
- Flexibilní schéma, semistrukturovaná data
- Částečné využití ACID[1] (transakčního zpracování)
- Podpora systému transakcí BASE[2] (opakem ACID)
- CAP teorém[3]
- Asynchronní vkládání, aktualizace dat
- Většina NoSQL databází jsou Open Source

[1]ACID (Atomicity, Consistency, Isolation, Durability)
[2]BASE (Basically Available, Soft-state, Eventually Consitent)
[3]CAP teorém (Consistency, Availability, Partition Tolerance)

Bezschémové nerelační databáze jsou stavěné tak, aby byly rychlé, dokázaly zvládnout obrovské objemy dat a dokázaly tolerovat výpadek sítě. Na úkor datové konzistence, tzn. data nejsou pro všechny uživatele v jednu chvíli stejná, ale „časem" se srovnají, tzn. uživatelé mají k dispozici stejná data.

Hlavní motivací NoSQL databází je dosažení horizontální škálovatelnosti[4] databázového zpracování v dynamickém prostředí distribuovaných databází, které obsahují semistrukturovaná data bez pevného databázového schématu. V nerelačních databázových systémech se obvykle neřeší transakční zpracování (ACID), referenční integrita, atd. Rychlost vykonávání dotazů u NoSQL databází je výrazně lepší než u robustnějších relačních databázích.

2.2 ACID

Jedná se o základní funkcionalitu databázového systému zajišťující transakční zpracování, kdy je potřeba, aby data byla konzistentní a při výskytu chyby bylo možné vrátit celou operaci do původního stavu. Mezi hlavní vlastnosti transakčního zpracování patří:

- **Atomicita (Atomicity)** - v transakci proběhnout buď všechny operace nebo žádná (každá transakce je atomická)
- **Konzistence (Consistency)** - systém musí zajistit, aby data byla konzistentní (výsledkem transakce jsou platná data)
- **Izolace (Isolation)** - transakce se nevzájem neovlivňují, např. při paralelním zpracování
- **Trvalost (Durability)** - změny zapsané úspěšnou transakcí jsou v databázi uloženy natrvalo

Databázový systém splňující všechny čtyři vlastnosti ACID transakcí se nazývá „silně konzistentní". Tyto databázové systémy jsou potřeba jen v určitých případech, např. při použití v bankách a obchodech. Systémy, které ne zcela podporují ACID vlastnosti se nazývají „případně konzistentní". V případě úmyslného zanedbání „silné konzistence" lze získat více dostupnosti a tím i výhodu lepší škálovatelnosti. Tento přístup bez „silné konzistence", často využívají právě NoSQL databáze. [6]

[4]Horizontální škálování - rozdělení výpočtu na paralelní úlohy mezi více serverů

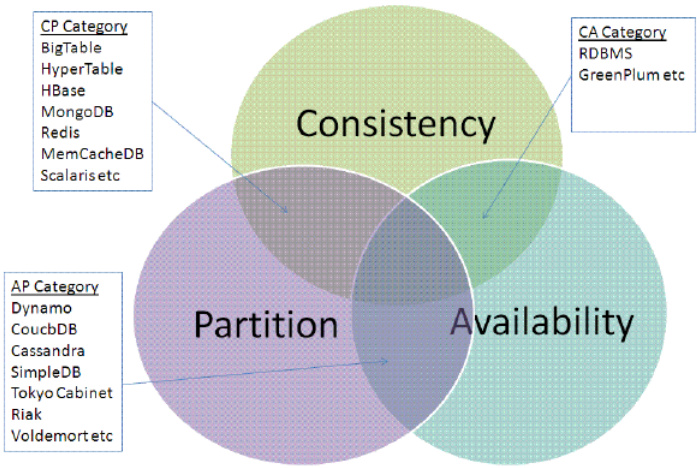
2.3 CAP Teorém

CAP teorém se používá při navrhování distribuovaného systému a obsahuje tři základní požadavky na konzistenci, dostupnost a odolnost proti výpadku sítě. CAP teorém říká, že neexistuje distribuovaný systém, který by splňoval všechny tři požadavky najednou.

Požadavky na distribuované prostředí:

- Konzistence (Consistency) - znamená, že v určitém čase všechny uzly distribuovaného systému vidí stejná data.
- Dostupnost (Availability) - znamená, že každý klient po svém dotazu dostane informaci o tom, zda-li byla operace úspěšná, nebo neúspěšná.
- Tolerance rozdělení sítě (Partition Tolerance) - znamená, že pokud část sítě vypadne (přeruší se spojení, anebo dojde ke ztrátě přenosu mezi uzly), systém bude stále schopný odpovídat na dotazy.

Ve skutečnosti je nemožné vytvořit návrh distribuovaného systému, který splňuje dvě z vlastností CAP teorému. Tuto skutečnost dokazuje Obrázek 2.1, kde je vyznačeno několik databázových systémů, především z řad předních zástupců NoSQL databází, které spadají do skupiny splňují dvě ze tří vlastností CAP teorému. [3]



Obrázek 2.1: Obecné znázornění CAP Brewerova teorému v NoSQL.[12]

2.4 Dotazování

V NoSQL databázích je dotazování nejméně propracovanou částí. Nad některými NoSQL databázemi se lze dotazovat jazykem SQL, avšak pouze jeho omezenou formou (SimpleDB, OrientDB). V této omezené formě není k dispozici např. operace spojení, agregace a zanořování dotazů, které lze nalézt u dotazovacích jazyků GOL (Google Query Language) nebo HQL (Hypertext Query Language). Oba tyto dotazovací jazyky jsou podmnožinou jazyka SQL. Operace spojení (JOIN) a řazení (ORDER BY) nejsou v NoSQL podporovány, a to z důvodu horizontálního škálování dat (viz škálování). Do budoucna je více než pravděpodobné, že bude vytvořen nějaký standardizovaný dotazovací jazyk použitelný pro celou skupinu NoSQL.[6]

2.5 Škálování

Relační databáze jsou ve většině případů umístěny na jednom serveru a provádí se tzv. vertikální škálování (scale-up), které funguje na principu přidání více procesorů nebo více vnitřní či vnější paměti. Druhým způsobem je škálovat data horizontálně (scale-out), což přináší výhodu v rozdělení výpočtu na paralelní úlohy mezi více serverů. Tento prostředek škálování se obzvláště používá ve spojitosti s NoSQL databázemi. Data v NoSQL databázích mohou být škálována i vertikálně a to tak, že se rozdělí záznam na části, přičemž každá z nich je uložena na jiném uzlu. Horizontální škálování zároveň komplikuje dosažení ACID vlastností.[6]

2.6 Použití v praxi

Rozhodování, kdy použít ověřené relační databáze nebo neznámou NoSQL databázi, není jednoduché. V tomto případě hraje roli jak technická tak i ekonomická stránka. Důvodem je také to, že NoSQL databáze jsou relativně mladou technologií, která se pořád vyvíjí. NoSQL databáze nemají takové možnosti při dotazování jako databáze relační, např. pokud aplikace podporuje jedinou DB operaci, a to uložení řádku a jeho načtení podle nějakého ID. To vede k tomu, že na aplikaci je přenášena odpovědnost v podobě generování sekundárních indexů, která je (většinou) součástí databázového enginu. V případě použití NoSQL by muselo dojít k předělání celé aplikace.

NoSQL databáze je doporučeno používat v situaci, která nám umožňuje snadné vytvoření datového modelu, nejlépe od začátku vývoje. Jelikož náklady spojené s přechodem na NoSQL databáze mohou být vysoké (školení programátorů, správa SW v infrastruktuře, záloha, apod.).

Databázové systémy patřící do NoSQL nacházejí uplatnění nejčastěji v cloud computingu, aplikacích web 2.0[5] a hlavně v sociálních sítích (Facebook, Twitter), kde horizontální škálování zahrnuje obrovské množství uzlů. Mezi klienty využívající NoSQL patří společnosti Adobe, Mozilla, Google a LinkedIn. U všech těchto zmíněných oblastí použití lze očekávat velký provoz, tudíž velký objem ukládaných dat, které je nutné zpracovat, a proto je výhodnější použít právě NoSQL databáze.

2.7 Srovnání s relačními databázemi

Relačními databázemi jsou nazývané takové databázové systémy, u kterých jsou data organizována a řízena SŘBD (Systém řízená báze dat). Datové schéma pro ukládání dat reprezentuje tabulka, která obsahuje řádky, sloupce a vztahy mezi jednotlivými tabulkami. Relační databáze vycházejí z datových modelů a schémat, do nichž ukládáme data z reálného světa. Jeho dotazovacím jazykem je SQL, který je standardizován. Největší výhodou relačních databází je snadná transformace objektů z reálného světa (tak, jak je vidí člověk) do tabulkového schématu.

I relační databáze mají své nevýhody. Nad určitou mezí objemu požadavků lze narazit na zásadní výkonnostní limit, který má za následek horší škálovatelnost a snížení rychlosti vykonaných dotazů nad databází. Složitěji se distribují (sharding) a pro velké weby (Facebook, Google, Twitter) již nestačí z důvodu uchovávání a zpracování velkého objemu dat, který roste exponenciálně. Největším úskalím relačních databází jsou samotné vztahy mezi tabulkami, a to v případě, kdy se dotazujeme databázového serveru na data z více tabulek, kdy je potřeba dané tabulky mezi sebou propojit. K propojení více jak jedné tabulky se používá příkaz JOIN, který je součástí dotazovacího jazyka SQL. Při nadměrném použití tohoto příkazu může docházet k výraznému snížení rychlosti při vykonání dotazu nad více tabulkami.

[5]Web 2.0 - webové stránky obsahující prostor pro sdílení a společnou tvorbu obsahu.

NoSQL databáze byly popsány na začátku této kapitoly. V této části se budu zabývat spíše výhodami oproti relačním databázím. V současné době se musíme potýkat s problémem, jakým způsobem uchovávat a zpracovávat velké množství dat, které pořád narůstá. Příkladem zpracování a uchování velkého objemu dat jsou např. osobní údaje o uživatelích (např. Facebook), sociální grafy, geolokace, uživatelsky generovaný obsah, atd. Problém je v tom, že zpracovávat takové množství dat v relačních databázích je velmi nákladné a často to má neblahý vliv na výkon celé databáze. Pro tyto účely byla vytvořena nová skupina nerelačních databázích zvaná NoSQL.

NoSQL databáze nemají oproti klasickým relačním databázovým systémům pevně stanovéná schémata pro ukládání dat. Pro každou situaci se hodí použít jinou nerelační databázi, která obsahuje jiný datový model, např. grafová, dokumentově orientovaná, sloupcová, atd. Nevýhodou je použití různých dotazovacích jazyků pro různé nerelační databáze oproti standardizovanému dotazovacímu jazyku SQL, který používá většina relačních databází.

2.8 Datový model

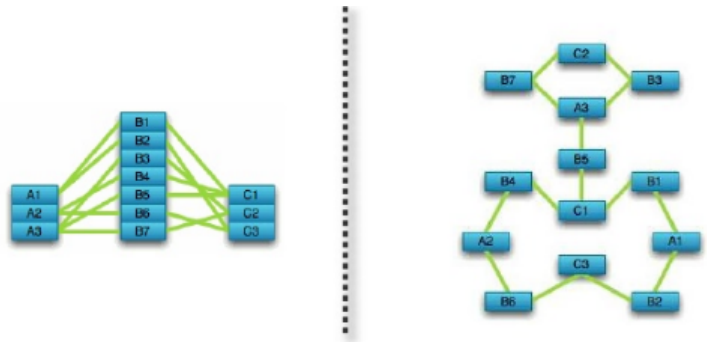
Ve světě databází se k popisu databáze využívá datový (logický) model. NoSQL uznává jiný přístup, který je intuitivní a bez předepsaných norem, podle kterého se různí i terminologie. V NoSQL existuje několik datových modelů, které se líší svojí strukturou.

- Grafové databáze (Graph database)
- Typ klíč/hodnota (Key-value database)
- Dokumentově orientované (Document Oriented database)
- Objektově orientované (Object database)
- Sloupcové orientované (Column Oriented database)
- XML (XML database)

2.9 Grafové databáze

Grafové databáze představují jednu z kategorií NoSQL systémů pro uchovávání dat reprezentovatelných prostřednictvím grafu, tj. elementů s nedeterminovaným počtem vzájemných propojení. Pracují na obdobném principu jako síťové databázové systémy. Pouze jejich vrcholy a hrany představují data strukturovaná jako množiny dvojic klíč/hodnota. Tímto odpadají problémy s uložením grafu / stromu do relační databáze, která není pro tyto účely příliš vhodná. V porovnání s relačními databázemi jsou grafové databáze při zpracování asociatvních dat často rychlejší, jelikož se obejdou bez výpočetně drahých operací JOIN. Navíc jsou schopné mnohem lépe mapovat strukturu objektově orientovaných aplikací bez pevně stanového datového schématu. Grafové databáze představují mocný nástroj pro řešení grafových úloh, jakými jsou například nalezení nejkratší cesty mezi dvěma vrcholy.

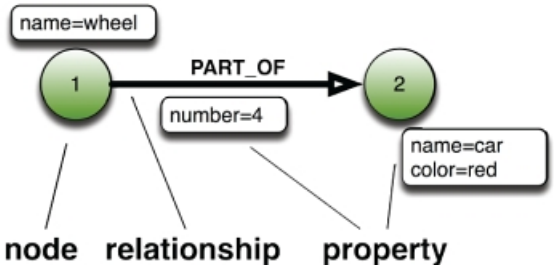
Na obrázku 2.2 je vidět rozdíl mezi datovým modelem relační databáze (vlevo), kde jednotlivé „obdelníčky" reprezentují daný řádek v tabulce a grafové databáze (vpravo), kde jednotlivé obdelníčky reprezentují vrcholy spojené hranou.



Obrázek 2.2: Srovnání datového modelu relační a grafové databáze. [9]

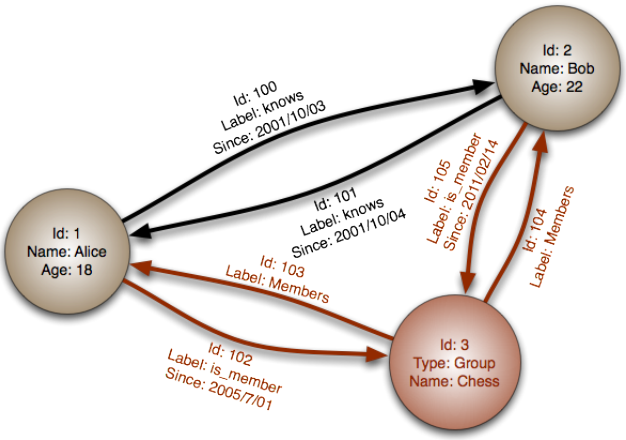
Vrchol v grafu reprezentuje jeden konkrétní záznam z tabulky relační databáze. Hrana reprezentuje daný vztah mezi dvěma tabulkami, např. 1:N, M:N, atd. Graf může např. reprezentovat trojice používané v datovém modelu RDF známém z webových aplikací a sociálních sítí. Grafová data mohou být jistě implementována pomocí relačního SŘDB, nicméně jejich zpracování, zejména rekurzivních dotazů je složité a pro zjišťování cest v grafu je třeba použít mnoho operací spojení. Výhodou grafových databázových systémů je, že velmi mnoho situací či modelů lze snadno popsat grafovou strukturou.

Na obrázku 2.3 je zobrazen datový model grafu popisující vlastnosti vrcholů a hran. Vrchol může být reprezentován nějakým objektem (člověk, auto) a mezi dvěma vrcholy je hrana (relace). Každý vrchol nebo hrana může obsahovat tzv. vlastnost (popis objektu). Relace mezi dvěma vrcholy mohou být různého charakteru. Základní typy relací jsou přímé a nepřímé. U přímé relace směřuje šika od rodiče k potomkovi a vrcholy jsou ve vztahu odpovídající relačním databázím, např. 1:N, M:N, 1:1. U nepřímé relace jsou šipky na obou stranách vrcholů a tím jsou vrcholy ve vzájemné relaci.



