Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Možnosti využití grafových databází v informačním systému internetového obchodu

Bc. Luboš Palíšek

Vedoucí práce: Ing. Adam Šenk

29. dubna 2015

Poděkování

Mé poděkování patří vedoucímu této diplomové práce Ing. Adamovi Šenkovi za ochotu a čas, který mi po dobu řešení práce věnoval. Dále bych rád poděkoval své rodině za neuvěřitelnou podporu během celého studia. V neposlední řadě musím také poděkovat firmě KOBOZ SERVICE s.r.o. za umožnění spolupráce a vstřícné jednání.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 29. dubna 2015 .....

Odkaz na tuto práci

Palíšek, Luboš. *Možnosti využití grafových databází v informačním systému internetového obchodu.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Cílem této diplomové práce je najít vhodné využití grafové databáze v systému internetového obchodu. Práce se zabývá principy a výhodami grafových databází. Na základě těchto poznatků je popsán návrh, implementace a integrace systému pro doporučování produktů do existujícího internetového obchodu. Při vývoji byl použit databázový systém OrientDB. Část textu je věnovaná důvodům zvolení tohoto databázového systému a jeho specifickým vlastnostem. Součástí práce je kapitola věnovaná testování a ladění výkonnosti implementovaného systému.

Klíčová slova doporučovací systém, grafová databáze, NoSQL, OrientDB

Abstract

The aim of this master thesis is to find suitable usage of graph database in concrete e-shop. The work deals with the principles and advantages of graph databases. Based on these findings, the design and implmenetation of the system for product recommendation is described and integrated into existing e-shop. Graph database managment system OrientDB was chosen as the best suitable for this purpose. Part of the text summarizes reasons for choosing this database and covers its specific features. One chapter is about testing and tuning implemented system.

Keywords recommendation system, graph database, NoSQL, OrientDB

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

Úvod

Cíl práce

Tato práce má za úkol seznámit čtenáře s odvětvím NoSQL databází, které se specializuje na ukládání grafových struktur a dotazováním nad nimi. Grafové databáze jsou relativně mladá technologie, a proto v textu budou popsány základní principy grafových databází, jejich vlastnosti a používané implementace. V neposlední řadě se zaměřím na motivaci vývojářů pro jejich použití.

Dále bude čtenáři představena teorie doporučovacích systémů. Hlavním cílem práce je využít získané informace k návrhu systému pro doporučování produktů založeném na vybrané grafové databázi. Systém bude zaměřen na personifikované doporučování produktů pro jednotlivé zákazníky. Následně bude v práci popsán proces implementace takového systému a jeho integrace do stávajícího internetového obchodu.

Popis problému

V České republice bylo v roce 2014 přibližně 35 000 internetových obchodů [1], které musí bojovat o každého zákazníka. První fází úspěšné transakce je přivést uživatele na stránky internetového obchodu a zaujmout jeho pozornost. K tomu může sloužit pěkný design stránek, nízké ceny, detailní fotky nebo přehledně popsané parametry produktů. Provozovatel obchodu musí vynaložit práci a peníze, aby návštěvníka zaujal. Pokud se mu to povede a návštěvník na stránkách obchodu zůstane, je v zájmu provozovatele obchodu, aby se mu investované prostředky vrátily a zákazník utratil co nejvíce peněz za produkty z jeho nabídky. Druhou fází je zákazníkovi kromě zobrazení produktů, za kterými cíleně do internetového obchodu přišel, nabídnout i produkty, které by si s největší pravděpodobností mohl přikoupit a tím zvýšit hodnotu objednávky. Každý jednotlivec má jiné preference a proto je nutné vybírat nabízené produkty dynamicky na základě informací, které o sobě konkrétní zákazník zane-

chal chováním na stránkách obchodu.

Problému kvalitního a rychlého doporučování se věnuje teorie doporučovacích systémů, která bude v práci popsána a poté použita pro návrh a implementaci řešení.

Systém, který je součástí řešení této diplomové práce bude sloužit k doporučování produktů pro existující internetový obchod nazuby.cz[1]. Původně používaný algoritmus doporučování produktů funguje na základě náhodného vygenerování sady produktů z akční nabídky. Tento stav je nevyhovující, neboť nedokáže reagovat na aktuální potřeby zákazníka.

Nový systém by měl být implementován a integrován do stávajícího informačního systému. S ohledem na to se předpokládá použití stejného programovacího jazyka Java a frameworku Spring. Systém by měl shromažďovat a zprostředkovávat přehledné statistiky o kvalitě doporučování. Provozovatel obchodu by zároveň měl mít možnost na základě těchto statistik nebo vlastního rozhodnutí ovlivnit četnost doporučování jednotlivých produktů.

Členění práce

Práce je členěna do dvou hlavních částí. První z nich se věnuje teoretickým pojmům potřebným pro práci s grafovými databázemi a pro implementaci doporučovacího systému.

V druhé části bude čtenář seznámen s postupy, které byly použity při implementaci doporučovacího systému s použitím grafové databáze. Tato část popisuje výběr grafového databázového systému, seznamuje s architekturou naprogramovaného systému a jeho implementací. Následně je popsán postup při ladění výkonu zvolené grafové databáze OrientDB. Poslední část práce se zaměřila na měření kvality doporučovaných produktů.

[1] https://www.nazuby.cz

Kapitola 1

Teoretická část

1.1 Graf

Graf je struktura definovaná jako uspořádaná dvojice množin G=(V, E), kde V představuje množinu uzlů (někdy nazývanou množinou vrcholů) a E reprezentuje množinu hran. Hrana grafu je určena dvojicí uzlů. Hrana bez krajních uzlů nemůže existovat, a proto odebráním krajního uzlu zaniká. Graf se stává ohodnoceným, pokud hranám (nebo uzlům) přiřadíme hodnoty, například reálná čísla.[2]

1.1.1 Neorientovaná hrana

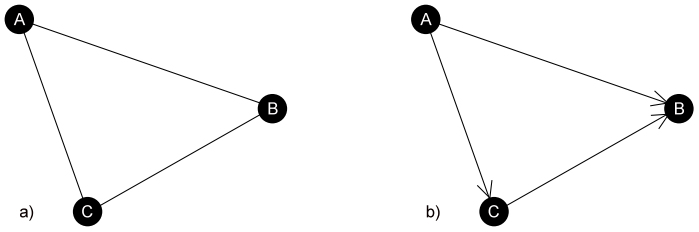
Neorientovaná hrana je reprezentována neuspořádanou dvojicí uzlů. Graf složený pouze z neorientovaných hran nazýváme neorientovaný graf (obrázek 1.1).

1.1.2 Orientovaná hrana

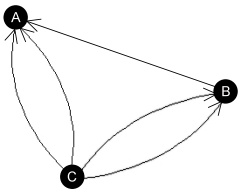
V případě, že hrana je zapsána jako uspořádaná dvojice uzlů, jedná se o orientovanou hranu a grafu složenému z takových hran říkáme orientovaný. Při grafickém znázornění grafu se orientovaná hrana zakresluje šipkou ve směru orientace (obrázek 1.1).

1.1.3 Multigraf

Teorie grafů zavádí pojem multigraf pro strukturu uzlů a hran, která může obsahovat vícenásobné neorientovane hrany nebo orientované hrany ve stejném směru mezi dvěma uzly. Ve specifikacích grafových databází se často pojem multigraf a graf zaměňuje.



Obrázek 1.1: a) neorientovaný graf b) orientovaný graf



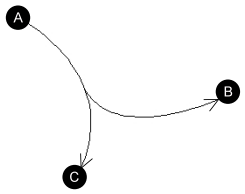
Obrázek 1.2: Multigraf

1.1.4 Hypergraf

Hypergraf je zobecněná forma grafu. Hrany se nazývají hyperhrany. V hypergrafu je povoleno, aby jedna hyperhrana spojovala libovolný počet uzlů (oproti grafu, kde je hrana definována přesně dvěma uzly). Příklad hypergrafu je zobrazen na obrázku 1.3.

1.1.5 Motivace pro studování grafů

Přestože se to může zdát jako odvážné tvrzení, lze říci, že s grafy se setkáváme v životě téměř všude. Graf jako reprezentace objektů a vztahů mezi nimi lze aplikovat na mnoho situací. Celá lidská společnost se dá charakterizovat jako velký graf, ve kterém jednotlivé osoby představují uzly a jejich vzájemné interakce hrany. Lidé, kteří jako první začali tuto strukturu využívat ke komerčním účelům, dnes patří k nejúspěšnějším na světě. Stačí si uvědomit, že přesně na principu vztahů mezi lidmi stojí většina sociálních sítí, které v posledních le-



Obrázek 1.3: Hypergraf

tech dokázaly změnit chování lidí na internetu. Z těch nejznámějších jmenuji Facebook[2], Twitter[3], LinkedIn[4].

Jiná globálně úspěšná společnost, která založila svůj úspěch na využití grafů je Google. Algoritmus PageRank, který stál za počátečními úspěchy internetového vyhledávače, vyhodnocoval kvalitu webových stránek na základě vztahů mezi nimi (hypertextovými odkazy).

Grafy lze použít i na menší problémy než výše popsané. Díky již objeveným algoritmům je lze například aplikovat při plánování optimálních tras v dopravě či optimalizacích toků nejen v počítačových sítích. Při řešení každého problému je obecně dobrým nápadem zamyslet se, zda by problém nešel převést na grafovou strukturu.

1.2 Vývoj databázových systémů

1.2.1 Databáze

Databáze je systematická struktura dat, která umožňuje, aby tyto informace mohly být vyhledávány počítačem [3]. Databázi organizuje speciální aplikace zvaná *systém řízení bází dat* (zkratka SŘBD) známá více pod anglickou zkratkou DBMS[5], mezi jejíž nejdůležitější funkce patří definice způsobu uchování informací, definování přístupu a možnosti úpravy uložených informací, popis datového modelu a dalších atributů. Dnes nejpoužívanější a nejpopulárnější implementace DBMS používají relační datové schéma založené na tabulkách. Relační databáze považuji za obecnou znalost čtenáře, a proto jejich popisu nebudu věnovat více prostoru.

[2] www.facebook.com
[3] www.twitter.com
[4] www.linkedin.com
[5] zkratka pro Database Managment software

1.2.2 Principy NoSQL databáze

V posledních letech[6] se ve světě informačních technologií začínají prosazovat DBMS založené na jiných datových modelech než relačních. Skupina takových databázových systémů se jednotně nazývá NoSQL databáze a dále se dělí podle způsobu, jak data uchovávají. Důvodem vzniku nových typů databázových systémů je masivní rozšíření internetových technologií v posledních letech a s tím spojený nárůst informací vygenerovaných uživateli. Datový model relačních databází přestává být dostatečně flexibilní a výkonný pro potřeby vývojářů a proto vznikají nová schémata, které jsou specifická pro konkrétní oblast použití.