Obrázek 2.3: Terminologie pro popis vlastnosti grafu.[10]

Na obrázku 2.4 je vidět příklad jednoduchého modelu grafové databáze skládající se např. z několika vrcholů, hran a vlastností. Každý vrchol obsahuje vlastnosti (Id, Name, Age). Mezi dvěma vrcholy existuje alespoň jedna hrana, která obsahuje vlastnosti (Id, Label, Since). Vlastnosti lze přirovnat k atributům (sloupcům) u relačních databází. [10]



Obrázek 2.4: Ukázka jednoduchého modelu grafové databáze.[9]

2.9.1 Neo4j

Jedná se o grafový databázový systém se všemi vlastnostmi robustní databáze. Při použití knihovny Neo4j lze dosáhnout zvýšení rychlost dotazování až v řádů tisíců oproti konvečním relačním databázím (podle autora).[9]

- Splňuje ACID vlastnosti transakcí
- Podpora jazyků Java, .NET, Ruby, Python, ...
- Automatické indexování, fulltext, Apache Lucene
- Dotazování - Cypher (deklarativní), Gremlin (imperativní), Java API
- Licence - GPL (nekomerční použití), AGPL (komerční použití)

2.9.2 OrientDB

Open source řešení databázového systému s vlastností dokumentu a grafu, který je napsán v jazyce Java. Dokumentově orientovaná databáze je založena na grafové struktuře s přímým spojením mezi jednotlivými záznamy. [5]

- Splňuje ACID vlastnosti transakcí
- Dotazování pomocí SQL (bez JOIN)
- Jazyk Java, HTTP, RESTful s JSON formátem
- Hooks - alternativa k triggerům
- Open source (zdarma)

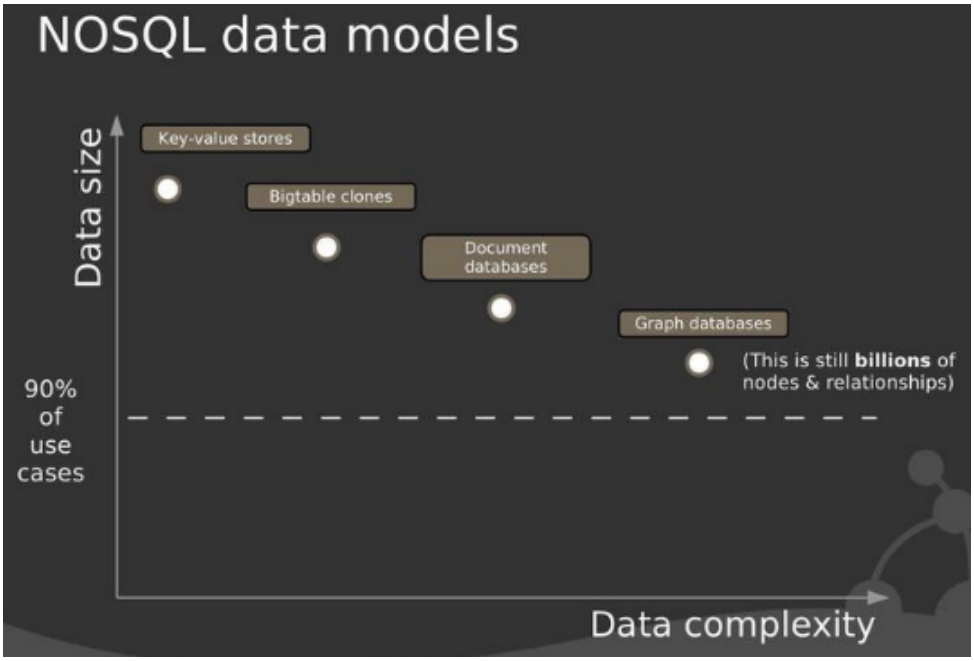
2.9.3 DEX

Vysoce výkonná a škálovatelná grafová databáze napsaná v jazyce Java a C++. Implementace knihovny obsahuje funkce pro analyzování a dotazování řádově bilionů objektů, které vyžadují malé množství úložného prostoru na disku a to díky kompresi. [14]

- Splňuje ACID vlastností transakcí
- Export do formátu - graphml, grahviz, ygraphml
- Podpora jazyků Java, .NET a C++
- Optimalizace vykonávání dotazů (poskytované kompilátorem)
- Kompatibilita s OS - Windows, Linux, Mac, iOS
- Licence - komerční a nekomerční (vývoj, testování, výzkum)

2.10 Obecné srovnání nerelačních databází

Poslední podkapitola v NoSQL databázích se zabývá obecným srovnáním hlavních zástupců této skupiny. Na obrázku 2.5 je graf závislosti složitosti dat na velikosti dat, se kterými zvládne daná databáze pracovat. Jak je patrné z grafu databáze typu „Key-Value stores" poskytují nejmenší složitost, ale mají nejvyšší kapacitu. Naproti tomu grafové databáze poskytují největší složitost, ale nejmenší kapacitu dat.



Obrázek 2.5: Obecné srovnání hlavních zástupců NoSQL databází. [8]

3 Neo4j

Neo4j spadá do skupiny NoSQL databází. Jedná se o grafový databázový systém vytvořený společností Neo Technology se všemi vlastnostmi robustní databáze. Práce s ní spočívá v programování objektově orientované síťové struktury oproti tradičním statickým tabulkám, které se běžně užívají v relačních databází. Podle autorů lze dosáhnout použitím knihovny Neo4j zvýšení rychlosti dotazování až v řádů tisíců, oproti konvečním relačním databázím. Potvrdit nebo vyvrátit toto tvrzení můžeme až po testování databáze Neo4j.

Neo4j se využívá v situacích, kdy potřebujeme modelovat četné vícenásobné vztahy, tzn. spojení mezi jednotlivými entitami a nevyhovuje nám klasický model relačních databází. [9]

3.1 Základní charakteristika

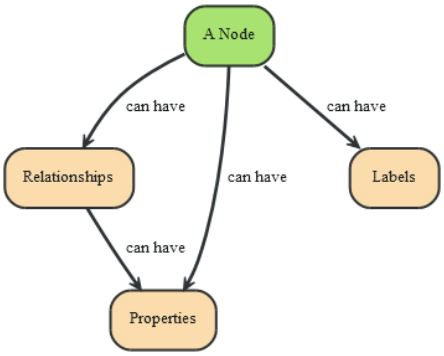
- Splňuje ACID vlastnosti transakčního zpracování
- Vysoký stupeň odolnosti a ochrany, který zaručuje, že žádná vložena data se neztratí
- Podpora jazyků Java, .NET, Ruby, Python, ...
- Autoři zmiňují, že klíčovým faktorem dobrého výkonu je mít hodně paměti RAM (odpovídající velikosti grafu)
- Databáze je limitována počtem 32 biliónů vrcholů, 32 biliónů hran a 64 biliónů vlastností
- Vynikající podpora technologie Spring
- Automatické indexování, Apache Lucene, fulltextový index
- Procházení grafem - Cypher, Gremlin, Java API a grafové algoritmy (Shortest paths, Dijkstra, A*),
- Připojení k databázi - embedded, REST API
- Licence - GPL (nekomerční), AGPL (komerční)

3.2 Datový model

Neo4j je databázový systém, který reprezentuje uložená data prostřednictvím grafové struktury, a proto je vhodný v situacích, kdy lze data jednoduše převést do grafové struktury. Data uvnitř grafu jsou uložena do generických struktur, např. List, Tree, Map. Datový model, který Neo4j používá se skládá z vrcholů (nodes), hran (relationships) a vlastností (properties). [9]

3.2.1 Vrcholy (nodes)

Vrcholy v datovém modelu reprezentují danou entitu v grafu a každý z nich může obsahovat jednu či více vlastností, která jej blíže specifikuje. Nejjednoduší graf je tvořen jedním vrcholem a od každého vrcholu musí vést minimálně jedna hrana. Vrcholy mohou být seskupené do pojmenovaných částí grafu a ty, které patří do stejné skupiny, jsou označené stejným štítkem (label). Při vytváření dotazu je jednoduší a efektivnější pracovat s pojmenovanými částmi grafu namísto celého grafu.[9]

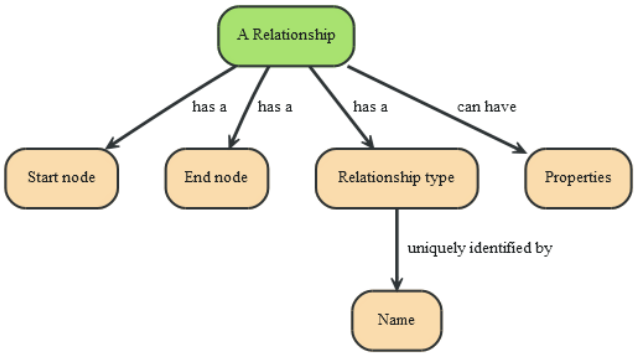


Obrázek 3.1: Ukázka reprezentace vrcholů v datovém modelu Neo4j.[5]

3.2.2 Hrany (relationships)

Hrana spojuje vždy dva vrcholy u nichž musí být platný začátek a konec. Hrany jsou popsány jednou či více vlastnostmi a mohou být orientované nebo rovnocené (neorientované) v obou směrech. Každá hrana musí mít typ

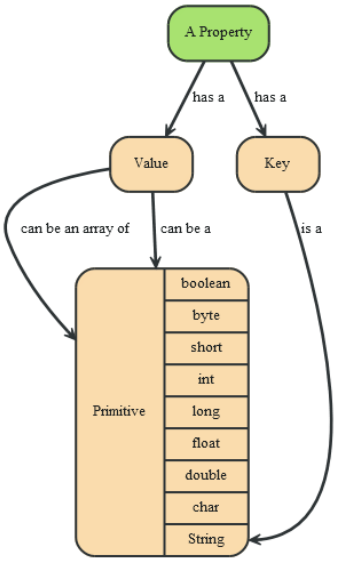
(relationship type) prostřednictvím nichž je možné najít určité vrcholy, do kterých směřuje hrana s daným typem. [9]



Obrázek 3.2: Ukázka reprezentace hran v datovém modelu Neo4j.[5]

3.2.3 Vlastnosti (properties)

Vlastnosti popisují jak vrcholy tak i hrany a jsou definovány jako dvojice klíč/hodnota, kde klíčem je vždy nějaký řetezec a hodnotou může být libovolný primitivní datový typ (viz obrázek 3.3). [9]

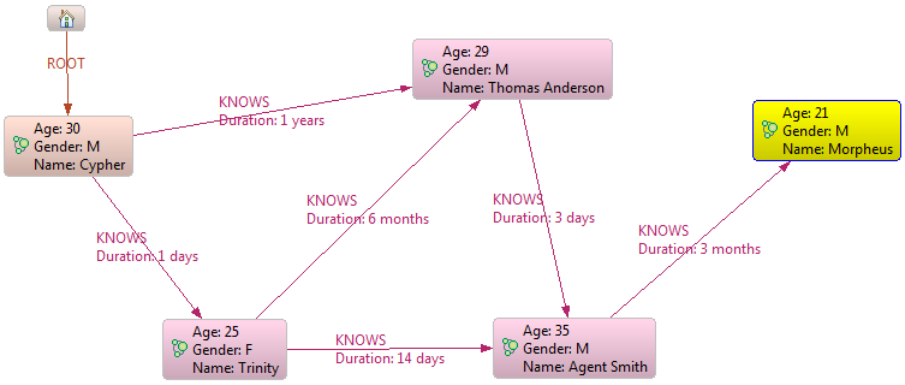


Obrázek 3.3: Ukázka reprezentace vlastností v datovém modelu Neo4j.[5]

3.3 Procházení grafem a dotazování

V předchozí části jsme popsali datový model grafové databáze Neo4j. Nyní je třeba popsat, jakým způsobem lze procházet vytvořený graf a jaké prostředky na to použít. Při procházení grafu platí, že data jsou uložena ve vrcholech, které blíže specifikují jejich vlastnosti. Vrcholy jsou spojeny pomocí hran, které mohou být blíže specifikovány svými vlastnostmi. Hrany se nejčastěji znázorňují ve směru od nadřazeného uzlu k podřízenému, tzn. od „rodiče" k „potomkovi" apod. Některé hrany není potřeba zdvojovat oběma směry. Například pokud máme typ hrany „zná se s", tak pokud se Martin zná s Janou, tak logicky plyne, že i Jana musí znát Martina. V tomto případě stačí hrana jedním směrem. Avšak u hrany typu „miluje" to vzájemné být nemusí a potom tuto vlastnost lze uvést v rámci relace.

Vlastnosti vrcholů a hran jsou důležitým faktorem při procházení grafu, jelikož obsahují informace o hledaném vrcholu nebo hrany. Například máme graf (viz obrázek 3.4) skládající se z vrcholů, které uchovávají informace o lidech (Name, Age, Gender) a hran popisující vlastnost „KNOWS". V tomto grafu chceme najít osoby, které se znají s osobou nazvanou „Thomas Anderson". Postup je následující, začneme procházet graf od vrcholu s vlastností „Thomas Anderson" a hledat vrcholy, které jsou s tímto vrcholem spojeny přímo hranou s vlastností „KNOWS" a nebo nepřímo přes nějaký další vrchol. Neo4j nabízí několik mechanismů na procházení grafem (Cypher, Gremlin, Java API).



Obrázek 3.4: Příklad na procházení grafem v Neo4j.[5]

3.3.1 Cypher

Jedná se o deklarativní grafový dotazovací jazyk, jehož způsob dotazování je podobný dotazovacímu jazyku SQL používanému v relačních databázích. Tímto prostředkem lze dosáhnout vysoce efektivního procházení grafem bez nutnosti použití složitých mechanismů, které nabízí Java API či Gremlin. Cypher je navržen tak, aby bylo možné jednoduše a efektivně klást dotazy bez nutnosti znalosti složitých příkazů.

Cypher je inspirován řadou různých přístupů, které navazují na zavedené postupy pro expresivní dotazování (např. jazyk SPARQL). Většina klíčových slov jako např. *WHERE* a *ORDER BY* jsou inspirovány jazykem SQL. Jak už bylo zmíněno, Cypher patří do skupiny deklarativních programovacích jazyků, které jsou založeny na myšlence programování pomocí definic, tzn. „co se má udělat" a nikoliv „jak se to má udělat". Příkladem použití dotazovacího jazyka Cypher je níže a používá graf z předchozího příkladu (viz Obrázek 3.4), kde je úkolem najít osoby, ke kterým vede identická hrana s vlastností „KNOWS".

```
// Příklad použití jazyka Cypher
START n = node(*)
MATCH n-[:KNOWS]->person
RETURN person.name, person.age
```

Příkaz `START` specifikuje jaké vrcholy se mají prohledat, ve většině případů použijeme znak „*", který reprezentuje všechny vrcholy. `MATCH` slouží pro identifikaci hran, které se mají zahrnout do procházení grafu. Poslední příkaz `RETURN` označuje množinu, která se má zobrazit na výstupu. V tomto případě se jedná o název a věk osoby. Person je pojmenovaná skupina vrcholů, do kterých vede hrana uvedená v příkazu `MATCH`.

3.3.2 Gremlin

Gremlin je doménově specifický jazyk pro procházení grafů, založený na jazyku Groovy. Jedná se imperativní programovací jazyk, u kterého se definuje postup jakým se má vykonat daný dotaz. Výhodou jazyka Gremlin je nezávislost na grafové databázi Neo4j a využívá Blueprints (analogie JDBC pro grafové databáze). Skripty dotazu poslané od klienta jsou spuštěny na serveru, kde je umístěna databáze a výsledky jsou vráceny jako množiny uzlů a

hran, které jsou kompatibilní s rozhraním Neo4j. Tento způsob udržuje konzistentní typy v celém REST[1] API. Skripty je možné posílat ze serveru i ve formátu JSON.