Jednou z častých vlastností NoSQL databází je schopnost přizpůsobit se moderní metodice agilního vývoje, kdy se struktura uchovávaných dat může často měnit. Datové schéma bývá velmi volné a do databáze je tím pádem možné uložit libovolná data bez nutnosti znát například datový typ. Odpovědnost za dodržování datového schématu je tím přenesena z DBMS výhradně na aplikaci a jejího vývojáře.

Další motivací pro použití NoSQL databází je potřeba snadno a efektivně distribuovat úložiště dat mezi fyzicky oddělenými datacentry (neboli horizontální škálování). Problém horizontálního škálování řeší NoSQL databáze různými způsoby, které budou popsány později. V neposlední řadě je třeba zmínit, že jednotlivé NoSQL databáze se specializují na konkrétní případy užití a díky tomu mohou být pro tyto situace lépe optimalizované a dosahovat lepších výsledků než relační databázové systémy, které jsou navrženy pro obecné použití.

Důležitým rokem pro NoSQL databáze se stal rok 2000, kdy Eric Brewer prezentoval CAP teorém [5]. Myšlenka CAP teorému zavádí množinu tří základních vlastností, které jsou požadavkem na provozování distribuovaného systému[7]. Tyto vlastnosti jsou:[6]

- **Consistency** — zápis záznamu do databáze je atomický a dotaz na čtení dat z jakéhokoli uzlu distribuovaného systému vrátí vždy aktuální výsledek
- **Availability** — na každý požadavek je odeslána odpověď (nezáleží, jestli úspěšná nebo chybová) – neexistuje vypršení požadavku
- **Partition Tolerance** — systém nepřestává fungovat, přestože jeho část selhala nebo se přerušila komunikace mezi některými uzly

CAP teorém říká, že je možné implementovat a nasadit distribuovaný systém, který bude mít všechny tři výše uvedené vlastnosti, ale je nereálné, aby splňoval všechny tři požadavky souběžně.

[6] Pojem NoSQL byl poprvé použit v roce 1998 (Carlo Strozzi)[4], ale do širšího povědomí se dostal až kolem roku 2009

[7] vzájemně propojené výpočetní uzly sdílející stejná data

Například relační databázový systém MySQL v základní konfiguraci splňuje vlastnosti *consistency* a *availability*, ale nelze zde mluvit o *partition tolerance*, neboť distribuovatelnost se řeší obvykle replikací dat mezi servery. V případě, že uzly distribuovaného databázového systému mezi sebou musí komunikovat, aby provedly požadovanou operaci (každý uzel má pouze část informací), je nutné obětovat vlastnost *consistency* nebo *availability*.

Vývojáři jednotlivých NoSQL databázových systémů si zvolili různé způsoby, jak toto omezení implementují, a uživatel si musí být vědom, s jakými výhodami a nevýhodami se bude u konkrétní NoSQL databáze potýkat. Jako ukázku NoSQL databáze, která splňuje vlastnosti *consistency* a *partition tolerance*, lze uvést dokumentově orientovanou databázi MongoDB[8]. Naopak column-based databáze Apache Cassandra[9] je zaměřena na splnění *partition tolerance* a *availability*.

S novým přístupem k uchovávání dat, se kterým přišlo NoSQL se také ztrácí přísný požadavek na neustále konzistentní stav databáze zajištěný transakčností operací. Tato vlastnost je v relačních databázích známá pod zkratkou ACID[10].

ACID vlastnosti jsou ve většině případů nekompatibilní s požadavky na výkonnost, škálovatelnost a dostupnost rozsáhlého online systému. Proto NoSQL databáze přicházejí s alternativním pojetím stavu databáze skrytým pod zkratkou BASE[11] založeném na následujících principech:[7]

- **Basicaly Available** — říká, že systém je dostupný v kontextu CAP teorému
- **Soft state** — znamená, že datové úložiště nemusí být při zápisu konzistentní
- **Eventual consistency** — systém se do konzistentního stavu určitě v budoucnosti dostane

Vývojář používající NoSQL databázi si musí být vědom, jak je vlastnost BASE implementovaná v systému, se kterým pracuje a musí tomu přizpůsobit implementaci.

1.3 Typy NoSQL databází

Po zavedení CAP teorému zaznamenalo odvětví NoSQL databází velký rozmach. Vznikalo mnoho různých druhů NoSQL databází z nichž jmenuji ty nejznámější (v některých případech zachovávám anglické pojmenování, neboť české ekvivalenty nejsou ustálené).

[8] https://www.mongodb.org
[9] http://cassandra.apache.org
[10] anglická zkratka pro atomicitu, konzistenci, izolaci, trvanlivost
[11] Basicaly Available, Soft state, Eventual Consistency

1.3.1 Column based databáze

Základní datový model column based databází se skládá ze sloupce (*column*), do kterého je možné uložit klíč, hodnotu a časovou značku, pro nalezení nejnovějšího záznamu. Je nutné zdůraznit, že pojem sloupec, zde nemá stejný význam jako pojem sloupec v relačních databázích. Jednotlivé sloupce lze shlukovat do tzv. *column family*, které jsou indexovány na základě klíče řádku (*key row*). Všechny *column family* jsou podřazeny jednomu prostoru označovanému *keyspace*. Hlavní výhodou *column based* databází je rychlost s jakou jsou schopny zpracovávat gigabyty dat. Ze známých implementací *column based* DBMS jmenuji Apache Cassandra nebo Hbase.

1.3.2 Dokumentově orientované databáze

Každý záznam v dokumentově orientované databázi je reprezentován jako dokument. Dokument v tomto světě představuje strukturu dat v různých formátech (HTML, XML, JSON, PDF a další). Některé dokumentové databáze extrahují z dokumentu metadata, dle kterých poté vyhodnocují dotazy. Dokumenty jsou ukládány do databáze pod unikátními klíči. V dokumentově orientovaných databázích je ve většině případů vývojář zcela zodpovědný za strukturu uložených dokumentů a není ze strany DBMS omezován (rozdíl oproti relačním databázím). Největší výhodou těchto databází je rychlost při vyhledávání záznamů. Jako příklad dokumentově orientovaných databází uvedu MongoDB a Apache CouchDB.

1.3.3 Objektové databáze

Záznamy v objektově orientovaných databázích si lze představit jako instance tříd v objektově orientovaném programování (dále jen pod zkratkou OOP) včetně známých vlastností OOP jako je zapouzdření, nebo přetěžování metod. Objekty jsou v databázi uchovávány ve stejném formátu, jako jsou definovány v kódu programu, čímž usnadní programátorovi jejich persistenci. Oproti relačním databázím odpadá složité spojování tabulek a načítání celých objektů z různých částí databáze. Tento typ databází byl populární v komerční sféře ještě před rozšířením popularity ostatních NoSQL []. Ze známých DBMS jmenuji ObjectDB a Db4o.

1.3.4 Grafové databáze

Posledním typem NoSQL databází, kterým se bude tato práce věnovat po zbytek textu jsou grafové databáze. Vzhledem k tomu, že se jedná o klíčový typ NoSQL databáze s ohledem k tématu práce, bude jejich popisu věnovaná celá další kapitola.

1.4 Grafové databáze

Grafová databáze je označení pro databázový systém, který používá jako datové schéma grafovou strukturu. Tento typ databází je určen pro použití v OLTP aplikacích označující systémy, které pracují s daty online za pomoci transakcí. Některé DBMS implementace označující se jako grafové databáze pracují navenek s grafy, ale interně data serializují do jiné struktury (například relační databáze nebo objektově orientované databáze). Těmito DBMS se dále nebude tento text věnovat, neboť nenabízí výhody, které mají „opravdové“ grafové databáze.

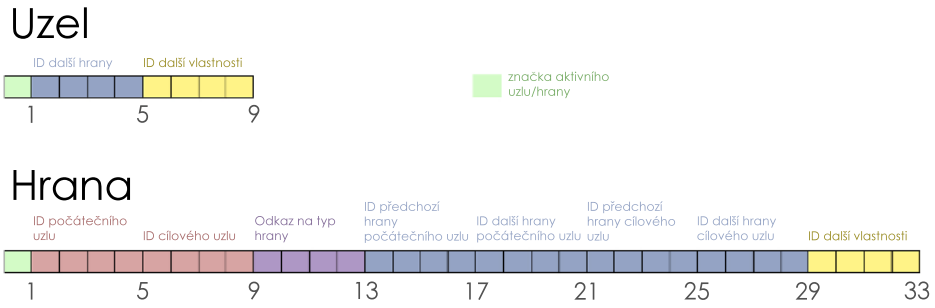
Znakem řadící DBMS mezi „opravdové“ grafové databáze je vlastnost nazývaná jako *index-free adjacency*[12]. Tato vlastnost říká, že vztah mezi dvěma uzly je možné najít a použít bez nutnosti prohledávat globální indexy, neboli že každý uzel má přímý odkaz na své sousední hrany nebo uzly. Toto je zásadní rozdíl proti relačním databázím, které ke spojování dat z více tabulek používají cizí klíče a výpočetně drahý SQL příkaz JOIN. Díky *index-free adjacency* jsou grafové databáze schopné mnohem efektivněji a rychleji prohledávat data na základě jejich vzájemných vztahů než v případě relačních databází, protože přechod mezi sousedními uzly není závislý na velikosti dat (a indexů nad nimi) a je konstantní $\mathcal{O}(1)$ [8]. Ve zjednodušené formě si lze představit, že příkaz JOIN známý z relačních databází se v grafové databázi provede pouze jednou při vytváření vztahu mezi uzly.

Grafové databáze implementují vlastnost *index-free adjency* ve většině případů seznamem sousedů[13]. Pro získání konkrétního uzlu nebo hrany v konstantním čase se často používá metoda označování uzlů nebo hran unikátním identifikátorem, který zároveň určuje fyzické umístění uzlu nebo hrany v datovém úložišti. Například databáze Neo4j používá oddělené soubory pro uzly, hrany, indexy a konstantní velikosti záznamů pro jednotlivé typy dat. Díky tomu je možné snadno dopočítat umístění hledané informace v souboru. Seznamy hran a uzlů jsou reprezentovány spojovými seznamy. Strukturu záznamu uzly a hrany v Neo4j popisuje obrázek 1.4.[9]

Jiný příklad grafové databáze, OrientDB, označuje uzly a hrany pomocí identifikátoru Record ID (RID). Databáze zároveň spravuje mapovací soubor, který přiřazuje RID ke konkrétním pozicím v datových souborech. RID hran jsou ukládány ve formě seznamů přímo do uzlů, ze kterých vycházejí.

[12] Pro tento výraz není v českém jazyce ustálené slovní spojení. Ve volném překladu znamená sousednost bez použití indexů.

[13] Seznam sousedů je typ reprezentace grafu. Každý uzel si uchovává informaci o hranách/uzlech, se kterými sousedí ve vlastním interním seznamu. Tato reprezentace je vhodná zejména pro řídké grafy.