Gremlin umožňuje spustit samovolně kód z jazyka Groovy, což může představovat v hostovaném otevřeném prostředí bezpečnostní riziko. Příklad na použití jazyka Gremlin bude stejný jako v předchozím příkladě.

```
// Příklad použití jazyka Gremlin
g.v(0).out("KNOWS")
```

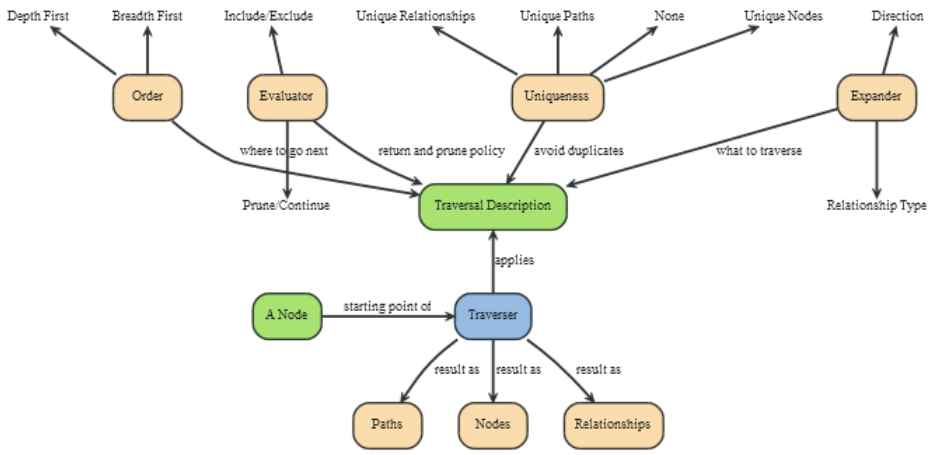
Tímto jednoduchým příkazem lze zobrazit množinu vrcholů, do kterých vede hrana s názvem *KNOWS* a pomocí příkazu `out` vypíšeme informace o nalezených vrcholech na výstup.

3.3.3 Traversal Framework

Další možností procházení grafu v Neo4j je využití vestavěného rozhraní API, které je nativně napsáno pro jazyk Java, ale existuje i podpora pro další jazyky prostřednictvím REST API, např. .NET, Ruby, Python. Traversal Framework se skládá z několika hlavních částí (viz Obrázek 3.5), kde je znázorněno blokového schéma rozhraní Traversal Framework.

- **TraversalDescription** - hlavní rozhraní pro inicializaci procházení grafem
- **Evaluator** - používá se při rozhodování na každém vrcholu v grafu (reprezentováno cestou), zda-li v cestě pokračovat nebo vrchol zahrnout do výsledku
- **Traverser** - objekt Traverser je výsledkem volání TraversalDescription reprezentující průchod v grafu a specifikuje formát zobrazení výsledku
- **Uniqueness** - množina pravidel pro úpravu již navštívených vrcholů během procházení
- **Expander**- definuje, co se má procházet, obvykle směr hrany vstupující do vrcholu a typ

[1]REST (Representational State Transfer) je architektonický styl navržený pro distribuované prostředí



Obrázek 3.5: Ukázka konceptu rozhraní Traversal Framework.[5]

- **Path** - základní rozhraní, které je součástí Neo4j API, umožňuje dva odlišné způsoby procházení:
  - Rozhraní traverser může vracet výsledky nalezené cesty v podobě navštívených vrcholů v grafu, které jsou označené za vrácené
  - Objekty cesty používají hodnocení pozic v grafu pro zjištění zda pokračovat v cestě od určitého bodu nebo ne, a zda je určitý vrchol zahrnut do výsledku či nikoliv

Dále uvedu jednoduchý příklad použití rozhraní Traversal Framework pro nalezení osoby z předchozího příkladu (viz Obrázek 3.4). Nejprve se inicializuje rozhraní *TraversalDescription* a poté se nastaví potřebné parametry, např. *depthFirst* (hloubka procházení), *relationships* (typ relace) a *uniqueness* (možina pravidel pro úpravu pozice při procházení). Nakonec se takto inicializované rozhraní použije pro získání jednotlivých vrcholů, které jsou popsány vlastnostmi. [9]

```
// inicializace
final TraversalDescription FRIENDS_TRAVERSAL = Traversal.description()
        .depthFirst()
        .relationships( Rels.KNOWS )
        .uniqueness( Uniqueness.RELATIONSHIP_GLOBAL );

//  Transformace objektu TraversalDescription na objekt Node (vrchol)
```

```
for ( Node currentNode : FRIENDS_TRAVERSAL
        .traverse( node )
        .nodes() )
{
    output += currentNode.getProperty( "name" ) + "\n";
}
```

3.3.4 Grafové algoritmy

Knihovna Neo4j obsahuje několik základních grafových algoritmů, které se používají v situaci, kdy potřebujeme najít nejkratší cestu z jednoho vrcholu do druhého. Nejčastěji používanými algoritmy jsou: [9]

- Shortest paths
- all paths
- all simple paths
- Dijkstra
- A*

Ukázka jednoduchého použití grafového algoritmu *Dijkstra* pro nalezení nejkratší cesty mezi vrcholem A a vrcholem B.

```
// příklad použití algoritmu Dijkstra
PathFinder<WeightedPath> finder = GraphAlgoFactory.dijkstra(
        Traversal.expanderForTypes( ExampleTypes.MY_TYPE,
        Direction.BOTH ), "cost" );
WeightedPath path = finder.findSinglePath( nodeA, nodeB );
path.weight();
```

Inicializuje se objekt `PathFinder`, který nastaví všechny důležité parametry (typ, směr hrany, váhu) pro výpočet nejkratší cesty pomocí *Dijkstrova* algoritmu. V objektu `WeightedPath` se jako parametry vloží odkaz na vrchol A a B. Nakonec se zavolá metoda *weight()*, která vrátí výslednou délku (váhu) nejkratší cesty.

3.4 Integrace a nasazení

Tato část kapitoly se zabývá začleněním knihovny Neo4j do projektu. Samotná integrace se skládá z instalace knihovny, výběrem verze knihovny (licence), minimálních a doporučených systémových požadavků a nakonec možnosti nasazení včetně reálného použití v praxi. Otestování vytvořené grafové databáze umožňují tzv. vizualizační nástroje (Gephi, Neoclipse a Web admin), které budou popsány na konci kapitoly.

3.4.1 Instalace

Neo4j knihovna může být nainstalována buď jako `embedded` databáze, tzn. databáze běží ve stejném procesu jako aplikace, která jí používá, nebo jako `standalone` databáze připojená k aplikaci přes rozhraní REST API, což dává velké možnosti ve výběru jazyků (.NET, Ruby, Python, ...) pro implementaci. Knihovnu pro `embedded` verzi databáze je možné stáhnout z oficiálních stránek Neo4j. Při použití vzdáleného připojení k databázi je nutné stáhnout ještě knihovnu pro REST API.

3.4.2 Licence

Neo4j je software, který je k dispozici ve dvou různých licencí. Pro nekomeční využití je k dispozici verze *Community* pod licencí GPL (General Public License), která je zdarma. V případě použití pro komerční účely jsou k dispozici verze *Advanced, Enterprise* pod licencí AGPLv3 (Affero General Public License), která je zpoplatněna.[9]

Verze	**Popis**	**Licence**
Community	základní databáze, plná podpora ACID	GPL
Advanced	Pokročilý monitoring (analýza aktivit databáze)	AGPLv3
Enterprise	online zálohování, klastrování, High availability	AGPLv3

Tabulka 3.1: Přehled dostupných licencí knihovny Neo4j.

3.4.3 Systémové požadavky

- **Procesor** - minimálně Intel Core i3, doporučeno Intel Core i7
- **Operační paměť RAM** - minimálně 2 GB, doporučeno 16 - 32 GB (podle velikosti grafu)
- **Pevný disk** - minimálně 10 GB SATA, doporučeno SSD w/ SATA
- **Operační systém** - Linux, Windows, Mac OS X (Neo4j založen na Javě => přenositelnost)

3.4.4 Nasazení

Po úspěšné instalaci a vytvoření modelu grafové databáze je nutné databázi nasadit. Jak už bylo zmíněno v části instalace, Neo4j využívá dva druhy přístupu k databázi.

- **Embedded** - jak bylo zmíněno v předchozí části kapitoly připojení k databázi probíhá lokálně, tzn. databáze je spuštěna ve stejném procesu jako aplikace, která jí používá. Při implementaci se pracuje s rozhraním `GraphDatabaseService`, které umožňuje přepínat mezi jednou a více instancemi databáze. Postup je jednouchý, pro jednu instanci se použije objekt `EmbeddedGraphDatabase` a pro více instancí se použije objekt `HighlyAvailableGraphDatabase`.
- **Standalone** - jedná se o vzdálené připojení k databázi, která je umístěna na jiném stroji než, na kterém je spuštěna databáze. Ke vzdálené databázi se lze připojit prostřednictvím rozhraní REST nebo dostupného ovladače konkrétního programovacího jazyka. [9]

3.5 Vizualizační nástroje

Vizualizační nástroje se používají pro zobrazení grafu nebo jeho části, která je výsledkem nějakého dotazu. Samotná vizualizace je účinným nástrojem pro vyjádření obsahu grafu, např. zvýraznění vzorů grafu, spojení mezi vrcholy, atd. Existují různé nástroje pro vizualizaci grafové databáze (nejen pro Neo4j), které se od sebe liší především nabízenou funkcionalitou.

3.5.1 Web admin Neo4j

Neo4j webové rozhraní je primárně uživatelské rozhraní pro správu databáze a její vizualizaci. Umožňuje monitorování Neo4j serveru, prohlížení a manipulování s daty a vykonávání dotazů prostřednictví konzole. Nástroj pro správu databáze je k dispozici na adrese *http://localhost:7474*, ale nejdříve je nutné spustit samotný server Neo4j. Administrační rozhraní se skládá z několika záložek:

- **Dashboard** - zobrazuje přehled spuštěných instancí Neo4j, počet všech vrcholů, hran, typů hran a vlastností.
- **Data browser** - slouží k procházení, přidávání nebo upravování vrcholů, hran a vlastností
- **Console** - vykonávání dotazů prostřednictvím jazyka Gremlin nebo Cypher, komunikování se vzdálenou databází pomocí HTTP protokolu
- **Server info** - zobrazuje detailní informace o nastavení serveru Neo4j a JVM

3.5.2 Neoclipse

Neoclipse je vizualizační nástroj implementovaný v jazyku Java a klade si za cíl podporovat vývoj Neo4j aplikací. Nástroj je k dispozici zdarma (Open source). Mezi jeho hlavní funkce patří

- Vizualizace grafové databáze
- Filtrování podle typů hran

- Vytvoření/smazání vrcholů a hran
- Vytvoření typů hran
- Vytvoření/smazání/editování vlastností popisující vrcholy či hrany
- Zvýraznění části grafu a možnost přidání ikon k vrcholům
- Podpora dotazovacího jazyka Cypher

3.5.3 Gephi

Gephi je interaktivní vizualizační desktopový nástroj, který umožňuje procházet a manipulovat s grafy. Nástroj je dostupný zdarma (Open source). Uživatel interaguje s grafem, manipuluje se strukturami - tvarem a barvami, aby se ukázaly skryté vlastnosti grafu. Užívá se hlavně k zobrazování objemných grafů v reálném čase a k urychlení prohlížení využívá 3D engine. Mezi jeho hlavní funkce patří: [2]

- Vizualizace v reálném čase
- Prohlížení a analyzování dat
- Analyzování propojení mezi jednotlivými vrcholy
- Analýza sociálních sítí
- Vytvoření kartografií
- Metrika grafu
- Klastrování hiearchických grafů

4 EEG/ERP portál

4.1 Vize portálu

„Prvotní myšlenka EEG/ERP portálu vznikla v roce 2008 na základě několika důležitých podnětů. Katedra informatiky a výpočetní techniky na Západočeské univerzitě v Plzni se zabývá projektem experimentálního měření mozkové aktivity. V době před vznikem portálu se naměřená data ukládala neuspořádaně na počítač v laboratoři na katedře. Samotná naměřená data nám ale nic neřeknou o měřené osobě, měřící osobě, použitých přístrojích a scénářích experimentů. Společně s naměřenými daty bylo nutné ukládat i informace o jednotlivých experimentech. Dále bylo zapotřebí data a informace z experimentů ukládat do databáze a umožnit přístup více uživatelům přes webové rozhraní. Dalším důvodem, proč portál vznikl, je jeho jedinečnost. Po prozkoumání dostupných alternativ bylo jednoznačně nutné portál vytvořit od základu a to tak, aby splňoval potřebná kritéria.

Samotná katedra na ZČU není jediná, která se výzkumem EEG zabývá. Existuje mnoho dalších zájmových skupin s podobným zaměřením, které by také potřebovaly své experimenty sdílet s ostatními. Jedna z vizí tohoto projektu je umožnit ostatním uživatelům přidávat a konzultovat své experimenty přes webové rozhraní."[4]

4.2 Popis portálu

Portál je webová aplikace sloužící ke správě neuroinformatických dat získaných měřením mozkové aktivity. Umožňuje skupinám výzkumníků ukládat, aktualizovat, stahovat data a metadata z EEG/ERP experimentů naměřených v laboratořích. Portál je vyvíjen jako samotný produkt s licencí GNU GPL. Přístup do databáze je pomocí webového rozhraní. Portál je napsán v jazyce Java a je založen na frameworku Spring MVC, Spring Security a technologii JSP. Datová vrstva pracuje s neuroinformatickou databází a používá k tomu systém Oracle 11g spolu s objektově relačním mapováním, které zajišťuje framework Hibernate.

Portál nabízí následující sadu funkcí:

- Registrace uživatelů
- Ukládání, aktualizace a stahování dat, scénářů a experimentů
- Nástroje pro zpracování signálů
- Historie stahování
- Systém pro správu obsahu (CMS)
- Fulltextové vyhledávání

Portál je testován na serveru Jetty. Na testovacím serveru je dostupný vždy denní *build* portálu. Produkční verzi portálu lze nalézt na adrese eegdatabase.kiv.zcu.cz, která běží také na serveru Jetty. Celý projekt EEG/ERP je dostupný na veřejném hostovacím serveru Github [1].

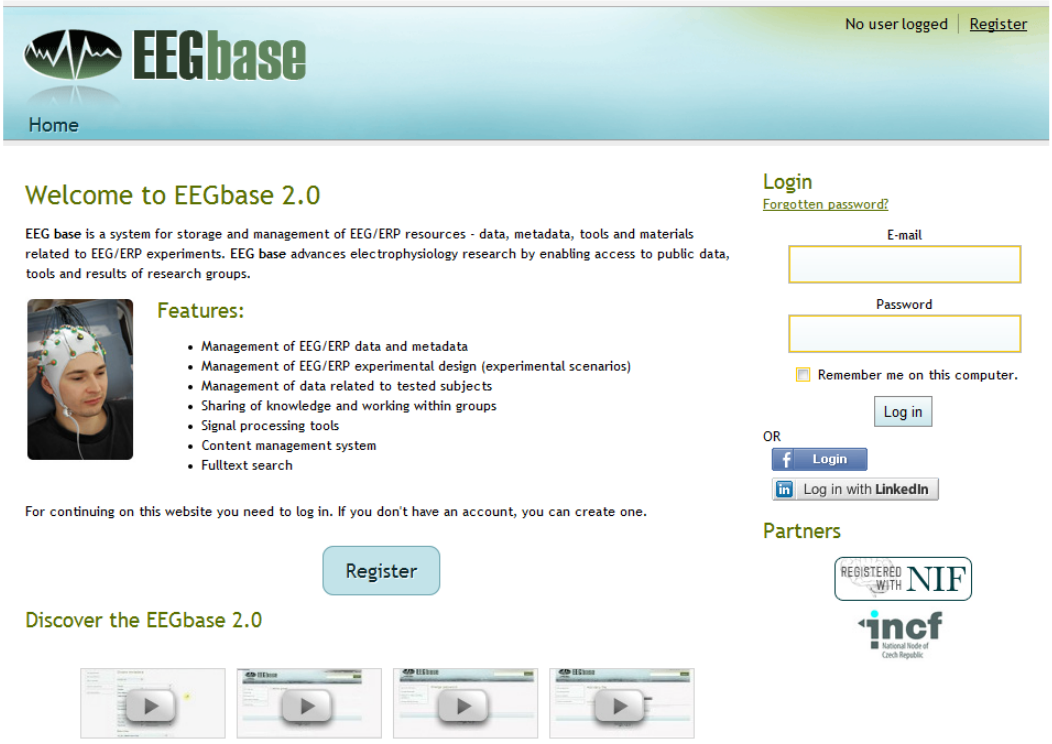
Databáze nese kromě informací o samotných experimentech a dat jim náležících i další struktury pro běh samotného portálu. Mezi ty hlavní patří např. informace o uživatelích, výzkumných skupinách, článcích, jejich komentářích, experimentech, atd. V současné době se posouvá význam portálu z běžného úložiště dat k velké webové aplikaci schopné analyzovat data a pracovat s nimi přímo na serveru.[13]

4.2.1 Role portálu

Uživatelé pracující s EEG/ERP portálem jsou většinou pracovníci katedry informatiky a výpočetní techniky, dále studenti, kteří se podílejí na měřeních, a nakonec i širší komunita zabývající se tématikou EEG/ERP. Aktuálně v portálu rozlišujeme celkem pět různých rolí.

- **Neregistrovaný uživatel** - uživatel může navštívit pouze domovskou stránku a může se registrovat
- **Čtenář** - registrovanný uživatel, který si může stáhnout veřejné experimety a nemá právo prohlížet osobní informace o měřících a měřených osobách. Dále může přidávat názory na veřejnou nástěnku. V případě, že je členem skupiny, pak může přidávat názory i na nástěnku skupiny.

- **Experimentátor** - registrovaný uživatel, který může přidávat experimenty, v rámci zákona má povoleno používat osobní data subjektů, na kterých provedl experiment. Dále může přispívat svými názory na veřejnou nástěnku a na nástěnku skupiny, jejíž je členem.
- **Administrátor skupiny** - má práva spravovat skupiny, přispívat svými názory a aktualitami na nástěnku skupiny, kterou on sám vytvořil. Dále může přispívat svými názory na veřejnou nástěnku a prohlížet evidenci stahování experimentů členů své skupiny.
- **Administrátor** - uživatel, který má nejširší práva ke správě portálu. Může spravovat evidenci stahování experimentů všech uživatelů portálu. [4]



Obrázek 4.1: Úvodní stránka EEG/ERP portálu.[11]

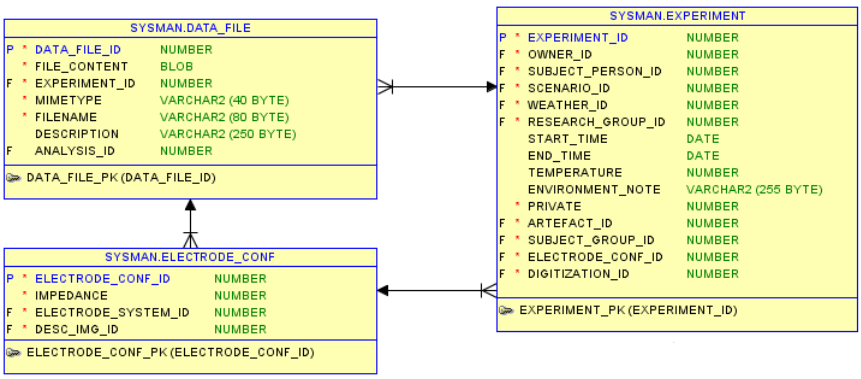
4.3 Analýza datového schématu

Cílem této analýzy je prozkoumat stávající datové schéma EEG/ERP portálu a vybrat vhodnou podmnožinu použitelnou pro modelování datové vrstvy v grafové databázi Neo4j.

Datové schéma EEG/ERP portálu je založené na databázové technologii Oracle 11g. Databáze je z větší části navržena tak, aby uchovávala informace o provedených experimentech. Mimo toho lze ukládat do databáze také informace o lidech, kteří pracují na nějakém experimentu, výzkumných skupinách, scénářích, článcích, atd. Kompletní datové schéma EEG/ERP portálu je k dispozici v příloze.

Během analýzy bylo zjištěno chybné propojení mezi některými tabulkami v datovém schématu, při kterém došlo ke vzniku cyklů. Takto vzniklé cykly mají neblahý účinek na celé schéma databáze, a proto je nutné je odstranit. Na obrázku 4.2 jsou zobrazeny tabulky, u kterých došlo k zacyklení. Ke vzniku cyklu došlo následujícím způsobem. Tabulka `EXPERIMENT` obsahuje cizí klíč, který odkazuje na tabulku `ELECTRODE_CONF`. Ta obsahuje cizí klíč, který odkazuje na tabulku `DATA_FILE`. V této chvíli ještě k žádnému zacyklení nedošlo, ale bohužel tato obsahuje dále cizí klíč, který odkazuje zpět na tabulku `EXPERIMENT` a tím došlo k zacyklení.

Tento problém vzniku cyklu lze řešit více způsoby, ale my se spokojíme s jednoduchým řešením, které spočívá v odstranění cizího klíče z tabulky `DATA_FILE`. Tímto je problém zacyklení vyřešen a můžeme pokračovat ve výběru vhodné podmnožiny datového schématu.

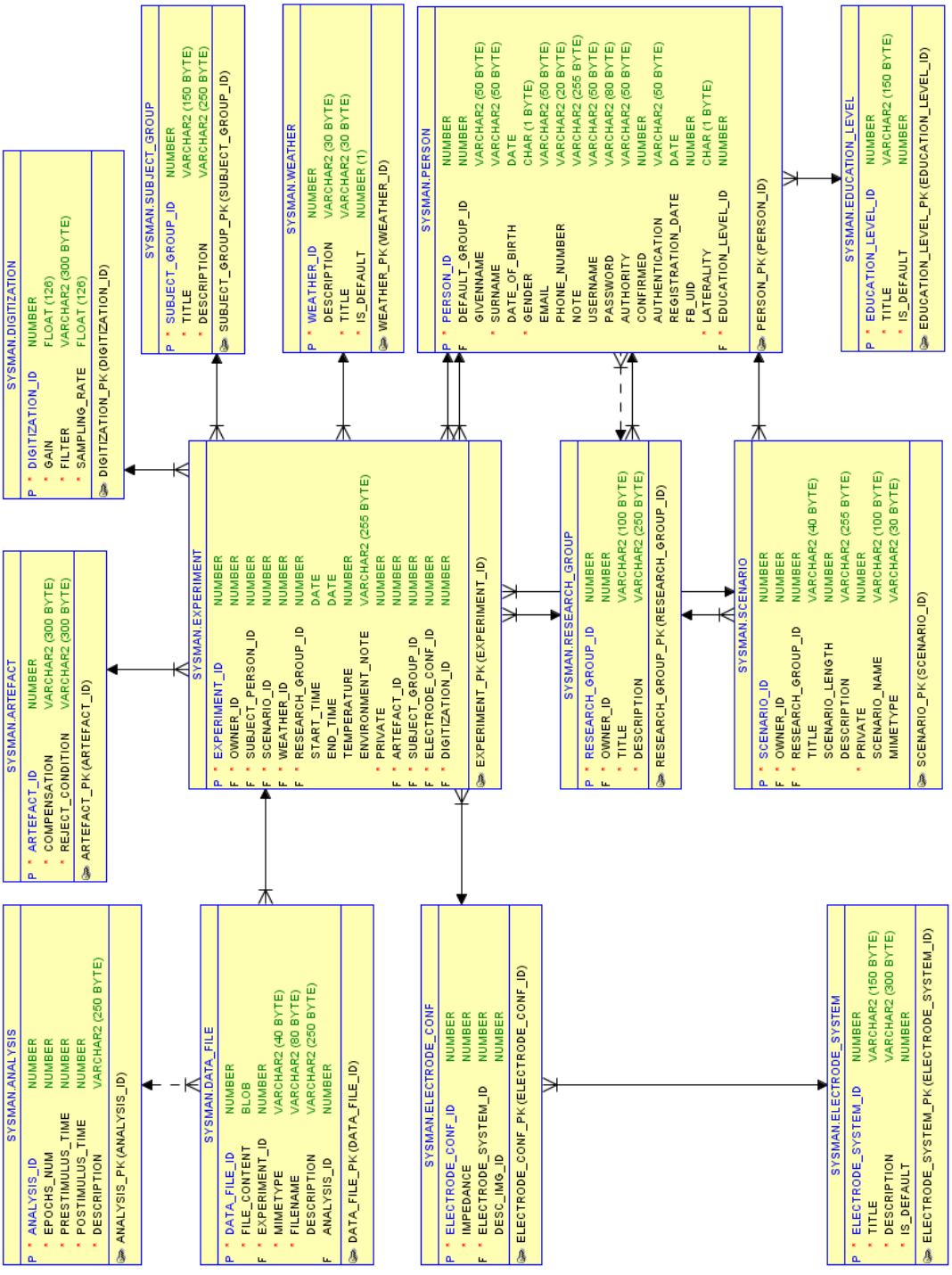


Obrázek 4.2: Ukázka zacyklení v datovém schématu EEG/ERP portálu.

4.4 Výběr vhodné podmnožiny

V této části analýzy datového modelu je nutné vybrat vhodnou podmnožinu datového schématu EEG/ERP portálu, která bude použita při modelování grafové databáze. Požadavky na výběr podmnožiny nebyly stanovené, ale je nutné, aby ve vybrané podmnožině byla taková část datového schématu, která je nejvíce používaná. Z důvodu toho, že větší část databáze tvoří informace o experimentech, je zcela žádoucí, aby byla za vhodnou podmnožinu datového schématu vybraná právě tabulka uchovávající informace o experimentech včetně všech závisejících tabulek.

Na obrázku 4.3 je znázorněno datové schéma vybrané podmnožiny, která obsahuje tabulku experimentů a také všechny závislé tabulky. Podmnožina datového schématu již neobsahuje žádné cykly, které by při modelování grafové databáze nebyly problémem, ale bylo nutné tento cyklus odstranit v datovém schématu relační databáze.



Obrázek 4.3: Ukázka podmnožina datového schématu EEG/ERP portálu.

4.4.1 Popis vybraných tabulek

EXPERIMENT
Tabulka experimentů uchovává informace o všech uskutečněných experimentech. V tabulce se zaznamenává hlavně čas začátku a čas konce experimentu, teplota, popis prostředí, počasí, konfigurace elektrod, digitalizace, výzkumná skupina, druh scénáře a další.

PERSON
Tabulka osob uchovává informace o všech registrovaných členech a testovaných subjektů, kteří se podílejí na výzkumné činnosti. Základní informace o těhto osobách jsou např. uživatelské jméno, výzkumná skupina a nejvyšší dosažené vzdělání, osobní údaje (jméno příjmení, email, pohlaví, ...), datum registrace, atd.

EDUCATION_LEVEL
V tabulce se uchovávají informace o nejvyšším dosaženém vzdělání dané osoby, např. název dosaženého vzdělání.

SUBJECT_GROUP
Tabulka obsahuje seznam skupin subjektů, která slouží pro rozlišení testovaných subjektů. Obsahuje informace o názvu a popisu skupiny.).

ELECTRODE_CONF
Tabulka uchovává informace o konfiguraci elektrod pro daný experiment, např. hodnotu maximální impedance, systém, dle kterého byly elektrody osazeny.

ELETRODE_SYSTEM
Slouží k uchovávání informací o jednotlivých systémech, na kterých byly elektrody osazeny. Obsahuje informace o názvu a popisu systému.

DIGITIZATION
Tabulka uchovává informace o procesu převodu analogového signálu do digi-

tálního k danému experimentu. Obsahuje informace o použitém filtru, vzorkovací frekvenci a hodnotě zisku (gain).

ANALYSIS
V této tabulce se ukládají informace o provedné analýze signálu k danému systému, např. délka signálu před a po stimulu, popis analýzy a počet epoch.

ARTEFACT
Tabulka artefaktů uchovává informace o metodách kompenzace artefaktů a podmínky pro zahození měření k danému experimentu.

DATA_FILE
Tabulka obsahuje informace o datových souborech, které jsou přidruženy k nějakému experimentu, např. název souboru, typ souboru, popis, název provedené analýzy a samotný binární datový soubor ve formátu *BLOB*.

SCENARIO
Tabulka scénářů uchovává informace k jednotlivým experimentům, jako např. název scénáře, popis, typ, osobu odpovědnou za vytvořený scénář a název výzkumné skupiny.

WEATHER
Do této tabulky se ukládají informace o počasí, např. popis aktuálního počasí a název.

RESEARCH_GROUP
V této tabulce se uchovávají informace o všech výzkumných skupinách, přičemž každá z těchto výzkumných skupin obsahuje informace jako např. název výzkumné skupiny, popis a správce skupiny.

5 Generování testovacích dat

Tato kapitola se zabývá generováním testovacích dat nad relační databází Oracle pro EEG/ERP portálu. Vygenerování testovacích dat je nutné provést již před samotnou implementací grafové databáze Neo4j, jelikož při vytváření grafové struktury budou tato data vyžadována.

5.1 Datanamic Data Generator for Oracle

Tento nástroj je určený právě pro generování testovacích dat nad Oracle databází. Produkt je možné stáhnout jako trial verzi na 30 dní zdarma, ale po uplynutí doby je nutné zakoupit licenci. Pro účely testování mi byla poskytnuta licence na tento produkt, která byla vydaná pro „University of West Bohemia". Nástroj poskytuje mnoho funkcí, např.

- Generování smysluplných testovacích dat
- Generování dat přímo do databáze nebo do souboru
- Generování dat založené na charakteristice sloupců
- Vytvoření vlastních generátorů dat
- Sada předpřipravených generátorů
- Podpora pro Oracle 9i, 10g, 11g
- Podpora pro databáze MySQL, Oracle, MS SQL Server, MS Access a PostgreSQL

5.1.1 Postup vygenerování dat

Než začneme s generováním dat, je potřeba nejprve vytvořit databázi na lokálním úložišti, z důvodu snadnějšího přístupu a práce s databází. K tomu je potřeba nainstalovat databázový server Oracle (aktuální verze 11g). Celý proces instalace je popsán v uživatelské příručce, která se nachází v Příloze

A. Dále je nutné nainstalovat samotný nástroj pro generování dat *Datanamic Data Generator for Oracle* a poté spustit. Po spuštění založíme projekt, ve kterém nastavíme parametry pro připojení k lokálně vytvořené databázi a poté vybereme tabulky, které budou zahrnuty do procesu generování. Po načtení tabulek vybereme u jednotlivých sloupců tabulek typ generátoru dat. Nástroj je natolik inteligentní, že podle názvu sloupce rozpozná, o jaký druh dat by mohl jít, a automaticky přiřadí možnou variantu z předem připravené sady generátorů. Například pokud se jedná o sloupec s názvem „FileName" automaticky vybere generátor pro název souboru. U ostatních sloupců, které nerozpozná, se vybere generátor podle datového typu.