Obrázek 1.4: Struktura uzlu a hrany v Neo4j

1.4.1 Datový model v grafových databázích

Nejčastější datový model používaný grafovými databázemi je tzv. *property graf*. Jedná se o graf doplněný o vlastnosti. Uzlu nebo hraně můžou být přiřazeny vlastnosti, které jsou reprezentované dvojicemi klíč–hodnota. Klíč obsahuje textový řetězec a hodnota může být libovolného datového typu. Vlastnosti uzlů definují entity a vlastnosti hran slouží jako atributy vztahů mezi nimi. Takový datový model je intuitivní a dobře představitelný pro uživatele. Často bývá graf s vlastnostmi doplněn o štítky. Uzly lze pomocí těchto štítků označovat a tím seskupovat do množin.

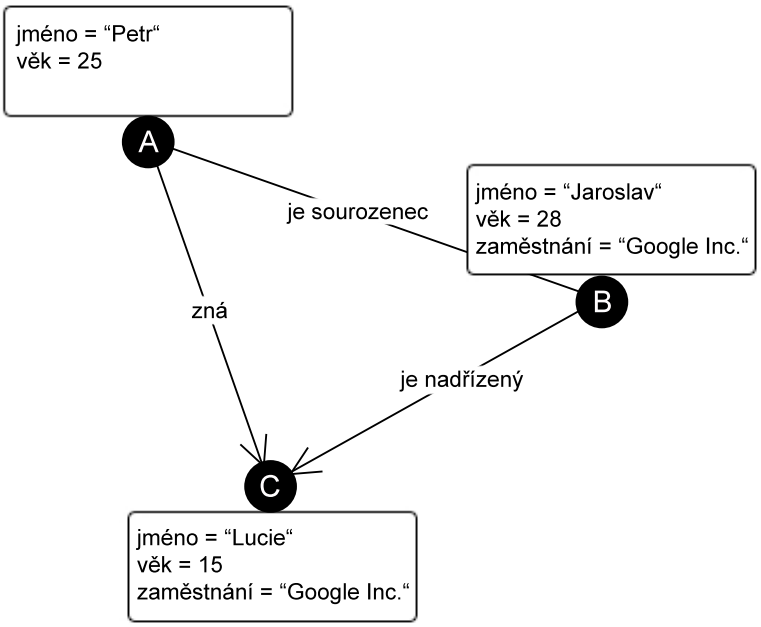
Mezi další používaný datový model patří hypergraf. Za grafovou databázi by mohl být považován i RDF graf složený z RDF triples známých ze sémantického webu [10].

Práci s datovým modelem v grafových databázích se snaží sjednotit projekt Blueprints[14]. Jedná se o rozhraní, které unifikuje přístup k datům v grafu a tím programátorovi usnadňuje přechod mezi jednotlivými databázovými systémy. Toto rozhraní implementují oblíbené grafové databázové systémy Neo4j, OrientDB, Titan a další. Blueprints je součást komplexního frameworku zvaného Tinkerpop, jež se soustřeďuje na zobecnění práce s grafovými databázemi.

1.4.2 Dotazování

Dotazovací jazyk SQL je velmi rozšířeným jazykem, který umožňuje pracovat s daty v relačních databázích. Vzhledem k tomu, že byl vyvinut pro potřeby tabulkových schémat, není zcela vhodný pro práci s grafovými databázemi. Nicméně SQL patří mezi základní znalosti programátorů a správců databází.

[14] http://blueprint.tinkerpop.com



Obrázek 1.5: Graf s vlastnostmi. Uzlům A, B, C by mohl být přiřazen štítek „Osoba“, protože se očividně jedná o stejný typ uzlu.

Právě díky jeho popularitě byl v lehce pozměněných podobách implementován i do některých grafových databází (OrientDB).

Pro pokročilé uživatele grafových databází se SQL stává nedostačujícím prostředkem zejména pro dotazování a je nahrazeno jiným jazykem nebo frameworkem určeným speciálně pro grafové struktury. Pro získávání dat z grafu je typické použít průchod grafem[15].

Následující odstavce popisují dva nejznámější jazyky pro definici průchodu grafem Cypher a Gremlin. Popisy jazyků jsou doplněny o příklad, na kterém je demonstrován zápis dotazu. Datový model k těmto příkladům lze nalézt v příloze A.1.

1.4.2.1 Cypher

Cypher je jazyk pracující s daty v grafech, který je vyvíjen spolu s grafovou databází Neo4j a jeho použití je zároveň podmíněno použitím databáze Neo4j.

[15] návštěva uzlů grafu následováním vycházejících hran dle daných pravidel

Jeho autoři ho označují jako deklarativní dotazovací jazyk, nicméně kromě dotazů zvládá i ukládání, mazání a upravování dat. Jazyk se soustředí na expresivní zápis, neboť jeho podoba má připomínat grafické znázornění požadovaného průchodu grafem. Uzly jsou zapisovány do kulatých závorek, které mají připomínat kruh, hrany mezi uzly jsou reprezentovnány pomocí šipek „->“.

Dotaz v jazyku Cypher se skládá ze tří základních částí. První z nich, MATCH, je povinná a určuje vzor (uzly, hrany), který se má z grafu vybrat. Poté následuje nepovinná část WHERE, která filtruje uzly a hrany získané z první části dotazu na základě jejich vlastností. Poslední složka dotazu, která je opět povinná, je RETURN. Tato část definuje, co dotaz vrátí (uzel, hranu, hodnotu parametru nebo jejich kombinace). Výsledky dotazu lze dále řadit, omezovat jejich počet, aplikovat na ně funkce a další operace. Kompletní přehled vlastností jazyka Cypher lze nalézt v jeho dokumentaci[16].

Následující dotaz v jazyce Cypher vyjadřuje dotaz: „Najdi všechny hudební festivaly pořádané roku 2014, na kterých se mohli potkat hudebníci Ozzy Osbourne a Bruce Dickinson nezávisle na kapele, ve které hráli.“

```
1  MATCH (ozzy:Person {name:'Ozzy Osbourne'})
2          -[:member]->
3                  (band)
4                          -[:performed]->
5                                  (festival:Festival)
6                          <-[:performed]-
7                  (band)
8          <-(bruce:Person {name:'Bruce Dickinson'})
9  WHERE festival.year = 2014
10 RETURN festival.name
```

Zdrojový kód 1.1: Ukázka dotazu v jazyku Cypher

1.4.2.2 Gremlin

Gremlin je jazyk pro dotazování a manipulaci s daty v grafových databázích, který není závislý na použití konkrétního databázového systému. Tento jazyk je vyvíjen skupinou TinkerPop[17] jako součást komplexního frameworku, který se snaží sjednotit rozhraní pro práci s daty v grafových databázích. Gremlin umí pracovat s takovými databázovými systémy, které implementují již zmíněné datové rozhraní Blueprints.

Jazyk Gremlin nebyl navržen s takovým důrazem snadné čitelnosti dotazů jako Cypher. Zápis se skládá z jednotlivých kroků, které je možné za sebe sklá-

[16] http://Neo4j.com/docs/stable/
[17] http://www.tinkerpop.com

dat a tím vytvářet složitější průchod grafem. V dotazech se místo grafických prvků používají názvy funkcí (např *in()* pro vstupující hranu) a zkratky (E pro hrany, V pro uzly). Zápis je v této podobě velmi krátký a přesto srozumitelný. Zároveň je možné lépe specifikovat přesný průchod grafem.

Pro ukázku dotazovacího jazyka Gremlin byl zvolen stejný dotaz jako pro Cypher. 1.1

```
1 g.V('name', 'Ozzy Osbourne')
2         .out('member')
3                 .out('performed')
4                         .as('festival')
5                 .in('performed')
6         .in('member')
7         .has('name', 'Bruce Dickinson')
8                 .back('festival')
9                         .has('year', 2014).name;
```

Zdrojový kód 1.2: Ukázka dotazu v jazyku Gremlin

1.4.3 Použití grafových databází

V kapitole 1.1.5 jsem nastínil motivaci pro aplikaci grafových struktur na řešení některých úloh. Grafové databáze jsou, stejně jako obecně všechny typy NoSQL databází, určené pro specifickou kategorii problémů. Z tohoto důvodu je logické, že není vhodné bezhlavě transformovat všechna existující fungující relační schémata do grafových databází (nebo jiných NoSQL databází). Pro použití grafové databáze musí být jasné opodstatnění.

Pokud programátor vyvíjí systém s požadavkem na zpracovávání dat v reálném čase a při analýze zjistí, že entity v datovém modelu jsou silně provázané různými vztahy, je to první indikátor k zamyšlení nad použitím grafové databáze. Vysoká výkonnost při práci s vzájemně propojenými daty je jejich největší výhodou. Ostatní vlastnosti, ke kterým patří snadná horizontální škálovatelnost nebo velká flexibilita datového schématu se vztahují obecně na většinu NoSQL databází.

Další výhodou, která je však podmíněna kvalitním návrhem datového modelu, je získání nového pohledu na uložená data. Zkoumání vztahů mezi daty může vést k nalezení nových informací, například lepšímu porozumění chování uživatelů v aplikaci. Tyto nově získané znalosti lze použít ke zlepšení služeb vyvíjeného systému.

Přestože jsou grafové databázové systémy mladou technologií, jejich výhody přesvědčily už mnoho společností k jejich nasazení do produkčního prostředí. Na oficiálních internetových stránkách nejpopulárnější [11] grafové databáze Neo4j je možné najít případy užití ve společnostech, které ji imple-

mentovali pro podporu svých procesů. Patří mezi ně například obchodní portál eBay[18], který grafovou databázi používá k optimalizaci doručovacích cest kurýrů. Dle slov senior programátora Volkera Pachera z eBay se díky použití grafové databáze Neo4j proces více než tisícinásobně zrychlil oproti předchozí verzi používající relační databázi MySQL a to za potřeby 10–100x méně programovacího kódu.[12]

Jiná velká obchodní společnost Walmart[19] používá grafovou databázi pro online doporučování produktů svým zákazníkům. V prohlášení vývojáře společnosti Walmart implementující tento systém stojí: „Neo4j nám pomohla porozumět chování našich zákazníků a vztahům mezi nimi a nabízenými produkty. Tímto jsme získali perfektní nástroj pro doporučování produktů v reálném čase“.[12]

1.4.4 Použití v systému internetového obchodu

Internetové obchody vedou nekonečný souboj o zákazníky s tradičními kamennými prodejnami. Oba principy nakupování mají své výhody i nevýhody. Jedním z klíčových rozdílů mezi přímým prodejem v kamenném obchodě a internetovým prodejem je způsob komunikace mezi obchodníkem a zákazníkem.

V kamenné prodejně má zákazník k dispozici odborný personál, který s ním osobně komunikuje, aby mu poradil s výběrem produktu nebo doporučil vhodné příslušenství. Z pohledu obchodníka je to nesporná výhoda, neboť se ocitá v roli, kdy je schopen přímo ovlivňovat mínění zákazníka. Z osobního setkání a rozhovoru lze o zákazníkovi získat mnoho informací, které obchodník může použít ke zvýšení svého zisku (doporučením produktů s vyšší marží, přemluvením zákazníka k přikoupení dalších produktů a podobně).