V průběhu nastavení generátorů jsem narazil na několik problémů. Například tabulka `ELECTRODE_CONF` obsahuje sloupec s názvem `IMPEDANCE`, který má datový typ *Number*, což reprezentuje obecnou číselnou hodnotu. Tato hodnota může nabývat buď celočíselné hodnoty nebo desetinné hodnoty. V tomto případě byla vyžadována celočíselná hodnota, pro kterou nebyl k dispozici vhodný generátor. Nástroj sice poskytuje funkci pro vytvoření vlastního generátoru hodnot, ale nepodařilo se mi takto vytvořený generátor umístit do nabídky generátorů. Většina předdefinovaných generátorů mají možnost zvolit si rozsah, v jakém se mají generovat hodnoty pro daný sloupec a v jakém intervalu se mají tyto hodnoty opakovat.

Ve vlastnostech projektu lze nastavit, např. počet vygenerovaných řádků pro všechny tabulky nebo formát generátoru datumů. Počet vygenerováných řádek je možné nastavit individuálně ke každé tabulce. V této chvíli již máme nastavené generování a zbývá jen nastavit pořadí tabulek, ve kterém se mají generovat data, aby se správně uložily hodnoty cizích klíčů. Po tomto kroku spustíme proces generování, ve kterém ještě definujeme, kam se mají vygenerovaná data uložit. První možností je uložit data přímo do databáze. Druhá možnost je generovat data přímo do souboru s příponou *.sql*. My zvolíme možnost první - generování dat do databáze protože kdybychom zvolili generování do souboru, tak by se neuložily hodnoty cizích klíčů.

Po dokončení generování testovacích dat můžeme ověřit jestli se všechna data vygenerovala. Otevřeme si databázi např. v programu *SQL Oracle Developer* a zkontrolujeme obsah naplněných tabulek. Každá z tabulek obsahuje jeden milion řádek, aby bylo možné důkladně otestovat výkonost celé databáze při velkém množství uložených dat.

Celý proces generování trvá v průměru mezi 20 - 30 hodinami. Ovšem záleží na fyzickém stroji, na kterém tento proces generování probíhá.

6 Návrh datového modelu v Neo4j

Předchozí kapitola se zabývala analýzou datového schématu EEG/ERP portálu, z něhož byla vybrána vhodná podmnožina. Cílem této kapitoly je navrhnout pro vybranou podmnožinu datový model grafové databáze Neo4j a popsat, jak vypadá výsledná datová struktura.

6.1 Mapování z datového schématu relační databáze

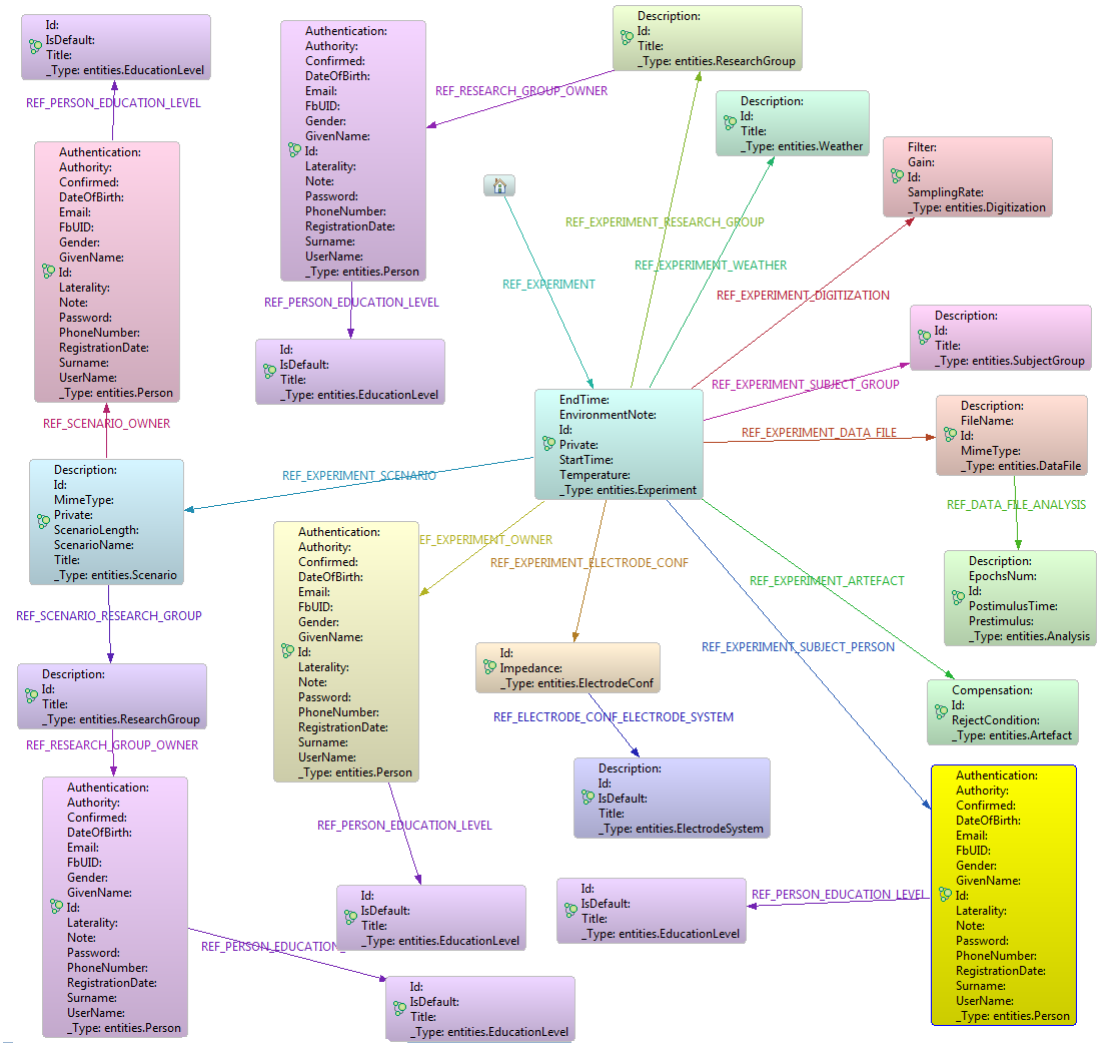
Datové schéma v relačních databázích je reprezentováno tabulkami, tzn. data jsou uložena do jednotlivých sloupců a řádků. Abychom mohli tato data převést do grafové struktury je nutné nejprve definovat pravidla za jakých je to možné.

Každá záznam v tabulce datového schématu relační databáze je v grafové struktuře vyjádřen jako vrchol. Jednotlivé sloupce v tabulce jsou v grafové struktuře uloženy jako vlastnosti, které blíže popisují daný vrchol. Vlastnosti v daném vrcholu jsou definovány jako dvojice klíč/hodnota. Klíčem je vždy název vlastnosti, který odpovídá sloupci v datovém schématu relační databáze a hodnotou, která odpovídá řádku v datovém schématu relační databáze může být libovolný primitivní datový typ (String, integer, boolean, ...). Mezi jednotlivými tabulkami relační databáze můžou existovat různé vztahy, které jsou v grafové struktuře vyjádřeny hranou. V relační databázi obvykle rozlišujeme dva typy vazby:

- **1:N** - tato vazba v terminologii relačních databázích znamená, že přiřazujeme jednomu záznamu v jedné tabulce více záznamů z jiné tabulky. V grafové struktuře tato vazba odpovídá orientované hraně, směřující od jednoho vrcholu ke druhému.
- **M:N** - v relačních databázích se u této vazby vytváří tzv. vazební tabulka, která tvoří vazbu mezi dvěma tabulkami. Tato vazba umožňuje několika záznamům z jedné tabulky přiřadit několik záznamů z tabulky druhé. V grafové struktuře tato vazba odpovídá neorientované hraně, kterou popisují vlastnosti obsahující data z vazební tabulky.

6.2 Schéma datového modelu

Na obrázku 6.1 je zobrané schéma datové struktury popsané grafem, který se skládá z vrcholů a hran. Graf obsahuje všechny typy vazeb, které jsou obsažené v datovém schématu relační databáze pro jeden konkrétní záznam. Graf je tvořen jedním počátečním vrcholem označeným jako kořenový, ze kterého jsou postupně vytvářeny další vrcholy.



Obrázek 6.1: Ukázka návrhu grafové struktury v Neo4j.

6.3 Popis datového modelu

Datový model je tvořen několika vrcholy, které jsou mezi sebou nějakým způsobem propojené. Společné vlastnosti všech vrcholů jsou např. identifikátor vrcholu (číselná hodnota) a typ, který je tvořen názvem třídy popisující danou entitu, např. `entities.Experiment`. Základním vrcholem této datové struktury je kořenový vrchol, o kterého se postupně vytvářejí ostatní vrcholy. Druhým vytvořeným vrcholem je vrchol typu `Experiment`, který obsahující informace o experimentech. Od tohoto vrcholu se vytvářejí ostatní vrcholy typu např. `Digitalization, Artefact, Scenario, atd.` Všechny vrcholy jsou propojené orientovanou hranou a neobsahují žádný cyklus. Každá hrana obsahuje vlastnost, např. `REF_EXPERIMENT` identifikující vrchol, do kterého hrana směřuje. Kompletní statistika o stavu vygenerovaného grafu je v tabulce 6.1.

Celkový počet vrcholů	20 034 839
Celkový počet hran	20 020 000
Celkový počet vlastností	94 630 475
Velikost databáze	14 GB

Tabulka 6.1: Statistika vytvořené grafové databáze Neo4j.

7 Implementace datového modelu v Neo4j

Tato kapitola popisuje postup implementace datového modelu v grafové databázi Neo4j za použití Java API, které bylo popsáno v kapitole o Neo4j. Vytvoření grafové databáze Neo4j je důležitou části v celém experimentu, aby bylo možné porovnat datový model relační databáze s datovým modelem grafové databáze.

7.1 Použité technologie

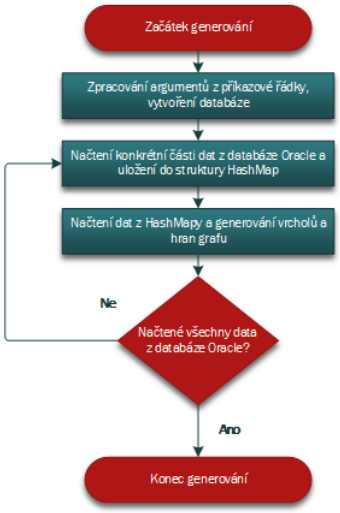
Při implementaci grafové databáze v Neo4j byly použity tyto technologie:

- **Neo4j Community 2.0.0 M02** - knihovna pro vytvoření grafové databáze
- **Java (JDK 1.7)** - jazyk, ve kterém se provede implementace grafové databáze Neo4j
- **Oracle 11g** - databázový server, na kterém běží relační databáze stávajícího EEG/ERP portálu

7.2 Popis implementace

Aplikace je členěna do několika logických bloků tzv. *packages*, přičemž samotná třída, která spouští aplikaci se nachází v kořenové části aplikace. Ve struktuře aplikace se nacházejí tyto bloky *entities, relationships, utils*. Podrobnější popis všech částí aplikace bude uveden dále.

Na obrázku 7.1 je vývojový diagram, který obsahuje jednotlivé kroky, kterými lze docílit k vytvoření kompletní grafové databáze. V rozhodovací části se algoritmus rozhoduje, zda-li ukončit vytváření grafové databáze, v případě, že jsou všechny data z relační databáze Oracle načtena, a nebo pokračovat v načítání dat.



Obrázek 7.1: Vývojový diagram vytvoření grafové databáze Neo4j.

Client
Hlavní třída, která obsahuje několik důležitých metod. První metodou je `main()`, která vytváří hlavní vlákno aplikace. V těle této metody se nejprve zpracují zadané argumenty, kde první argument je hodnota, od které se budou načítat data z Oracle databáze a druhý argument představuje cestu k vytvořené grafové databázi. Poté se vytvoří grafová databáze na disku podle zadané cesty a nakonec se zavolá metoda `generateGraphDatabase()`, která naplní vytvořenou grafovou databázi daty z databáze Oracle.

```
void generateGraphDatabase(BatchInserter batchDb,int startNumber,
DdConfig dbConfig)
```

Metoda nejprve načte data v daném rozsahu z Oracle databáze pomocí metody `loadDataFromDb()`, vytvoří strukturu grafové databáze a naplní jí načtenými daty z Oracle databáze a uzavře databázové připojení.

```
void clearDb(String pathDb)
```

Vymaže celou databázi v případě, že existuje. Tato metoda se volá jen v případě, pokud je hodnota prvního argumentu rovna nule. Proto aby bylo možné vytvořit prázdnout databázi v první iteraci generování grafové databáze.

DbConfig
Třída popisuje vlastnosti pro ukládání informací o připojení k databázi Oracle, např. přihlašovací jméno, heslo, název služby. Tyto informace jsou zadávány při spuštění této aplikace.

ImportDataFromDb
Tato třída poskytuje prostředek pro získání dat z Oracle databáze prostřednictvím *JDBC* ovladače. Obsahuje jedinou metodu `loadDataFromDb()`, která načte data z Oracle databáze.

```
void loadDataFromDb()
```

V těle metody dojde nejprve k připojení prostřednictvím *JDBC* ovladače k Oracle databázi a poté se vykoná dotaz, který má za úkol načíst určiou množinu dat podle zadané počáteční hodnoty, od které se začnou data načítat, a počtu dat. Takto načtená data se uloží do struktury *HashMap*, která se zpřístupní metodě `generateGraphDatabase()`.

Balík relationships
V tomto balíku se nachází výčtový typ `RelationshipTypeEnum`, který obsahuje názvy vlastností hran mezi vrcholy v grafové struktuře.

Balík entities
Tento balík obsahuje jednotlivé třídy reprezentující danou entitu. Všechny tyto třídy jsou odděděné od třídy `Entity`, která poskytuje všem svým potomkům společné vlastnosti (Id, Type). V každé z těchto tříd se volá metoda, která vytvoří vrchol s vlastnostmi dané entity. Entitou může být např. `Person, Experiment, DataFile`, a další.

Balík utils
Balík obsahuje jedinou třídu `Converter`, která primárně slouží pro převod mezi datovým typem *Date* a *String*. Tato třída obsahuje metody *convertFromDateToString()* a *convertFromStringToDate()*.

8 Testování a zhodnocení výsledků

První část této kapitoly je věnována samotnému testování relační databáze Oracle a grafové databáze Neo4j s cílem zjistit, který z testovaných databázových systému má flexibilnější datový model a rychleji vykonané dotazy. Všechny získané výsledky z první části kapitoly jsou zhodnoceny v druhé části kapitoly a na základně nich se rozhodne, zda-li se vyplatí použít grafovou databázi Neo4j nebo zůstat u relační databáze Oracle.

8.1 Testování

Testování je nedůležištější částí v této práci, které má pomoci rozhodnout, která ze dvou testovaných databází je vhodnější. Základní požadavky na výběr databáze jsou:

- **Flexibilní datový model** - možnost rozšíření současného datového modelu o další entity a změna stávající entity.
- **Rychlost vykonání dotazů** - dotaz by měl být vykonán rychleji než v dosavadní relační databázi Oracle

Proces testování probíhal na lokálním počítači z důvodu snažšího přístupu k databázovým serverům a v případě potřeby změnit nastavení těchto serverů. Dále bude uvedena konfigurace počítače, na kterém probíhalo testování.