Naproti tomu internetové obchody nemají možnost přímé komunikace se zákazníkem, pokud se sám zákazník nerozhodne kontaktovat prodejce například telefonicky pomocí zákaznické linky. Je to zároveň jeden z důvodů vysoké popularity této formy prodeje, protože zákazník se necítí být do ničeho nucen. Nicméně z pohledu prodejce je tato situace nevýhodná, neboť bez možnosti přímé komunikace s konkrétním návštěvníkem nezná jeho potřeby a musí se k němu chovat jako ke každému jinému zákazníkovi. Internetový obchod by měl být z tohoto důvodu optimalizovaný tak, aby měl každý návštěvník jeho webových stránek vždy dostatek relevantních a potřebných informací.

Zákazníci však přicházejí na internetové stránky obchodu s individuálními potřebami a proto nelze staticky přednastavit obsah webových stránek tak, aby byli všichni spokojeni. Je nutné najít způsob, jak co nejpřesněji napodobit komunikaci mezi zákazníkem a prodejcem. Populární přístup k tomuto problému je shromažďovat informace o návštěvníkovi z jeho chování na stránkách. Akce, které uživatel na webu provede, jsou obchodem průběžně ukládány

[18] http://www.ebay.com
[19] http://www.walmart.com/

a z těchto informací se vytvoří profil uživatele. Za pomocí tohoto profilu má internetový obchod možnost vytvořit předpoklady o budoucím chování uživatele a na jejich základě dynamicky změnit obsah stránek, které tomuto uživateli zobrazí.

Personalizované doporučování obsahu je velmi důležitá vlastnost internetového obchodu. Internetové obchody nabízejí velké množství produktů a bez zapojení doporučovacího systému musí zákazníci vynakládat velké úsilí, aby si vybrali z nabídky ten nejvhodnější produkt. Na základě tohoto faktu studie [13] ukazuje, že použití doporučovacího systému, který usnadní zákazníkovi nalézt vhodný produkt, zvýší pravděpodobnost na přikoupení dalších produktů tímto zákazníkem. Dále je ve stejné publikaci experimentálně ověřeno, že doporučovací systém může mít vliv na zvýšení počtu objednávek u produktů, které nepatří mezi nejprodávanější nebo nejpopulárnější, protože jsou zákazníkům díky doporučovacímu systému lépe dohledatelné.

1.5 Doporučovací systémy

V dnešní době uživatelé internetu každý den vygenerují obrovské množství informací, které jsou často duplicitní nebo neúplné. Pokud má člověk potřebu najít konkrétní odpověď na svůj dotaz, často se stává, že je zahlcen velkým počtem nerelevantních a nechtěných výsledků. V takové situaci je velmi obtížné najít nejlepší možný výsledek na hledaný dotaz. Nemusí se jednat pouze o situace, kdy uživatel cíleně vyhledává informace na základě klíčových slov. Dalším příkladem může být případ, kdy si uživatel přečetl článek a chtěl by pokračovat na další texty s podobným tématem, nicméně je pro něj složité takové články v množství informací na internetu najít. Z tohoto důvodu se v posledních letech zvýšila poptávka po systémech, které by dokázaly nechtěný obsah pro uživatele filtrovat a případně předvídat, jaký obsah by uživatel mohl chtít vidět. Tyto systémy se nazývají doporučovací sytémy a dají se použít v různých situacích jako například na zpravodajských portálech, internetových obchodech, portálech pro sledování videa, poslouchání hudby a dalších. Tato práce je zaměřena na použití doporučování v internetovém obchodě, z tohoto důvodu budu dále popisovat zejména toto odvětví.

Pro doporučování obsahu je nutné získat podklady, podle kterých systém vyhodnocuje, jaké položky jsou pro uživatele relevantní. Doporučování je obvykle personalizované a tím pádem každý uživatel získá jiné položky. Doporučovací systémy se proto dělí na dva základní typy.

1.5.1 Doporučování na základě obsahu

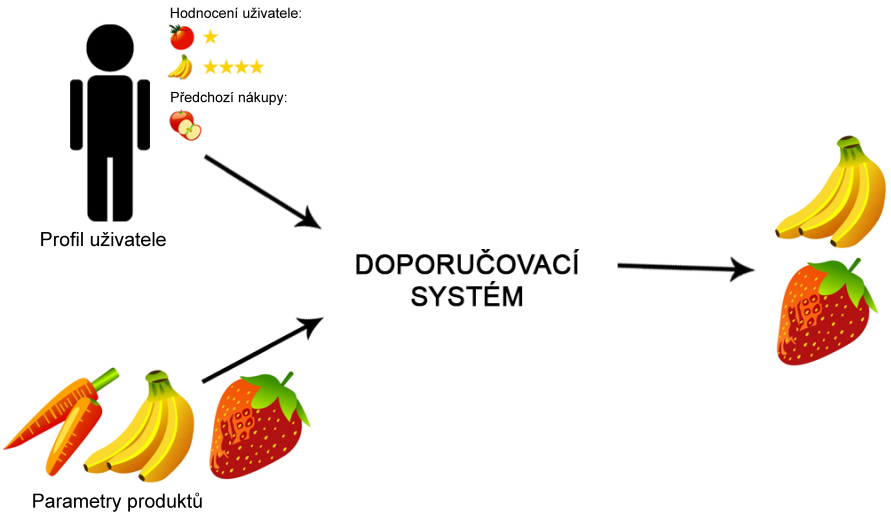
Princip doporučování na základě obsahu spočívá v doporučování položek, které jsou podobné s položkou, na kterou uživatel již reagoval (prohlížel ji, zakoupil ji). Podobnost může být vyhodnocena na základě parametrů definovaných u položky (u knih například autor, žánr, klíčová slova) nebo například podle

obsahu dokumentu (u knih analýzou textu). Při tomto typu doporučování je typické vytvořit profil uživatele a ten aplikovat pro přesnější výběr podobných produktů.[14]

Jako příklad uvedu návštěvníka internetového obchodu s ovocem a zeleninou. Příklad je podpořen obrázkem 1.6. Uživatel v minulosti udělal jednu objednávku a nakoupil jablka. Během prohlížení produktů dal najevo své preference tím, že banány ohodnotil kladně (5 bodů) a naopak rajčata ohodnotil nejmenším počtem bodů. Tím se vytvořil profil uživatele, který slouží spolu s parametry jednotlivých produktů jako vstupní data do doporučovacího systému. Doporučovací systém vyhodnotí uživatelův profil a doporučí mu z nabídky produkty na základě podobných parametrů.

V tomto případě jsou uživatelovi nabídnuty banány (z důvodu kladného hodnocení) a jahody, které jsou stejně jako jablka a banány z kategorie ovoce.

Kvalita doporučování závisí na schopnosti přesně ohodnotit podobnost jednotlivých položek a správné analýze návštěvníkova profilu. Nevýhoda tohoto přístupu spočívá v doporučování objektů, které jsou si podobné a tím pádem nerozšíří uživatelovy obzory.



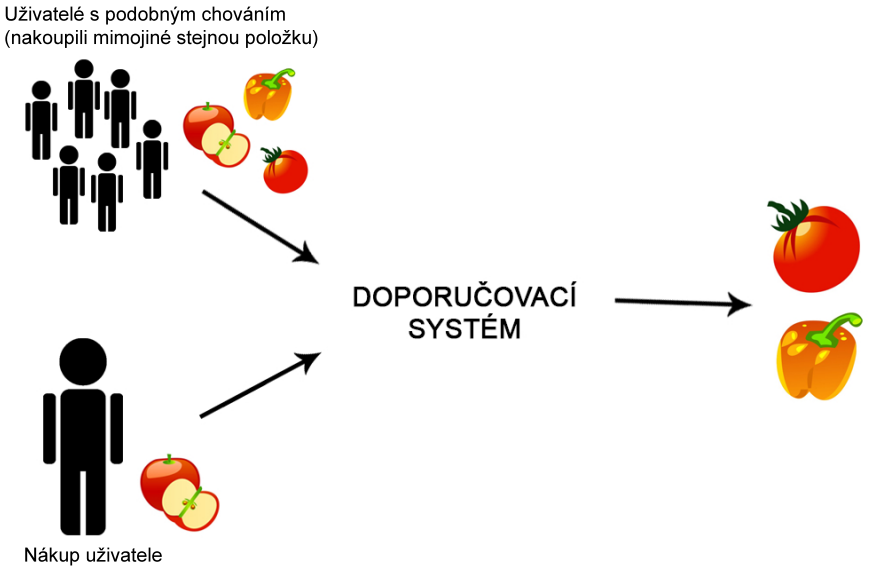
Obrázek 1.6: Doporučování na základě obsahu

1.5.2 Kolaborativní doporučování

Kolaborativní doporučování je založeno na myšlence, že nejkvalitnější doporučení získá člověk od přátel a známých, kteří mají podobné zájmy. Druhým

stavebním pilířem je teorie, která říká, že na položky, které měli uživatelé rádi v minulosti, budou mít stejný názor i v budoucnosti. Proces doporučování se skládá z identifikace skupiny lidí s podobným chováním a selekcí položek, které si tato skupina oblíbila.

Pro demonstraci kolaborativního doporučování použiji stejnou situaci jako v předchozím případě doporučování na základě obsahu. Tento příklad je doprovázen obrázkem 1.7. Uživatel má za sebou jeden nákup jablek. Doporučovací systém si vyžádá informace o ostatních uživatelích s podobným chováním (v tomto případě uživatelé, kteří ve svých objednávkách měli také jablka). Uživatelovi jsou doporučeny další produkty, které se v objednávkách vyskytovaly spolu s jablky nejčastěji – rajčata a papriky.



Obrázek 1.7: Kolaborativní doporučování

Znamená to, že algoritmy pro kolaborativní doporučování nejsou závislé na znalosti podrobností o položkách, které doporučují a rozhodují se výhradně na datech získaných při sledování chování uživatelů.

Kolaborativní doporučování se dále dělí na techniku doporučování dle podobnosti uživatelů (*user-based*) a techniku doporučování dle podobnosti položek (*item-based*).

Prvně jmenovaná metoda vypočítá pro každého uživatele v systému jeho podobnost s právě aktivním uživatelem. K výpočtu lze použít Pearsonův korelační koeficient nebo kosinovu míru podobnosti [15]. Za použití takto získané množiny nejpodobnějších uživatelů je pro každou položku vypočítaná hodnota udávající předpokládanou vhodnost pro doporučení. Položky s nejvyšším hod-

nocením jsou poté doporučeny uživateli. Tato metoda je nevhodná pro použití v systémech s velkým počtem uživatelů, protože je výpočetně značně náročná. Každá započítaná interakce uživatele (například ohodnocení produktu) způsobí nutnost přepočítat všechny hodnoty podobnosti.

Použití kolaborativního doporučování na základě podobnosti položek je v takovém případě vhodnější, neboť umožňuje předpočítat tabulky se vzájemnou podobností položek offline. Podobnost je v tomto případě myšlena ve smyslu interakcí uživatelů s položkami. V reálném čase systém pouze vybere z této tabulky nejpodobnější položky na základě posledních interakcí uživatele.

Kolaborativní doporučování patří dnes k používanějším metodám v doporučovacích systémech. Mezi nevýhody patří problém studeného startu, kdy po příchodu nového uživatele o něm systém nemá žádné informace a v první chvíli neví, které položky doporučovat. Stejný problém nastává při přidání nové položky do databáze, neboť taková položka nemá žádný vztah na uživatele a tím pádem nebude doporučovaná. Tento problém lze vyřešit kombinací doporučování na základě obsahu a kolaborativního doporučování. Tato metoda se nazývá hybridní doporučování.

1.5.3 Hybridní doporučování

Hybridní doporučování kombinuje různé doporučovací metody, aby minimalizovalo jejich individuální nedostatky. Jednou z variant je použití metody kolaborativního doporučování a doporučování na základě obsahu odděleně, navrhované výsledky z obou technik zkombinovat a poté zobrazit uživateli.

1.6 Motivace pro použití grafové databáze při implementaci doporučovacího systému

V kapitole 1.2.2 definující NoSQL databáze bylo řečeno, že síla NoSQL databází spočívá v jejich specializaci na konkrétní problémy, pro které jsou určeny. V následujících řádcích uvedu, proč jsem se při implementaci doporučovacího systému rozhodnul použít grafovou databázi.

Doporučovací systém má za cíl vytvořit předpoklad, jaký obsah je pro konkrétního uživatele relevantní. Se zvětšujícím se počtem různých informací začleněných do výpočtu se zvyšuje pravděpodobnost, že doporučený obsah bude kvalitní. V rámci doporučování produktů jsou tyto informace často reprezentovány jako vztahy. Může se jednat o interakce mezi uživatelem a internetovými stránkami, vztahy mezi produkty nebo jejich kategoriemi, ale například i mezi samotnými uživateli. Mezi snadno odvoditelné vztahy patří například „uživatel navštívil stránku produktu“, „uživatel zakoupil produkt“ nebo „produkt je zařazen do kategorie“. Grafové databáze jsou vhodné právě pro ukládání takových informací. Datové schéma grafu s vlastnostmi umožňuje ke vztahům přidávat atributy a díky tomu je lépe rozlišovat a filtrovat.

Sledováním vztahů se získá nový pohled na shromážděná data a bude možné najít nové postupy při doporučování.

Jednou z důležitých vlastností internetových stránek je rychlost jejich načítaní a reakcí na uživatelovy akce. Kratší odezvy zlepšují pocit z prohlížení stránek a naopak delší prodlevy můžou návštěvníka odradit od nákupu. Z tohoto důvodu musí doporučovací systém pracovat co nejrychleji. Vztahy mezi obsahem vytvářeným na straně internetového obchodu lze snadno dopředu předpočítávat a tyto hodnoty použít během doporučování. Vztahy mezi zákazníkem a webovými stránkami však vznikají v průběhu jeho návštěvy a tím pádem vzniká požadavek na jejich zpracování v reálném čase.

Publikace [16] srovnává výkonnost MySQL databáze s grafovou databází Neo4j na příkladu procházení sociálního grafu. Experiment ukazuje, že RDBMS MySQL je optimalizovaný pro dotazy s nejvýše jedním spojením a že se zvyšujícím se počtem příkazů JOIN v dotazu se rapidně zhoršuje výkonnost relačního databázového systému. Naproti tomu grafový databázový systém Neo4j zaznamenal s výrazně narůstajícím počtem záznamů jen minimální zvýšení času potřebného pro získání výsledků. Přestože byl experiment proveden za použití databázového systému Neo4j, výsledky experimentu lze zobecnit na většinu grafových databází, neboť hlavní důvod lepších výsledků oproti MySQL spočívá ve struktuře grafové databáze, která je optimalizovaná pro průchod grafem. Tabulka 1.1 znázorňuje data naměřená ve zmíněném experimentu.

Hloubka	MySQL - čas výpočtu (s)	Neo4j - čas výpočtu (s)	Výsledek dotazu
2	0,016	0,01	≈ 2500
3	30,267	0,168	≈ 125000
4	1 543,505	1,359	≈ 600000
5	výpočet nedokončen	2,132	≈ 800000

Tabulka 1.1: Výsledky experimentu na příkladu „najdi přátele mých přátel do hloubky X“. Databáze obsahovala 1 000 000 osob s průměrným počtem 50 přátel na osobu.[16]

Dalším důvodem pro zvolení grafové databáze je volnost jejich datového schématu. Ve většině grafových databázových systémů se data v uzlech nebo hranách ukládají do dvojic klíč-hodnota. Doporučovací systém tak může snadno shromažďovat různá data, která mohou být potřeba v budoucnosti. Tato vlastnost zároveň usnadní a zrychlí implementaci případných změn při ladění kvality doporučování.

Kapitola 2

Návrh a implementace

2.1 Popis internetového obchodu klienta

Internetový obchod nazuby.cz se specializuje na úzký sortiment zboží s jednoznačným zaměřením na ústní hygienu. Tomu odpovídá katalog produktů obsahující velké množství podobných výrobků, které se liší jen v několika parametrech. Produkty jsou rozdělené do kategorií, nicméně produkt může mít přiřazenou více než jednu kategorii. Kromě kategorií má každý produkt přiřazeny další atributy, které pomáhají při dalším filtrování produktů.

Kromě stránek s katalogem produktů obsahuje obchod sekci věnovanou článkům o ústní hygieně, odbornou poradnu a rejstřík pojmů. Tyto jednoznačně zaměřené stránky pomůžou lépe specifikovat uživatelský profil návštěvníka.

Webové rozhraní obchodu je implementované jako single-page aplikace[20] v jazyku Javascript a frameworku AngularJS. Veškerá komunikace se serverovou částí probíhá pomocí zabezpečeného HTTP protokolu jehož požadavky jsou přijímány REST rozhraním aplikačního serveru. Serverová aplikace používá programovací jazyk Java s frameworkem Spring. Veškerá data jsou uložena v RDBMS MySQL. Mapování Java objektů mezi aplikací a relační databází zajišťuje ORM nástroj Hibernate.

Tabulka 2.1 zobrazuje základní informace o používání webových stránek internetového obchodu. Z dat lze odvodit, že každý týden vznikne přibližně $2000 * 5,57 * 7 = 78000$ uzlů návštěv plus $100 * 7 = 700$ uzlů objednávek, které bude muset grafová databáze zpracovat. Dostupná data o používání webových stránek také poukazují na fakt, že větší část návštěvníků vytvoří objednávku až při opakované návštěvě, nicméně jen malá část využívá možnost vytvoření uživatelského účtu pro objednání.

[20] Typ webové aplikace, kdy se webová stránka načte pouze jednou. Veškeré další akce vyvolané uživatelem se provádí dynamicky. Obnovení stránky není potřeba.

Počet produktů	≈ 800
Přibližná návštěvnost	≈ 2000 uživatelů/den
∅ Počet navštívených stránek	$\approx 5,57$ stránek/návštěva
Četnost objednávek	≈ 100 objednávek/den
Objednávky založené novými návštěvníky	$45,37\%$
Objednávky založené vracejícími se návštěvníky	$54,63\%$
Objednávky založené registrovanými návštěvníky	$7,49\%$

Tabulka 2.1: Základní informace o používání stránek internetového obchodu nazuby.cz získané z nástroje Google Analytics

2.2 Výběr grafového databázového systému

Server Wikipedia[21] ke dni 13. března 2015 evidoval na anglické stránce věnované grafovým databázím 42 projektů vyvíjejících grafový databázový systém. Tyto projekty se nachází v různých fázích vývoje a výrazně se liší počty svých uživatelů. Při výběru grafového databázového systému pro doporučovací systém jsem zohlednil průzkum internetového portálu DB-Engines[22]. Tento server vyvinul metodu [17] pro porovnávání oblíbenosti databázových systémů a dle této metriky každý měsíc zveřejňuje aktuální žebříček. Dle výsledků průzkumu pro březen 2015 patří mezi nejpopulárnější grafové databáze Neo4j (skóre 27,62), Titan (skóre 3,92) a OrientDB (skóre 3,27) a proto byly zvoleny do užšího výběru kandidátů. Ostatní databázové systémy dosáhly skóre < 1 a z tohoto důvodu nebyly dále uvažovány.

2.2.1 Požadavky na grafový databázový systém

Na grafový databázový systém byly zvoleny následující požadavky:

- Implementovaná vlastnost *index-free adjacency*
- Programovací API v jazyce Java (případně podpora JAVA API frameworku Tinkerpop)
- Podpora dotazování pomocí průchodů grafem (Cypher nebo Gremlin)
- Licence umožňující nasazení do produkčního prostředí zdarma nebo za minimální cenu
- Kvalitní dokumentace
- Distribuovatelnost

[21] http://www.wikipedia.org/
[22] http://db-engines.com/

2.2.2 Neo4j

Počátek vývoje grafového databázového systému Neo4j se datuje do roku 2000, ale jeho první oficiální verze byla vydána až v roce 2010. V současné době jsou k dispozici dvě varianty aplikace. Community edice je bezplatná, nicméně opravňuje použít databázi Neo4j jen pro osobní potřeby. Ke komerčnímu užití je určena rozšířená verze Neo4j, která obsahuje pokročilé funkce[23], ale je zpoplatněna.[18]

Neo4j používá jako datový model orientovaný graf s vlastnostmi[24]. Od verze Neo4j 2.0 je model doplněn o štítky uzlů. Databáze Neo4j používá vlastní nativní grafové úložistě dat se specifickou architekturou a strukturou souborů. Nad úloženými daty na disku jsou implementované dvě úrovně cacheování, které zrychlují přístup k datům.

Pro dotazování nad daty je souběžně vyvíjen jazyk Cypher zmíněný v kapitole 1.4.2.1. Dále lze použít REST API, Java Traversal API, ale i Gremlin, neboť Neo4j implementuje framework Blueprints.

Ve verzi Neo4j 2.0 a novějších je umožněno definovat schémata a pravidla pro strukturu uzlů v databázi. Díky tomu je možné aplikovat omezení na jedinečnost vlastnosti uzlu a indexovat vlastnosti uzlů, což vede k rychlejšímu provádění dotazů.[19]

Každá operace, která přistupuje k datům v grafu, indexům nebo schématu, musí být provedena v transakci [20]. Neo4j implementuje vlastnosti ACID pro transakce za pomocí zamykání uzlů nebo hran. Aktivní transakce uchovává všechny změny v operační paměti. V případě úspěšného dokončení transakce jsou změny uloženy na disk.