Konfigurace počítače
Testování proběhlo na počítači s touto konfigurací:

- **Procesor**: Intel Core i7 CPU
- **Paměť RAM**: 4GB
- **Operační systém**: Window 7 Professional
- **Typ systému**: 64bitový operační systém
- **Hardisk**: WD 300GB

8.1.1 Flexibilita datového modelu

Datový model můžeme považovat za flexiblní, pokud ho lze rozšířit bez toho, aniž bychom porušili stávající koncept datového modelu. V případě relační databáze Oracle je flexibilita závislá na složitosti a provázanosti datového schématu, tudíž nemusí být jednoduché rozšířit stávající model o nějakou další entitu. V grafové databázi Neo4j máme více možností jak rozšířit stávající datový model o další entitu. První možností je vytvořit si nejprve novou třídu, která reprezentuje svými vlastnostmi danou entitu a tu prostřednictvím Java API přidat do stávajícího datového modelu. Druhou možností je použít některý z dostupných vizualizačních nástrojů, které byly popsány v kapitole 3. Pomocí těchto nástrojů lze poměrně snadno přidat další entitu do stávajícího datového modelu.

Dále si popíšeme jak rozšířit oba datové modely (Oracle, Neo4j) o novou entitu nazvanou `PHARMACEUTICAL`, která uchovává informace informace o lecích související s jednotlivými experimenty.

Oracle

V datovém schématu relační databáze Oracle vytvoříme novou tabulku s názvem `PHARMACEUTICAL` a poté přidáme nové sloupce (title, description), které reprezentující název a popis jednotlivých léků uložených v tabulce. Jelikož požadujeme, aby jeden záznam z tabulky `EXPERIMENT` mohl mít více záznamů z tabulky `PHARMACEUTICAL` a jeden záznam z tabulky `PHARMACEUTICAL` mohl patřit více záznamům z tabulky `EXPERIMENT` je nutné založit vazební tabulku s názvem `PHARMACEUTICAL_REL`, která bude obsahovat cizí klíč na tabulku `EXPERIMENT` a `PHARMACEUTICAL`.

Daleko větší problém je pokud bychom z tabulky `PHARMACEUTICAL` odstranili sloupec, který by používala jiná tabulka. Další problém může vzniknout při změně datového typu nějakého sloupce. V případě přidání nového sloupce do tabulky, kterému nastavíme defaultní hodnotu nebo ho nastavíme na nepovinný (not null) k problému nedojde.

Neo4j

Datový model grafové databáze Neo4j se skládá z vrcholů a hran a jeho rozšíření o další vrchol a hranu je jednoduší než v případě relačního datového sché-

matu. K rozšíření použijeme vizualizační nástroj *Neoclipse*, ve kterém nejprve vytvoříme nový typ hrany s názvem `REF_EXPERIMENT_PHARMACEUTICAL` a poté definujeme nový vrchol s vlastnostmi (title, description). Takto vzniklý vrchol napojíme do sítě stávajícího datového modelu.

V případě rozšíření o další sloupec se jedná jen o přidání nové vlastnosti včetně její hodnoty. Pokud bychom chtěli odebrat nějaký sloupec máme dvě možnosti. Buď nechat stávající sloupec a přidat jiný s tím, že ten starý se bude při dotazování ignorovat, a nebo ho vymazat a přidat nový sloupec jako v předchozím případě.

8.1.2 Rychlost vykonání dotazů

Rychlost vykonání dotazů je dalším požadavkem při výběru vhodného databázového systému. V kapitole porovnání relačních a nerelačních databázích byly popsány rozdíly mezi těmito databázemi, které se lišily hlavně v rychlosti provedení dotazů. V této části kapitoly je hlavním předmětem testování právě rychlost vykonaných dotazů, pro který bylo sestaveno pět dotazů a ty budou postupně vykonány nad relační databází Oracle a poté nad grafovou databází Neo4j. U každého dotazu bude uveden stručný popis, tzn. co který dotaz dělá, syntaxe obou dotazovacích jazyků (SQL, Cypher) a nakonec tabuka naměřených hodnot času stráveného vykonáním dotazu pro oba databázové systémy.

Celé testování probíhalo tak, že každý dotaz byl vykonán opakovaně pro různý počet záznamů, nejprve pro 1000 záznamů a poté pro 10 000 záznamů. Kvůli pamětové nebylo možné dotazovat se většího počet záznamů. Při pokusu o větší množství dat došlo ke zpomalení stroje, při kterém nebylo možné pokračovat v testování a bylo nutné ukončit spuštěný dotaz. Pro různý počet záznamů se dotaz vykonal pětkrát. Před samotným měřením bylo důležité rozhodnout o tom, jakým způsobem měřit čas vykonání dotazu. Na výběr je hodnota času vykonání dotazu, tzv. executing time, která představuje čas, za který se vykoná daný dotaz bez zobrazení množiny výsledků. Druhou možností je vzít čas vykonání dotazu a zobrazení výsledků dotazu, tzv. fetch time, který představuje čas, za který se daný dotaz vykoná a zobrazí se příslušná množina výsledků. Při tomto testování je důležitá hodnota vykonání dotazu a ne hodnota času zobrazení, jelikož vykreslení hodnot na výstup může mít za následek zpoždění, které nelze dopředu předpovídat. Jednotlivé hodnoty časů jsou uvedeny v sekundách.

Jako nástroj pro testování rychlosti vykonání dotazu v relační databázi Oracle byl použit *SQL Developer Oracle*, který se standardně používá pro práci s touto databází. Pro testování rychlosti vykonání dotazu v grafové databázi Neo4j byl zvolen nástroj *Web admin*, který je součástí knihovny Neo4j, a umožňuje vykonávat dotazy prostřednictvím konzole.

Dotaz č. 1 - Select persons

Dotaz vybere všechny osoby a ke každé osobě zobrazí informace o ni samotné včetně nejvyššího dosaženého vzdělání.

Oracle

```
SELECT p.person_id, p.givenname, p.authentication, p.surname,
p.date_of_birth, p.gender, p.email, p.phone_number, p.note,
p.username, p.password, p.authority, p.confirmed, p.fb_uid,
p.registration_date, p.laterality, el.title as
    education_level_title
FROM person p
INNER JOIN education_level el ON
  p.education_level_id = el.education_level_id
```

Výpis 8.1: Příklad skriptu pro výběr osob v *SQL*

	Jednotlivé kroky měření [s]					[s]
Počet záznamů	**1**	**2**	**3**	**4**	**5**	**Průměr**
1000	0,675	0,655	0,662	0,718	0,665	0,675
10 000	2,925	2,525	2,41	2,442	2,483	2,557

Tabulka 8.1: Tabulka s naměřenými časy vykonání dotazu nad Oracle.

Neo4j

```
START p = node:eegindex('_Type:entities.Person')
MATCH p-[:REF_PERSON_EDUCATION_LEVEL]->ed
```

```
RETURN p.Id, p.GivenName, p.Authentication, p.SurName,
p.DateOfBirth, p.Gender, p.Email, ed.Title as EducationLevel
p.Note, p.UserName, p.Password, p.Authority, p.Confirmed,
p.FbUID, p.RegistrationDate, p.Laterality, p.PhoneNumber
```

Výpis 8.2: Příklad skriptu pro výběr osob v *Cypheru*

	Jednotlivé kroky měření [s]					[s]
Počet záznamů	**1**	**2**	**3**	**4**	**5**	**Průměr**
1000	0,422	0,468	0,335	0,322	0,353	0,380
10 000	1,811	1,953	2,045	1,655	1,578	1,808

Tabulka 8.2: Tabulka s naměřenými časy vykonání dotazu nad Neo4j.

Dotaz č. 2 - Select experiment

Dotaz vybere všechny informace o experimentech a ke každému z nich zobrazí informace o něm samotném včetně entit, které jsou s ním v relaci.

Oracle

```
SELECT ex.experiment_id, ex.start_time, ex.environment_note, ex.
    end_time, ex.temperature, p.username as Owner, sp.username
    as SubjectPerson, w.title as Weather, sg.Title as
    SubjectGroup, rg.title as ResearchGroup, d.gain as
    Digitization, sc.title as Scenario, ec.impedance as
    ElectrodeConf, es.title, a.compensation
FROM experiment ex
INNER JOIN person p ON ex.owner_id = p.person_id
LEFT JOIN person sp ON ex.subject_person_id = sp.person_id
INNER JOIN scenario sc ON ex.scenario_id = sc.scenario_id
INNER JOIN weather w ON ex.weather_id = w.weather_id
INNER JOIN research_group rg ON ex.research_group_id =
  rg.research_group_id
INNER JOIN subject_group sg ON
  ex.subject_group_id = sg.subject_group_id
```

```
INNER JOIN electrode_conf ec ON
  ex.electrode_conf_id = ec.electrode_conf_id
INNER JOIN digitization d ON
  ex.digitization_id = d.digitization_id
INNER JOIN artefact a ON
   ex.artefact_id = a.artefact_id
INNER JOIN electrode_system es ON
  ec.electrode_system_id = es.electrode_system_id
```

Výpis 8.3: Příklad skriptu pro výběr experimentů v *SQL*

	Jednotlivé kroky měření [s]					[s]
Počet záznamů	**1**	**2**	**3**	**4**	**5**	**Průměr**
1000	3,97	4,167	3,575	4,018	3,973	3,941
10 000	43,17	35,155	30,89	36,385	38,365	36,793

Tabulka 8.3: Tabulka s naměřenými časy vykonání dotazu nad Oracle.

Neo4j

```
START exp = node:eegindex('_Type:entities.Experiment')
MATCH exp-[:REF_EXPERIMENT_OWNER]->ow,
  exp-[:REF_EXPERIMENT_SUBJECT_PERSON]->sp,
  exp-[:REF_EXPERIMENT_SCENARIO]->sc,
  exp-[:REF_EXPERIMENT_WEATHER]->w,
  exp-[:REF_EXPERIMENT_RESEARCH_GROUP]->rg,
  exp-[:REF_EXPERIMENT_SUBJECT_GROUP]->sg,
  exp-[:REF_EXPERIMENT_ELECTRODE_CONF]->ec,
  exp-[:REF_EXPERIMENT_DIGITIZATION]->dig,
  ec-[:REF_ELECTRODE_CONF_ELECTRODE_SYSTEM]->es
RETURN exp.Id,exp.StartTime, exp.EndTime, exp.Temperature,
    exp.EnvironmentNote, ow.UserName as Owner,
    sp.UserName as SubjectPerson, sc.Title as Scenario,
    w.Title as Weather, rg.Title as ResearchGroup,
    sg.Title as SubjectGroup, ec.Impedance as ElectrodeConf,
    dig.Gain as Digitization, es.Title as ElectrodeSys
```

Výpis 8.4: Příklad skriptu pro výběr experimentů v *Cypheru*

	Jednotlivé kroky měření [s]					[s]
Počet záznamů	**1**	**2**	**3**	**4**	**5**	**Průměr**
1000	0,458	0,415	0,361	0,387	0,376	0,399
10 000	2,855	3,000	3,261	3,675	3,238	3,206

Tabulka 8.4: Tabulka s naměřenými časy vykonání dotazu nad Neo4j.

Dotaz č. 3 - Select scenarios

Dotaz vybere všechny scénáře a ke každému scénáři zobrazí informace o něm samotném včetně výzkumné skupiny a správce scénáře.

Oracle

```
SELECT sc.scenario_id, sc.title, sc.scenario_length, sc.
    description, sc.private, sc.scenario_name, sc.mimetype, ow.
    username, rg.title as research_group_title
FROM scenario sc
INNER JOIN person ow ON
  sc.owner_id = ow.person_id
INNER JOIN research_group rg ON
  sc.research_group_id = rg.research_group_id
```

Výpis 8.5: Příklad skriptu pro výběr scénářů v *SQL*

	Jednotlivé kroky měření [s]					[s]
Počet záznamů	**1**	**2**	**3**	**4**	**5**	**Průměr**
1000	0,877	0,767	0,758	0,803	0,648	0,771
10 000	5,553	5,397	6,485	5,295	5,633	5,673

Tabulka 8.5: Tabulka s naměřenými časy vykonání dotazu nad Oracle.

Neo4j

```
START sc = node:eegindex('_Type:entities.Scenario')
MATCH sc-[:REF_SCENARIO_RESEARCH_GROUP]->rg,
      sc-[:REF_SCENARIO_OWNER]->ow
RETURN sc.Id, sc.Title, sc.Description, sc.ScenarioLength, sc.
    Private, sc.ScenarioName, sc.MimeType, rg.Title as
    ResearchGroup, ow.UserName
```

Výpis 8.6: Příklad skriptu pro výběr scénářů v *Cypheru*

	Jednotlivé kroky měření [s]					[s]
Počet záznamů	**1**	**2**	**3**	**4**	**5**	**Průměr**
1000	0,434	0,384	0,238	0,223	0,223	0,211
10 000	1,323	1,294	1,169	0,957	1,105	1,170

Tabulka 8.6: Tabulka s naměřenými časy vykonání dotazu nad Neo4j.

Dotaz č. 4 - Select research groups

Dotaz vybere všechny výzkumné skupiny a ke každé skupině zobrazí informace o ni samotné včetně počtu uživatelů v dané skupině.

Oracle

```
SELECT rg.title, rg.description, COUNT(p.person_id) as
    pocet_uzivatelu
FROM research_group rg
INNER JOIN person p ON
  p.person_id = rg.owner_id
GROUP BY rg.title, rg.description
```

Výpis 8.7: Příklad skriptu pro výběr výzkumných skupin v *SQL*

	Jednotlivé kroky měření [s]					[s]
Počet záznamů	**1**	**2**	**3**	**4**	**5**	**Průměr**
1000	1,067	0,759	0,938	0,803	0,899	0,893
10 000	2,228	2,052	3,086	1,847	2,717	2,386

Tabulka 8.7: Tabulka s naměřenými časy vykonání dotazu nad Oracle.

Neo4j

```
START rg = node:eegindex('_Type:entities.ResearchGroup')
MATCH rg-[:REF_RESEARCH_GROUP_OWNER]->ow
RETURN rg.Id, rg.Title, rg.Description, COUNT(ow.Id) as
    PocetUzivatelu
```

Výpis 8.8: Příklad skriptu pro výběr výzkumných skupin v *Cypheru*

	Jednotlivé kroky měření [s]					[s]
Počet záznamů	**1**	**2**	**3**	**4**	**5**	**Průměr**
1000	0,21	0,21	0,178	0,148	0,245	0,198
10 000	0,667	0,650	0,517	0,660	0,491	0,593

Tabulka 8.8: Tabulka s naměřenými časy vykonání dotazu nad Neo4j.

Dotaz č. 5 - Select electrode confs

Dotaz vybere všechny konfigurace elektrod, které mají menší impedanci než 500 a ke každé z nich zobrazí informace o ni samotné včetně místa osazení elektrod.

Oracle

```
SELECT ec.electrode_conf_id, ec.impedance, es.title as
    electrode_system
FROM electrode_conf ec
INNER JOIN electrode_system es ON
ec.electrode_system_id = es.electrode_system_id
WHERE ec.impedance < 500
```

Výpis 8.9: Příklad skriptu pro výběr konfigurací elektrod v *SQL*

Počet záznamů	**1**	**2**	**3**	**4**	**5**	**Průměr**
1000	0,327	0,296	0,359	0,375	0,395	0,350
10 000	2,373	2,561	2,42	2,39	2,36	2,421

Tabulka 8.9: Tabulka s naměřenými časy vykonání dotazu nad Oracle.

Neo4j

```
START ec = node:eegindex('_Type:entities.ElectrodeConf')
MATCH ec-[:REF_ELECTRODE_CONF_ELECTRODE_SYSTEM]->es
WHERE ec.Impedance < 500
RETURN ec.Id, ec.Impedance, es.Title as ElectrodeSystem
```

Výpis 8.10: Příklad skriptu pro výběr konfigurací elektrod v *Cypheru*

Počet záznamů	**1**	**2**	**3**	**4**	**5**	**Průměr**
1000	0,08	0,06	0,077	0,085	0,052	0,071
10 000	0,242	0,333	0,655	0,418	0,317	0,393

Tabulka 8.10: Tabulka s naměřenými časy vykonání dotazu nad Neo4j.