Dle hodnocení DB-Engines je Neo4j zdaleka nejpopulárnější grafový databázový systém. Tomu odpovídá i podrobná dokumentace, množství článků a publikací, které jsou zaměřeny právě na tento databázový systém.

2.2.3 Titan

Za vývojem grafové databáze Titan stojí mimojiné skupina vývojářů z týmu vyvíjejícího framework Tinkerpop. Titan je grafová databáze, kterou její autoři označují jako vhodný systém pro práci s velkými grafy v řádech stovek miliard uzlů a hran. Aktuální verze Titan 0.5.4 je vyvíjena jako open source projekt v jazyce Java pod licencí Apache 2. Používání databáze je zdarma, nicméně komerční uživatelská podporua je zpoplatněna.

Titan podporuje rozhraní Blueprints, takže jako datový model je použit graf s vlastnostmi. Spolu s Blueprints Titan podporuje ostatní projekty Tinkerpop, mezi které patří Frames (objektově grafové mapování) a Gremlin (dotazovací framework pro průchod grafem).

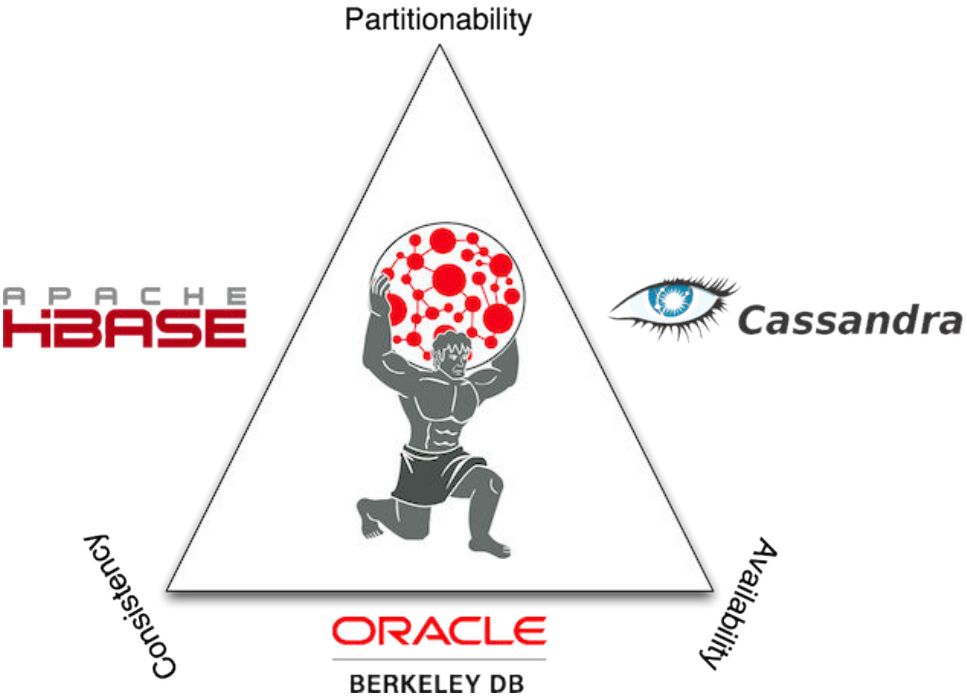
[23] Clustering, zálohování za běhu, pokročilé monitorování, výkonnější cacheování

[24] Neo4j nepodporuje neorientované hrany

Databázový systém Titan neimplementuje vlastní datové úložiště a místo toho podporuje zapojení jednoho ze tří existujících NoSQL úložišť: Apache Cassandra, Apache HBase nebo Oracle BerkeleyDB. Každý z těchto systémů má jiné možnosti škálování, dostupnosti a udržování konzistence dat a tím pádem ovlivňuje tyto vlastnosti vzhledem ke grafové databázi. Obrázek 2.1 ve stručnosti zobrazuje tyto tři databázové systémy v trojúhelníku CAP teorému.

Titan implementuje ACID transakce. Každá první operace s grafem, která není součástí transakce vytvoří novou transakci, která je ukončena explicitním zavoláním funkce *commit()* pro potvrzení transakce nebo *rollback()* pro odvolání změn provedených transakcí.

Dokumentace k databázi Titan je dostačující, nicméně chybí podrobnější tutoriály a návody pro začínající uživatele, které například Neo4j nabízí zdarma.



Obrázek 2.1: Porovnávní vlastností datových úložišť pro grafovou databázi Titan

2.2.4 OrientDB

První verze NoSQL databázového systému OrientDB vyšla v roce 2010. Databáze OrientDB se označuje jako multi modelový databázový systém, neboť je založena na ukládání dokumentů. Vzhledem k tomu, že vztahy mezi uchovávanými dokumenty splňují vlastnost *index-free adjency*, lze s ní pracovat

zároveň jako s grafovou databází. Spojením těchto dvou přístupů vznikl produkt, který není jednostraně zaměřený jako ostatní NoSQL databáze a má potenciál pro široké uplatnění. Stejně jako Neo4j a Titan je i OrientDB vyvíjeno v jazyce Java. Poslední verze OrientDB 2.0.5. byla vydána pod licencí Apache 2. Běžnou bezplatnou Community edici doplňuje placená Enterprise edice, která navíc nabízí nástroje pro monitorování a ladění výkonu nebo konfiguraci clusterů. Tyto nástroje jsou ve verzi 2.0.3 zdarma dostupné pro proces vývoje aplikace, zpoplatněné je nasazení do produkčního prostředí. Dále popisované vlastnosti jsou platné pro použití OrientDB jako grafovou databázi.

Stejně jako u předchozích databázových systémů je implementováno rozhraní Blueprints, takže datový model je orientovaný graf s vlastnostmi. Databáze OrientDB zavádí možnost udržovat pevné schéma databáze pomocí omezení a tříd (podobně jako u OOP), ale zároveň je schopná pracovat ve stavu bez definovaného schématu. Vlastností OrientDB je také podpora clusterů, které pomáhají shlukovat data stejného typu pro snažší distribuci mezi více úložišti.

Načítání dat z databáze je umožněno pomocí jazyka Gremlin a rozšířeného jazyka SQL. Operace s daty jsou prováděny v transakcích, které mají ACID vlastnosti. Toto je implementováno pomocí MVCC[25] mechanismu. Podporovány jsou dvě úrovně izolace (read commited a repeatable read). Databáze OrientDB zároveň umožňuje práci v režimu bez transakcí.

Jako jediný z testovaných databázových systémů podporuje zabezpečení pomocí uživatelských účtů a rolí, které řídí přístup k databázi.

Dokumentace databázového systému je kvalitní a doplněna o bezplatný výukový kurz, který začínající uživatele seznámí s důležitými vlastnostmi OrientDB.

2.2.5 Zvolení databáze

Všechny tři studované grafové databázové systémy splňují s menšími výhradami požadavky specifikované na začátku této kapitoly. Pro další vývoj systému byla zvolena databáze **OrientDB**. Klíčovým faktorem při rozhodování byla licence, s jíž je projekt vydáván, která opravňuje užívat databázi pro komerční účely bez poplatků. Pro vývoj a testování aplikace bude použita Enterprise edice databáze OrientDB, která je pro tyto účely bezplatná a bude nápomocná při ladění výkonu. Do produkčního prostředí bude nasazena verze Community.

[25] Multiversion concurrency control - mechanismus zajišťující konzistenci databáze

2.3 Specifické vlastnosti zvolené databáze

V této kapitole budou blíže popsány vlastnosti, které jsou specifické nebo důležité pro práci s databází OrientDB a odlišují ji od ostatních grafových databází.

2.3.1 Záznam

Záznam je nejmenší jednotka, kterou lze uložit nebo načíst z databáze. Každý záznam je jednoznačně identifikován pomocí RID (Record ID), který má formát: `#<cluster-id>:<cluster-position>`.

RID jednoznačně určujte fyzickou pozici záznamu v databázi. Každý záznam si uchovává číslo verze, které je zvyšováno při jeho aktualizaci. Tato hodnota poté slouží při vyhodnocování správnosti transakcí.

2.3.2 Třída

Třída v OrientDB definuje typ záznamu. Koncept tříd je zde podobný jako u OOP. Třídy slouží jako prostředek pro definici datového modelu, neboť jim lze přiřazovat atributy, na které se aplikují omezení a indexy. Databáze podporuje vlastnost OOP dědičnost tříd. Každé nově vytvořené třídě v databázi je automaticky přiřazen cluster.

2.3.3 Cluster

Cluster je úložiště jednotlivých záznamů databáze. Každá třída má přiřazen minimálně jeden cluster, do kterého jsou ukládány její záznamy. Definováním více clusterů pro jednu třídu lze rozdělit její záznamy mezi více souborů a tím zvýšit výkonnost databáze (zrychlení dotazů, paralelizace dotazů).

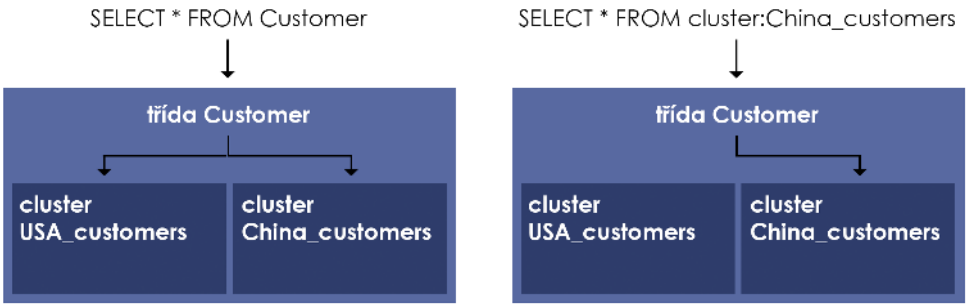
Oficiální dokumentace OrientDB uvádí příklad použití více clusterů na třídě *Customer*. Pro tuto třídu byly vytvořeny dva clustery *USA_customers* a *China_customers*. Aplikace je tak schopná fyzicky oddělovat nové záznamy sdílející stejnou třídu a individuálně v nich vyhledávat (obrázek 2.3.3). Tento přístup umožňuje optimalizovat rychlost dotazování do databáze.[26]

2.3.4 Schéma

Pomocí hierarchie tříd s proměnnými lze vytvořit datové schéma. Proměnné můžou nabývat hodnot některého z podporovaných datových typů[27]. Stejně jako v relačních DBMS lze pro proměnné definovat omezení. Implementovaná jsou následující omezení: minimální a maximální hodnota proměnné, povinně

[26] Příklad byl převzat z oficiální dokumentace OrientDB:
http://orientdb.com/docs/last/orientdb.wiki/Tutorial-Clusters.html

[27] Podporované datové typy jsou popsány v oficiální dokumentaci OrientDB http://orientdb.com/docs/last/orientdb.wiki/Types.html



Obrázek 2.2: Ukázka provedení dotazu nad celou třídou *Customer* a dotaz pouze and clusterem *China_customers*.

specifikovaná proměnná, proměnná pouze pro čtení, nenulová (*null*) hodnota proměnné, unikátnost proměnné a proměnná odpovídající regulárnímu výrazu.

2.3.5 Datová úložiště

Pro práci s databází OrientDB je možné zvolit ze tří podporovaných datových úložišť. Nejjednoduším z nich je operační pamět. Po startu serveru je vždy vytvořena nová databáze dle konfigurace. Veškeré operace s databází jsou dočasné a jejich efekt trvá pouze po dobu běhu databázového serveru. Po ukončení serverového procesu jsou všechna data nenávratně ztracena.

Druhým dostupným datovým úložištěm jsou soubory uložené na disku. Toto úložiště je označované jako *plocal*[28]. Databázová data jsou uchovávaná pomocí definované struktury[29] souborů v adresáři na disku.

Poslední metoda připojení k databázi je označovaná *remote* a umožňuje vzdálený přístup k databázi. Může se jednat o samostatnou databázi běžící na stejném serveru nebo vzdálenou databázi v lokální síti nebo internetu.

2.3.6 Indexování

Pokud je v databázi vytvořena třída s alespoň jednou proměnnou, lze nad ní vytvořit index pro rychlejší vyhledávání jejích záznamů v databázi. OrientDB implementuje 3 typy indexů:

- SB-Tree indexy (implicitní) — Dělí se na unikátní (nedovoluje vložení duplicitní hodnoty), neunikátní (umožňuje duplicitní hodnoty), fulltex-

[28] Zkratka pro anglický výraz „paginated local storage“ — lokální úložiště rozdělené na stránky

[29] Dokumentace struktury datových souborů OrientDB — http://orientdb.com/docs/last/Paginated-Local-Storage.html

tové (indexuje jednotlivá slova hodnoty) a slovníkové (neumožňuje duplicitní hodnoty, při pokusu o vložení duplikátu odstraní starou hodnotu). Jsou vyváženou variantou pro zápis i čtení.

- Hash index — Dělí se stejně jako SB-Tree indexy na unikátní, neunikátní, fulltextové a slovníkové. Umožňují rychlejší načítání konkrétních položek než SB-Tree indexy, ale není možné je použít při načítání rozsahů záznamů.

- Apache Lucene[30] — Omezené použití na fulltextové vyhledávání a hledání dle zeměpisných souřadnic.

Kromě výše uvedených indexů lze použít jiný externí indexovací nástroj.

2.3.7 Rozšířený jazyk SQL

Vzhledem k popularitě a rozšířenosti jazyka SQL je v OrientDB implementovaná jeho rozšířená varianta. Syntaxe vychází z SQL-92, ale doplňuje jazyk o možnosti využití tříd, clusterů nebo vztahů mezi záznamy. Rozšířený jazyk SQL lze použít z Java API, konzole nebo webového rozhraní databáze.

2.3.8 Nástroje pro vývoj a ladění výkonu

S verzí Community uživatel získá kromě databázového systému také webové rozhraní s možnostmi grafické vizualizace grafu, procházení exekučních plánů dotazů nebo vytváření schématu databáze. Dalšími dostupnými nástroji jsou konzole pro ovládání databáze a konzole pro dotazování pomocí jazyka Gremlin.

Verze Enterprise, která je pro vývojovou fázi dostupná zdarma [21], navíc obsahuje nástroj pro ladění dotazů, monitorování stavu databáze, správu clusterů a konfiguraci oznámení o událostech.

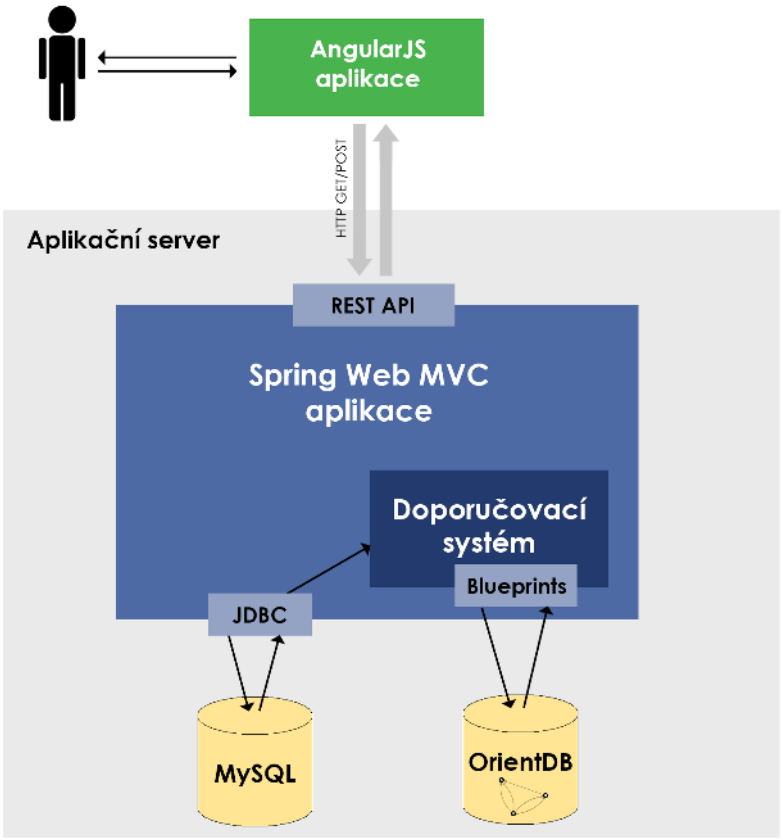
2.4 Architektura systému

Vzhledem k tomu, že vyvíjený doporučovací systém bude součástí již existující aplikace internetového obchodu, architektura systému musela být přizpůsobena stávajícímu řešení.

Obrázek 2.3 zobrazuje architekturu systému doplněnou o komponenty doporučovacího systému. Na schématu je zachycena komunikace mezi stávající aplikací a novými komponentami.

OrientDB — zvolený grafový databázový systém, který je odpovědný za uchovávání a poskytování informací potřebných pro proces doporučování. Databáze může běžet jako samostatná služba nebo ve stejném procesu s aplikací.

[30] https://lucene.apache.org/core/



Obrázek 2.3: Architektura internetového obchodu — komponenty a komunikace

Tuto vlastnost má databázový systém OrientDB, ale i další grafové databázové systémy. Grafové úložiště komunikuje výhradně s modulem doporučovacího systému za pomocí rozhraní Blueprints. Díky tomu je možné v případě nutnosti vyměnit databázový systém za jiný, který také implementuje rozhraní Blueprints.

MySQL — stávající relační databázový systém, na kterém je založen internetový obchod. Uchovává veškerá data pro potřeby internetového obchodu a jeho administraci. Komunikuje s aplikací a doporučovacím modulem za pomocí rozhraní JDBC. Doporučovací systém z relační databáze importuje data, která používá pro doporučování.

AngularJS aplikace — klientská webová aplikace, která zpracovává interakce uživatelů. Komunikace se serverovou aplikací probíhá za pomocí asynchronních HTTP požadavků GET a POST. Tyto požadavky jsou zasílány pomocí AngularJS služby *$http*[31]. Obsah požadavků je ve standardizovaném formátu JSON [22].

REST API — rozhraní pro komunikaci klientské AngularJS aplikace se serverovou částí. Pro potřeby doporučovacího systému bylo rozšířeno o nové zdroje.

Doporučovací systém — modul implementující logiku doporučovacího systému. Architekturu samotného modulu popisuje obrázek 2.4. Grafový databázový systém komunikuje výhradně se skupinou tříd, kterou jsem pojmenoval *Repository* (třídy s názvy **Repo.java*). Tyto třídy dědí obecné metody z naprogramované knihovny pro ukládání a načítání dat z grafové databáze. Zároveň implementují nové metody pro konkrétní použití.

Skupina tříd *Repository* očekává, že vstupující objekty jsou instance doménových tříd. Doménové třídy musí být doplněny o platné anotace definované v naprogramované knihovně pro ukládání a načítání dat z grafové databáze.

Plánovač je jednoduchá služba, která rozvrhuje vykonávání pravidelných operací. Mezi tyto operace patří například synchronizace dat s relační databází, odstraňování neplatných hodnot z grafové databáze a podobně.

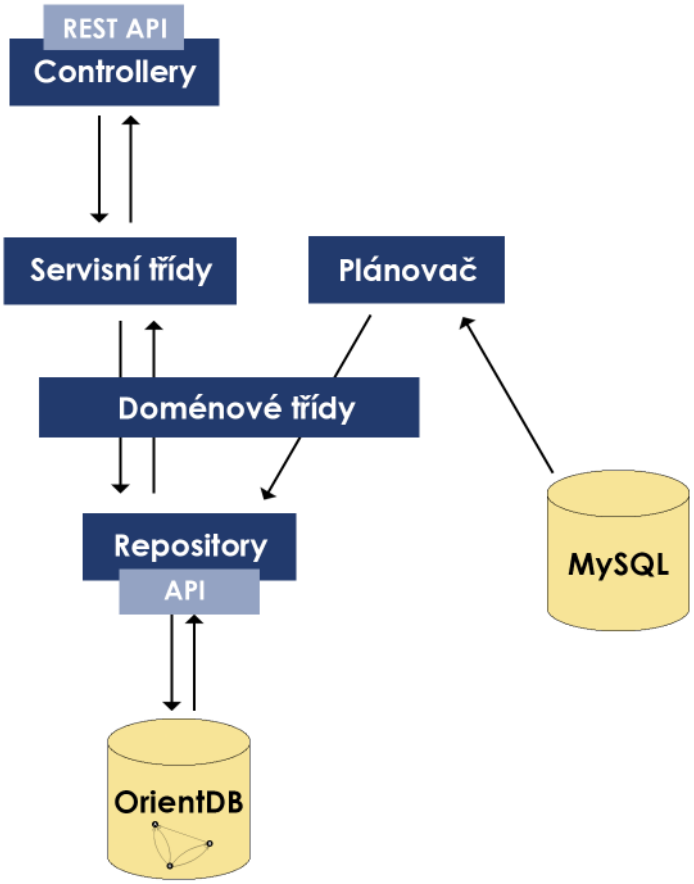
Komunikace klientů s doporučovacím systémem probíhá za pomocí HTTP rozhraní REST, které je implementováno pomocí skupiny tříd *Controllery*. Tyto třídy získávají a reagují na požadavky přicházející z AngularJS aplikace.

2.5 Datový model

Datový model byl navržen tak, aby umožňoval vytvořit doporučovací strategie na základě kolaborativního i obsahového přístupu. Grafová databáze slouží jako podpora pro doporučovací systém, takže obsahuje pouze uzly, vztahy a atributy potřebné pro doporučování.