8.2 Zhodnocení výsledků

V předchozí části kapitoly byly otestovány dva požadavky na datový model relační databáze Oracle a nerelační grafové databáze Neo4j. Prvním požadavkem byla **flexibilita datového modelu**, která by umožnila snadné rozšíření datového modelu o další entitu. Druhým požadavkem byla **rychlost vykonání dotazu**.

Z hlediska flexibility datového modelu lze prohlásit, že u grafové databáze Neo4j je rozšíření datového modelu snadnější a přístupnější než u relační databáze Oracle. Důvodem je skutečnost, že v případě Neo4j stačí vytvořit nový vrchol a ten napojit do sítě vrcholů a pojmenovat vlastnosti. Tato změna nikterak nenarušuje současný koncept datového modelu. U relační databáze Oracle je tomu naopak, jelikož rozšíření stávajícího datového schématu o další tabulku vyžaduje postup, který je mnohdy komplikovanější.

Druhým požadavkem na testování byla rychlost vykonání dotazu nad oběma databázovými systémy. Testování rychlosti probíhalo tak, že každý z dotazů byl opakovaně vykonán a to vždy pro různý počet záznamů, tzn. 1000 a 10 000 záznamů, aby se ověřila výkonost databáze při výběru většího počtu záznamů. Pro detailnější pohled naměřených hodnot časů obsahuje tabulka 8.11 průměrné časy spuštěných dotazů v konkrétním databázovém systému (Oracle, Neo4j) a u každého z nich případné zrychlení.

U obou databázových systému byla provedena optimalizace a to především prostřednictvím vytvoření indexů. V případě databáze Oracle se jednalo o vytvoření indexů pro primární a cizí klíče. Dále také optimalizaci spuštěných dotazů, kterou nejlépe zajistí databázový engine. U grafové databáze Neo4j byly vytvořeny indexy pro všechny vlastnosti popisující vrcholy. Indexování slouží především ke zrychlení vykonávání dotazů.

Z výsledku těchto naměřených průměrných hodnot a urychlení je vidět, že vykonání dotazů nad grafovou databází Neo4j je řádově rychlejší než v případě relační databáze Oracle. Výrazně se tento rozdíl projevuje u dotazů, které přistupují k datům z jiných entit. Například u dotazu, který zobrazuje informace o experimentech a přitom přistupuje k datům z jiných entit, je urychlení Neo4j až 11x oproti relační databázi Oracle.

Velikost grafové databáze Neo4j má vliv na velikost operační paměti RAM. Tato paměťová závislost se projevuje hlavně při vykonávání dotazů, kdy je celá databáze vložena do operační paměti RAM, ale jen při prvním spuštění dotazu. Při dalším spuštění dotazu už jsou data přístupná jen z operační paměti RAM. Operační paměť je určená pro dočasné uložení dat a má rychlejší přístup než vnější paměť (např. pevný disk). Tato skutečnost je podložena v následujicí citaci z dokumentace, která je k dispozici na oficiálních stránkách Neo4j.

„*Neo4j tries to memory-map as much of the underlying store files as possible. If the available RAM is not sufficient to keep all data in RAM, Neo4j will use buffers in some cases, reallocating the memory-mapped high-performance I/O windows to the regions with the most I/O activity dynamically. Thus, ACID speed degrades gracefully as RAM becomes the limiting factor.*"[9]

V závěru zhodnocení testování je nutné rozhodnout, zda-li je vhodnější použít grafovou databázi Neo4j nebo zůstat u současné relační databáze Oracle. Z dosažených výsledků testování je patrné, že v obou kritériích (flexibilita, rychlost) je vhodnější použít grafovou databázi Neo4j. Jedinou překážkou může být paměťová závislost na fyzické paměti RAM, ale záleží na hardwaru daného stroje, na kterém databáze poběží. Z mého pohledu se Neo4j jeví jako vhodná pro použití v EEG/ERP portálu, ačkoliv se mi nepodařilo kvůli paměťové náročnosti zajistit úplně zatížění Neo4j databáze při větším počtu záznamů v dotazech. I tak si myslím, že tato knihovna má velkou budoucnost a uplatnění se na trhu.

Dotaz	**Oracle** [s]	**Neo4j** [s]	**Zrychlení**
select persons	0,675	0,380	1,41x
select experiments	36,793	3,206	11,48x
select scenarious	5,673	1,170	4,85x
select research groups	2,386	0,593	4,02x
select electrode confs	2,421	0,393	6,16x

Tabulka 8.11: Zrychlení jednotlivých dotazů pro 10 0000 záznamů.

9 Závěr

V rámci diplomové práce jsem nejprve prostudoval možnosti nerelačních databází a poté provedl srovnání s databázemi relačními. Zaměřil jsem se jen na databáze grafové a vybral tři zástupce (Neo4j, OrientDB a DEX). Z těchto tří zástupců jsem vybral databázi, která vyhovovala nejlépe a tou je Neo4j. V dalším kroku jsem se důkladně seznámil s touto grafovou databází a poté provedl analýzu stávajícího datového schématu EEG/ERP portálu a vybral vhodnou podmnožinu. Na této podmnožině relační databáze Oracle za pomoci nástroje Datanamic Data for Oracle jsem vygeneroval testovací data, která jsou důležité pro implementaci datového modelu grafové databáze Neo4j. Na základě této vybrané podmnožiny jsem navrhl datový model grafové databáze Neo4j a ten implementoval pomocí Java API.

Po implementaci jsem grafovou databázi Neo4j otestoval spolu s relační databází Oracle, kde bylo cílem rozhodnout, která z testovaných databází je vhodnější pro použití v EEG/ERP portálu. Testovaly se dva základní požadavky nad oběma databázema - flexibilita datového modelu a rychlost vykonání dotazu. Podle zhodnocení dosažených výsledku (viz předchozí kapitola) byla zvolena za vhodnou databázi právě grafová databáze Neo4j, která dosahuje lepších výsledků v testování požadavků na flexibilitu a rychlost vykonání dotazů. Samozřejmě se jedná jen o doporučení, které nemusí být realizováno z důvodů, které byly popsány v předchozí kapitole. Například se může jednat o paměťovou závislost databáze na velikosti operační paměti RAM.

Tato práce byla pro mě velmi zajímavá z ohledem na to, že jsem si vyzkoušel práci s relativně mladou, ale slibnou databází Neo4j, která se řadí do skupiny NoSQL databází. V současné době se NoSQL databáze stále rozvíjí a do budoucna se mohou stát velmi využívanou skupinou databází v mnoha oblastech. Hlavním důvodem je rostoucí objem dat a flexibilní datové schéma, na které již klasické relační databáze nestačí.

Výše uvedené body dokazují, že cíle diplomové práce byly úspěšně splněny a také se vymezuje proti oborovému projektu, který se zabýval studií tří grafových databází.

Literatura

[1] Eeg database. URL `https://github.com/INCF/eeg-database`. Navštíveno: 20.04.2013.

[2] Jacomy M Bastian M., Heymann S. Oficiální stránky vizualizačního nástroje gephi. URL `http://www.gephi.org`. Navštíveno: 22.04.2013.

[3] McCrory's Blog. Cap theorem and clouds. URL `http://blog.mccrory.me/2010/11/03/cap-theorem-and-the-clouds/`. Navštíveno: 08.05.2013.

[4] Ing. Petr Brůha. *Diplomová práce EEG/ERP portál a prostředky sémantického webu*. 2011.

[5] Luca Garulli. Oficiální stránky technologie orientdb. URL `http://www.orientdb.org/orient-db.htm`. Navštíveno: 15.04.2013.

[6] Richard Günzl. *Bakalářská práce na téma NoSQL databáze*. 2012.

[7] Aleksa Vukotic Jonas Partner. *Neo4j in Action*. Manning Publications, 2013. Navštíveno: 08.04.2013.

[8] Sergejus Barinovas | Microsoft MVP. Prezentace na téma nosql - what's that? URL `http://www.slideshare.net/sergejus/nosql-whats-that`. Navštíveno:: 08.05.2013.

[9] Inc Neo Technology. Oficiální stránky technologie neo4j. URL `http://www.neo4j.org/`. Navštíveno: 05.04.2013.

[10] Peter Neubauer. Graph databases, nosql and neo4j. URL `http://www.infoq.com/articles/graph-nosql-neo4j`. Navštíveno: 08.05.2013.

[11] The University of West Bohemia. Oficiální stránky eeg/erp portálu. URL `http://eegdatabase.kiv.zcu.cz/home.html`. Navštíveno: 10.04.2013.

[12] Amit Piplani. U pick 2 selection for nosql providers. URL `http://amitpiplani.blogspot.cz/2010/05/u-pick-2-selection-for-nosql-providers.html`. Navštíveno: 01.05.2013.

[13] Martin Strbačka. *Diplomová práce Vytvoření publikačního standardu v oblasti evokovaných potenciálů*. 2012.

[14] Sparsity Technologies. Oficiální stránky technologie dex. URL `http://www.sparsity-technologies.com/index/`. Navštíveno: 28.04.2013.

A Uživatelská dokumentace

Tato kapitola popisuje průběh celého procesu testování, od samotné instalace potřebných nástrojů až k testování požadavků. První část kapitoly se zabývá popisem instalace nástrojů pro samotné testování a druhá kapitola popisuje použití těchto nástrojů při testování. Na přiloženém CD se nacházejí nástroje využité v průběhu testování kromě Oracle serveru, který je ke stažení na ofiiciálních stránkách společnosti Oracle.

A.1 Příprava nástrojů pro testování

Ke zprovoznění celého procesu testování je důležité nainstalovat několik nástrojů, které mají na starosti nějakou činnost spojenou s procesem testování. Nejprve nainstalujeme Oracle server v aktuální verzi 11g případně verzi 10g, která je ke stažení na oficiálních stránkách společnosti Oracle. Po úspěšné instalaci je potřeba vytvořit lokální databázi. K tomu použijeme nástroj Database Configuration Assistant, který je součástí nainstalovaného Oracle serveru. V průběhu vytváření lokální databáze vytvoříme uživatele, jehož přihlašovací údaje budou sloužit pro přístup k vytvořené databázi. Dále specifikujeme název služby, pod kterou databáze poběží. Po tomto kroku máme vytvořenou databázi, ke které se připojíme například pomocí nástroje Oracle SQL Developer, který se používá ke správě databáze.

Na přiloženém CD se nachází složka *scripts*, která obsahuje skript pro vytvoření databáze a skripty testovacích dotazů pro oba databázové systémy. V této složce najdeme soubor s názvem *create_model.sq*, který otevřeme pomocí již zmíněného nástroje Oracle SQL Developer a poté spustíme. Tento skript nám vytvoří strukturu tabulek vybrané podmnožiny EEG/ERP portálu. Když máme vytvořené datové schéma podmnožiny EEG/ERP portálu, je nutné jej naplnit testovacími daty. Využijeme k tomu nástroj Datanamic Data for Oracle, prostřednictvím něhož vygenerujeme testovací data přímo do vytvořené databáze.

V tuto chvíli již máme vše potřebné pro otestování relační databáze Oracle, ale nejdříve vytvoříme grafovou databázi Neo4j. Ve složce *Apps* se nachází složka Neo4j Graph Database Creating, ve které je soubor *run.bat*, který spustí proces vytváření grafové databáze Neo4j. V tomto souboru je

nutné před spuštěním nastavit parametry pro připojení k relační databázi Oracle, tzn. přihlašovací jméno, heslo a název spuštěné služby. Ukázka spuštění procesu vytváření grafové databáze Neo4j včetně parametrů je níže.

```
Neo4JEEG.jar \%\%i [filePath] [userName] [password] [serviceName]
```

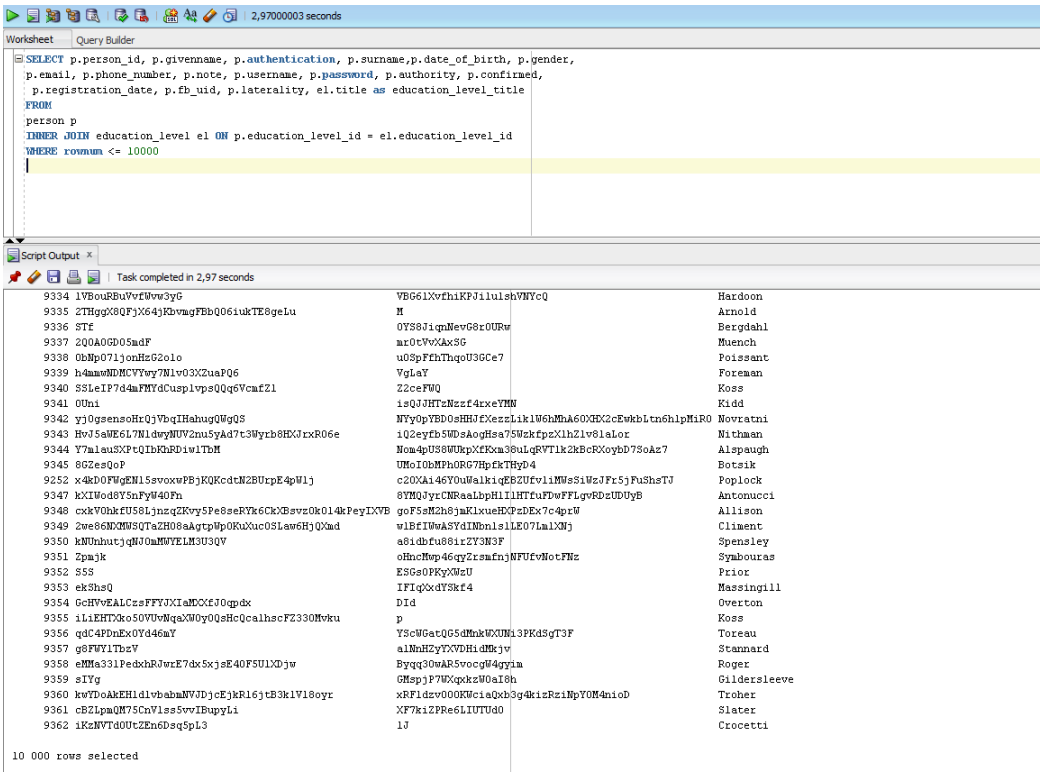
Spuštění aplikace probíhá opakovaně, a to vždy s jiným vstupním parametrem *%%i*, kerý označuje část dat, která se má načíst z relační databáze Oracle. Další parametry jako např. userName, password slouží pro připojení k databázi Oracle. Důvodem opakovaného spuštění aplikace je velké množství dat, které nebylo možné najednou načíst do paměti. Řešením je tedy postupné načtení části dat z relační databáze Oracle, které se použijí při vytváření grafové databáze Neo4j. Tato aplikace využívá k vytvoření grafové databáze prostředky, které jsou obsaženy v knihovně Neo4j.

A.2 Spuštění testování

V této fázi testování již máme vytvořené obě databáze a naplněné testovacími daty. Nyní můžeme otestovat oba požadavky, které byly zmíněné v kapitole testování. Pro upřesnění uvedu, že jde o požadavek na flexibilní datový model databáze a rychlost vykonání dotazu. Testovací scénáře dotazů pro oba databázové systémy jsou umístěné ve složce *scripts*.

A.2.1 Testování databáze Oracle

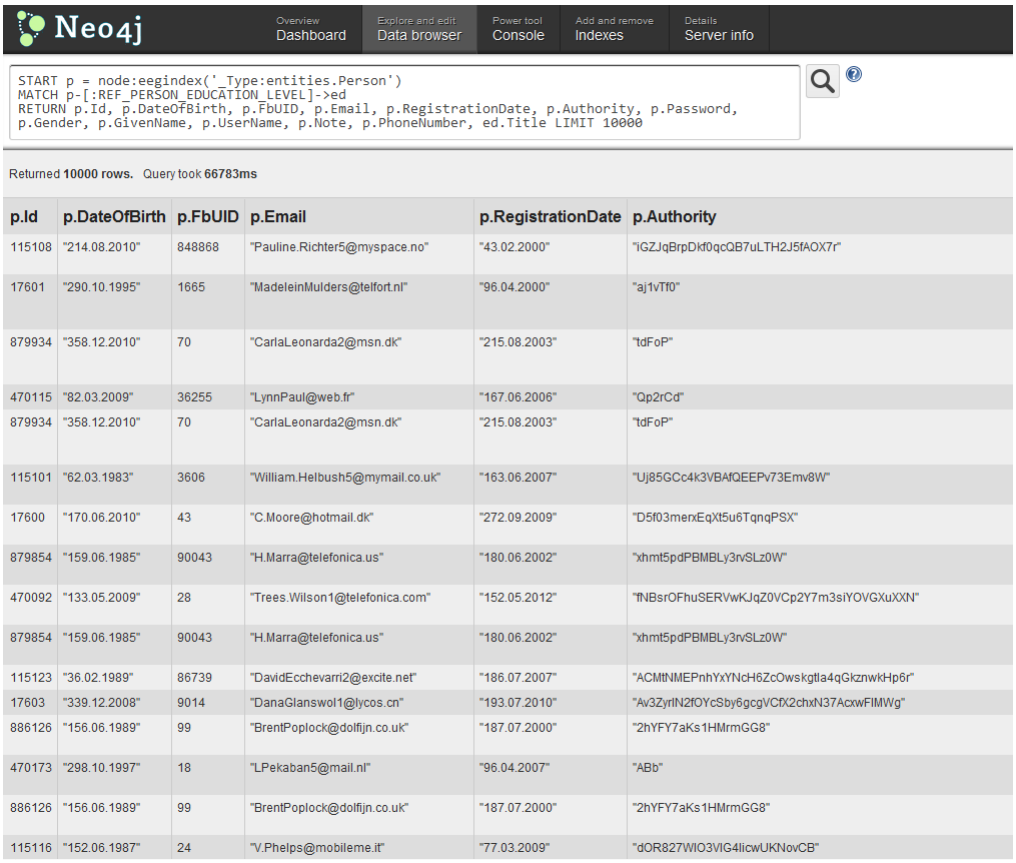
Nejdříve otestujeme relační databázi Oracle. K testování použijeme nástroj SQL Oracle Developer v němž otevřeme konkrétní skript s dotazem, který chceme otestovat. Například otevřeme soubor *select_person.sql* a spustíme dotaz, který nám vrátí seznam osob z tabulky `Person`. Na obrázku A1 je vidět ukázka testování tohoto dotazu, který byl vykonán pro 10 000 záznamů. V případě otestování flexibility datového schématu otevřeme skript s názvem *pharmaceutical_table.sq* a spustíme. Vytvoří se nám tabulka `PHARMACEUTICAL` a vazební tabulka `PHARMACEUTICAL_REL`.



Obrázek A.1: Ukázka použití nástroje SQL Oracle Developer při testování.

A.2.2 Testování databáze Neo4j

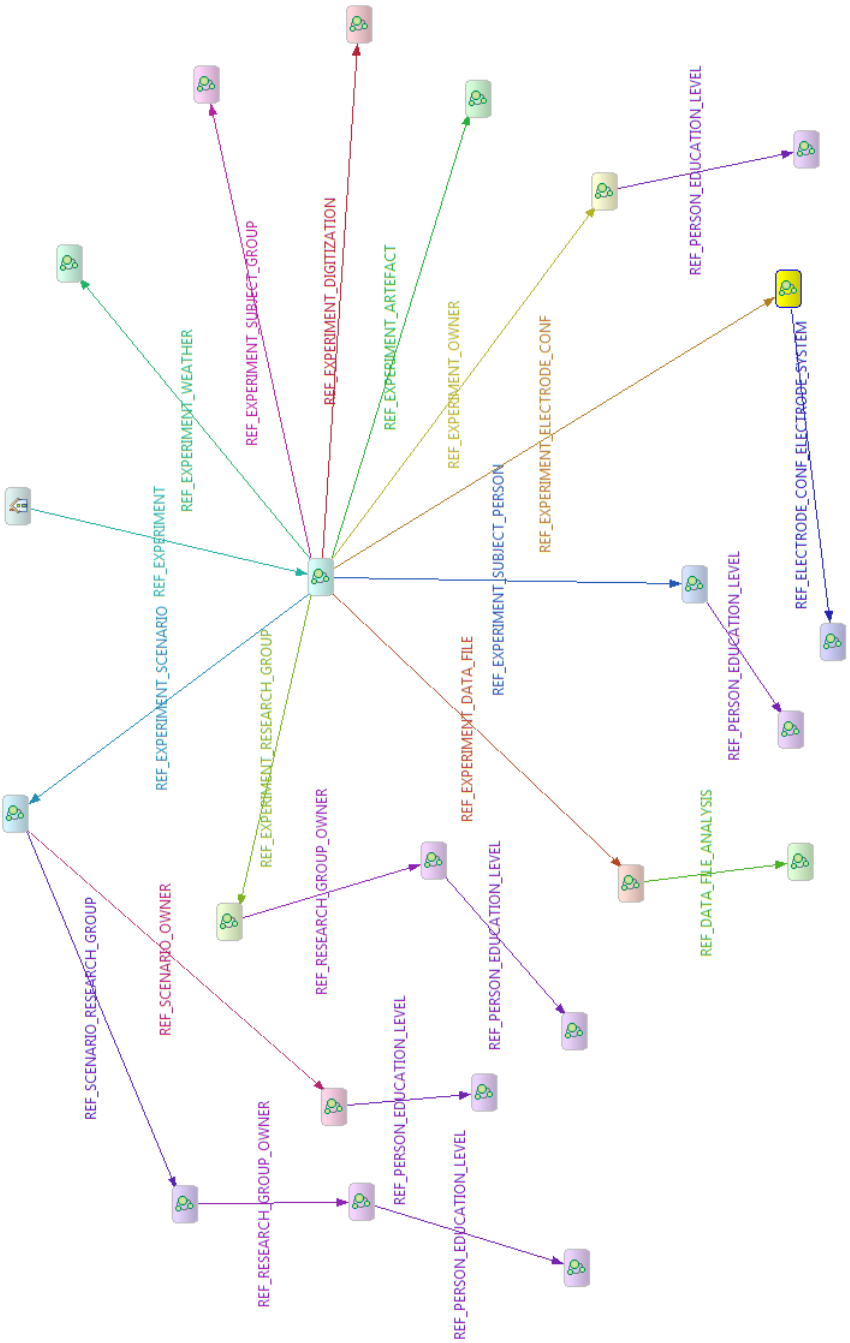
Pro otestování grafové databáze Neo4j použijeme webové rozhraní Web admin, které je součástí knihovny Neo4j umístěné ve složce *Apps*. Soubor ke spuštění tohoto webového rozhraní se nachází ve složce *bin* pod názvem *neo4j.bat*. Samotné webové rozhraní si zobrazíme pomocí prohlížeče, ve kterém zadáme adresu *http://localhost:7474*. Ve webovém rozhraní prostřednictvím konozle otestujeme rychlost vykonání dotazu, a to tak, že do konzole vložíme dotaz v jazyce Cypher. Dotaz vrátí 10 000 záznamů, které obsahují informace o lidech, kteří se podílejí na experimentech. Při prvním spuštění dotazu dojde k načtení celé grafové databáze do operační paměti RAM, proto je lepší změřit rychlost výkonu dotazu až při dalším spuštění. Ukázka spuštění dotazu v nástroji Web admin je vidět na obrázku A2.



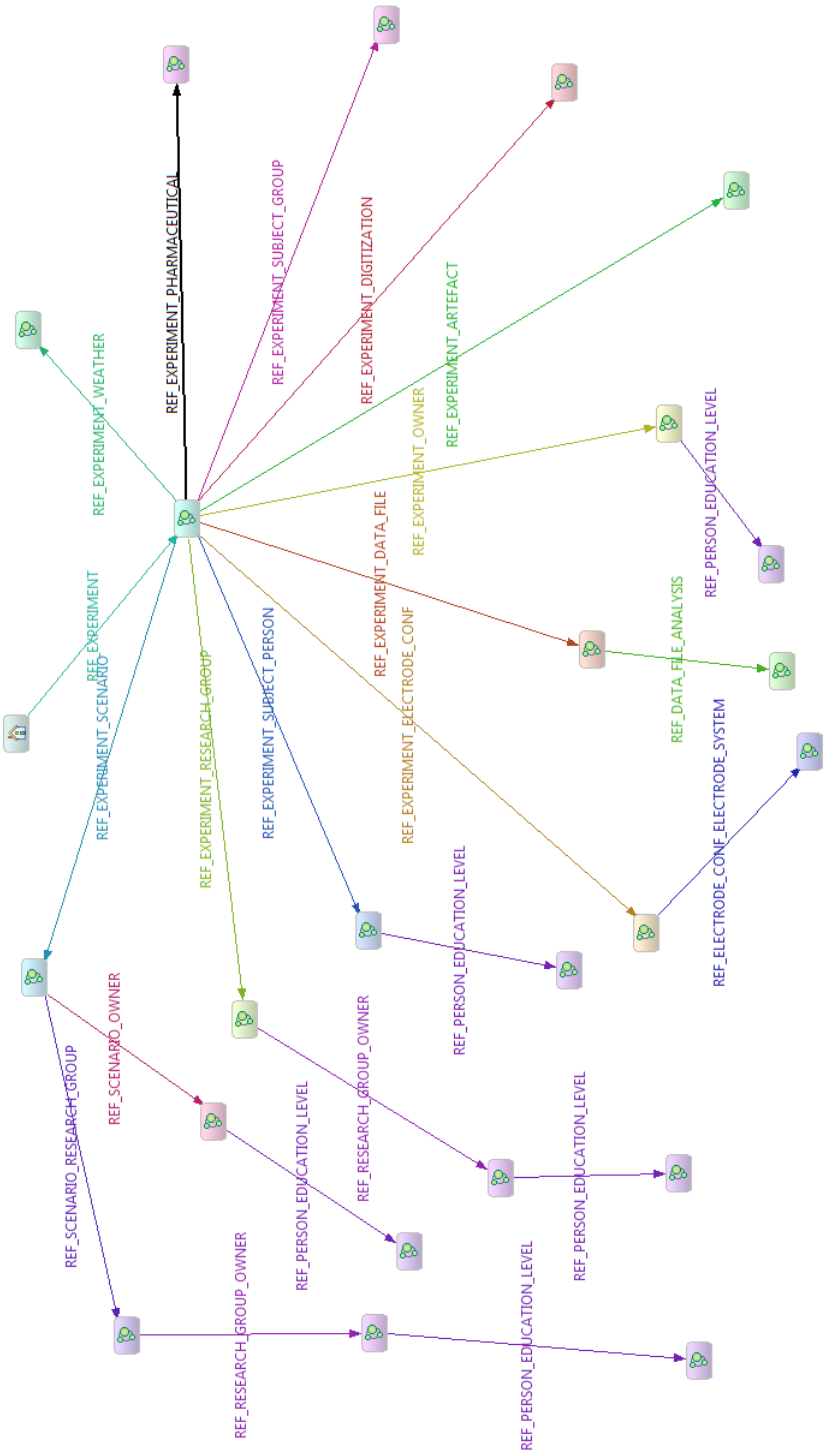
p.Id	p.DateOfBirth	p.FbUID	p.Email	p.RegistrationDate	p.Authority
115108	"214.08.2010"	848868	"Pauline.Richter5@myspace.no"	"43.02.2000"	"iGZJqBrpDkf0qcQB7uLTH2J5fAOX7r"
17601	"290.10.1995"	1665	"MadeleinMulders@telfort.nl"	"96.04.2000"	"aj1vTf0"
879934	"358.12.2010"	70	"CarlaLeonarda2@msn.dk"	"215.08.2003"	"tdFoP"
470115	"82.03.2009"	36255	"LynnPaul@web.fr"	"167.06.2006"	"Qp2rCd"
879934	"358.12.2010"	70	"CarlaLeonarda2@msn.dk"	"215.08.2003"	"tdFoP"
115101	"62.03.1983"	3606	"William.Helbush5@mymail.co.uk"	"163.06.2007"	"Uj85GCc4k3VBAfQEEPv73Emv8W"
17600	"170.06.2010"	43	"C.Moore@hotmail.dk"	"272.09.2009"	"D5f03menxEqXt5u6TqnqPSX"
879854	"159.06.1985"	90043	"H.Marra@telefonica.us"	"180.06.2002"	"xhmt5pdPBMBLy3rvSLz0W"
470092	"133.05.2009"	28	"Trees.Wilson1@telefonica.com"	"152.05.2012"	"fNBsrOFhuSERVwKJqZ0VCp2Y7m3siYOVGXuXXN"
879854	"159.06.1985"	90043	"H.Marra@telefonica.us"	"180.06.2002"	"xhmt5pdPBMBLy3rvSLz0W"
115123	"36.02.1989"	86739	"DavidEcchevarri2@excite.net"	"186.07.2007"	"ACMtNMEPnhYxYNcH6ZcOwskgtIa4qGkznwkHp6r"
17603	"339.12.2008"	9014	"DanaGlanswol1@lycos.cn"	"193.07.2010"	"Av3ZyrIN2fOYcSby6gcgVCfX2chxN37AcxwFIMWg"
886126	"156.06.1989"	99	"BrentPoplock@dolfijn.co.uk"	"187.07.2000"	"2hYFY7aKs1HMrmGG8"
470173	"298.10.1997"	18	"LPekaban5@mail.nl"	"96.04.2007"	"ABb"
886126	"156.06.1989"	99	"BrentPoplock@dolfijn.co.uk"	"187.07.2000"	"2hYFY7aKs1HMrmGG8"
115116	"152.06.1987"	24	"V.Phelps@mobileme.it"	"77.03.2009"	"dOR827WIO3VIG4licwUKNovCB"

Obrázek A.2: Ukázka použití nástroje Web admin při testování.

B Schéma grafové struktury v Neo4j



Obrázek B.1: Ukázka grafové struktury v Neo4j bez vlastností.



Obrázek B.2: Ukázka rozšíření grafové struktury o entitu *Pharmaceutical*

C Obsah přiložené CD

Na CD, jež je přiloženo k této diplomové prác, se nachází tato struktura:

- **Doc** - obsahuje elektronickou podobu této dokumentace
- **Apps** - v této složce jsou nástroje pro spuštění celého procesu testování
  - **Datanamic Data Generator for Oracle** - nástroj pro spuštění generování testovacích dat k relační databázi Oracle
  - **Neo4j Community 2.0.0.2** - knihovna práci s grafovou databází Neo4j obsahující nástroj *Web admin*
  - **Neo4j Graph Database Creating** - aplikace pro vytvoření grafové databáze podle vybrané podmnožiny EEG/ERP portálu
- **Scripts** - v této složce se nacházejí veškeré testovací skripty dostupné v jazyku SQL a Cypher

C.1 Struktura aplikace Neo4j Graph Database Creating

- **dist** - obsahuje spustitelný soubor s příponou *.jar*
- **run.bat** - soubor, který spustí aplikaci opakovaně s různým parametrem pro zadání začátku načítání dat z databáze
- **src** - složka obsahující zdrojové kódy aplikace
  - **entities** - balíček obsahující třídy reprezentující jednotlivé entity
  - **relationships** - balíček obsahující výčtový typ definující jednotlivé vztahy v grafu
  - **utils** - obsahuje jedinou třídu s pomocnými metodami