Datový model aplikace se skládá ze tříd reprezentujících uzly (package *domain.nodes*) a tříd reprezentujících hrany (package *domain.relationships*)

[31] https://docs.angularjs.org/api/ng/service/$http



Obrázek 2.4: Architektura doporučovacího modulu

v grafu. Jedná se o POJO[32] třídy doplněné o anotace z implementované knihovny pro ukládání a načítání dat z databáze. Obrázek v příloze A.2 zobrazuje strukturu datového modelu pro doporučovací systém. Každý uzel a hrana má svůj interní unikátní identifikátor (atribut *id*), který spravuje databázový systém a používá ho ke splnění vlastnosti index-free adjency. Všechny entity byly doplněny o vlastní atribut s unikátní identifikátorem, pod kterým jsou uloženy v relační databázi internetového obchodu. Tento atribut slouží pro načítání konkrétní entity z grafové databáze.

[32]Objekt v jazyce Java bez dalších omezení spojených s dalšími frameworky, konvencemi nebo knihovnami

GraphUser — Uzel reprezentující jednotlivé návštěvníky internetového obchodu. Informace získané o návštěvníkovi jsou uloženy do atributů a hran. Tato data umožňují vytvořit uživatelský profil pro konkrétního návštěvníka. Uživatelé jsou rozlišováni jednoznačným identifikátorem (atribut *userId*). Identifikátor je textový řetězec ve formátu:

```
{id}.{čas první návštěvy}.{čas poslední návštěvy}
```

První část *id* je náhodně vygenerovaný textový řetězec odpovídající regulárnímu výrazu `[a-zA-Z0-9]{10}`. Čas první a poslední návštěvy je reprezentován unixovým časem[33]. Díky tomuto formátu je systém schopný zjistit délku uživatelovy návštěvy na webové stránce. Formát je inspirován identifikátorem používaným společností Google pro nástroj Google Analytics [23].

- Pro entitu *GraphUser* jsou sledovány následující vztahy:
  - *viewed_page*, *viewed_product*, *viewed_category* — Vztahy reprezentující pohyb uživatele na webových stránkách internetového obchodu. Pro každý vztah je uložena informace, kdy byla návštěva uskutečněna (atribut *visitedTime* typu `DateTime`), zda se jednalo o vstupní stránku[34] do internetového obchodu (atribut *initialVisit* typu `boolean`) a kolik sekund na dané stránce strávil jejím prohlížením (atribut *visitLength* typu `long`). Tyto hrany jsou implementovány jako multihrany, neboť pro systém je důležitá i informace o opakovaných návštěvách stejných stránek.
  - *created_order* — Vztah spojující uživatele s objednávkou, kterou vytvořil.

GraphWebPage — Uzly reprezentující webové stránky internetového obchodu, které uživatelé navštívili.

GraphOrder — Uzly reprezentující objednávky vytvořené v internetovém obchodě. Kromě unikátního identifikátoru objednávky *orderId* je zaznamenáváno datum a čas vytvoření objednávky (*orderCreated* typu `DateTime` a celková cena objednávky *totalItemsPriceWithVat* včetně dodatečných poplatků například za poštovné).

- Vztahy sledované pro entitu *GraphOrder*:
  - *contain_item* — Vztah spojující produkty s objednávkami, ve kterých byly obsaženy.

[33] Unixový čas – počet sekund uplynulých od času (UTC) 00:00:00 dne 1. ledna 1970

[34] Vstupní stránkou se označuje stránka webu, která byla uživatelem navštívena poprvé. Vstupní stránka je často důležitou informací o důvodu návštěvy uživatele.

GraphProduct — Každý uzel reprezentuje produkt nabízený internetovým obchodem. Unikátním identifikátorem je číselná hodnota shodná s identifikátorem v produktové databázi obchodu. Všechny atributy uzlu jsou pravidelně synchronizovány s produktovou databází. V databázi se uchovávají všechny produkty včetně neaktivních, aby se neztratila informace o jejich vztazích. Atribut *inSale* typu `boolean` uchovává informaci, zda je produkt aktivní a tím pádem může být doporučován.

- Vztahy sledované pro entitu *GraphProduct*:
  - *belong_to_category* — Vztah přiřazující produkt do kategorie. Atribut *mainCategory* typu `boolean` označuje přiřazení k primární kategorii produktu.
  - *brand_of*, *has_advisor_category*, *is_manufactured*, *is_product_type* — Vztahy, které přiřazují produkt ke specifickým kategoriím (značka, určení produktu, výrobce, typ produktu). Tyto vztahy slouží k doporučování na základě obsahu, neboť jejich následováním je možné získat podobné produkty vzhledem k jednotlivým sekcím. Jeden produkt může mít přiřazeno více vztahů stejného typu.

GraphCategory — Jednotlivé kategorie, do kterých jsou rozřazeny produkty. Produkt má přiřazenou jednu primární kategorii, ale může být zahrnut ve více kategoriích.

- Vztahy sledované pro entitu *GraphCategory*:
  - *is_subcategory* — Kategorie tvoří hierarchickou strukturu. Podřízená kategorie má vztah *is_subcategory* na svoji nadřazenou kategorii.

GraphBrand — Informace o značkách produktů jsou uloženy v uzlech označených *GraphBrand*.

GraphManufacturer — Uzly se štítkem *GraphManufacturer* uchovávají informace o producentech zboží.

GraphProductType — Kategorie, které slouží pro interní členění produktů v administraci internetového obchodu. Vyjadřují určitou podobnost mezi produkty.

GraphAdvisorCategory — Uzly *GraphAdvisorCategory* reprezentují typ použití daného produktu. U většiny produktů se jedná o problémy nebo nemoci, které konkrétní produkt léčí.

2.6 Ukládání a načítání dat z grafové databáze

Aplikace je implementovaná v objektovém jazyce Java. Entity datového modelu jsou tudíž reprezentovány pomocí objektů inicializovaných na základě definicí tříd, které mají jinou strukturu než grafové úložiště. Z tohoto důvodu je nutné zajistit mechanismus pro transformaci a následnou persistenci dat mezi grafovou databází a objekty v aplikaci. Ve světě relačních databází tento problém řeší technika zvaná objektově relační mapování (ORM). Mezi známé implementace ORM patří framework Hibernate[35] nebo EclipseLink JPA[36], které za sebou mají mnoho let vývoje.

Vzhledem k tomu, že grafové databáze jsou mladou technologií, podobné nástroje pro synchronizaci objektů s databází teprve vznikají. Jeden z nadějných projektů se jmenuje Frames, který je součástí již zmíněného frameworku vyvíjeného skupinou Tinkerpop. Doménový model s použitím Frames je reprezentován množinou Java rozhraní. Pro správné fungování je vyžadováno dodržování konvencí názvů metod a používání připravených anotací nad definovanými metodami (`@Property`, `@Adjacency` a `@Incidence`). Použití Frames není podmíněno konkrétním databázovým systémem.

Druhým aktivním projektem řešícím objektově grafové mapování je Spring Data Neo4j[37], který se soustředí, jak napovídá název, na podporu databázového systému Neo4j. Princip Spring Data Neo4j je založen na použití tříd (POJO) reprezentujících datový model. Za pomocí anotací se k třídám, proměnným nebo metodám přidají informace, které slouží knihovně Spring Data Neo4j ke správné manipulaci s daty. V projektu je implementován dotazovací jazyk Cypher specifický pro databázový systém Neo4j.

Po důkladné analýze obou projektů bylo rozhodnuto, že ani jeden projekt nevyhovuje potřebám vyvíjeného doporučovacího systému. Základní myšlenku abstrakce kódu aplikace od kódu potřebného pro správu dat v grafové databázi oba projekty splňují, nicméně byly nalezeny zásadní nedostatky, kvůli kterým nebude možné tyto nástroje použít. Nástroj Frames od Tinkerpop využívá příliš odlišné principy od klasického OOP. Definice datového modelu je zajištěna neobvyklým způsobem pomocí rozhraní a příliš využívá přístupu „convention over configuration“[38]. Tyto vlastnosti by mohly zpomalovat vývoj i následnou správu projektu ostatními členy vývojářského týmu, u kterých se nepředpokládá, že jsou s Frames důkladně seznámeni. Vzhledem k obecnosti Frames nejsou podporovány specifické vlastnosti databázových systémů. Spring Data Neo4j použitím tříd POJO pro definici datového modelu zachovává obvyklý princip OOP. Dostupné anotace jsou srozumitelně pojmenované a dobře po-

[35] http://hibernate.org/orm/

[36] http://eclipse.org/eclipselink/jpa.php

[37] Projekt je pokračováním původního projektu Spring Data Graph, jehož vývoj je pozastaven

[38] Přístup, kde převládají běžné zvyky nad konfigurací. Konfigurace produktu je přednastavena dle běžných potřeb použití a nemusí se přesně specifikovat

psané v dokumentaci projektu. Zásadní nevýhodou projektu je jeho závislost na databázi Neo4j, která byla vyhodnocena jako nevyhovující pro použití při vývoji doporučovacího systému.

Na základě výše popsaných důvodů byla vyvinuta knihovna pro databázový systém OrientDB, která zajišťuje persistenci dat. Databázový systém OrientDB nabízí programovací rozhraní (API), které je dostupné v jazyce Java a bylo použito pro implementaci knihovny. Inspirace pro případy použití byly převzaty z projektu Spring Data Neo4j. Vlastní implementace řešení persistence dat umožňuje využít specifických vlastností, které OrientDB nabízí. Patří mezi ně třídy, RID (identifikátor záznamu v databázi), indexy nebo clustery.

Základní myšlenkou implementované knihovny je abstrahování logiky práce s daty databáze od logiky doporučovacího systému. Datový model složený z uzlů a hran je proto reprezentován POJO třídami, které jsou doplněny implementovanými anotacemi. Implementovány byly následující anotace.

2.6.1 Dostupné anotace pro definici uzlů

- **@Node** — Anotace pro třídu. Definuje třídu jako typ uzlu v grafové databázi.
  - *name* (`String`) — Povinný parametr určující název třídy uzlu, pod kterým bude instance uložena do databáze.
- **@NodeProperty** — Anotace pro proměnnou třídy. Definuje proměnnou třídy jako atribut uzlu. Datový typ atributu je zděděn z datového typu proměnné.
- **@Unique** — Anotace pro proměnnou třídy. Definuje proměnnou, nad kterou se vytvoří unikátní index. Může být použita pouze pro jednu proměnnou ve třídě. Povinná, pokud je uzel součástí hrany.
- **@Indexed** — Anotace pro proměnnou třídy. Definuje proměnnou, nad kterou se vytvoří neunikátní SB-Tree index v databázi.

2.6.2 Dostupné anotace pro definici hran

- **@Relationship** — Anotace definující třídu jako typ hrany v grafové databázi.
  - *type* (`String`) — Povinný parametr určující název hrany, pod kterým bude instance třídy uložena do databáze.
  - *unique* (`boolean`) — Nepovinný parametr. Určuje, zda hrana tohoto typu může být vícenásobná. Původní hodnota nastavena na `false`.

- **@RelationshipProperty** — Anotace pro proměnnou třídy. Definuje proměnnou třídy jako atribut hrany. Datový typ atributu je zděděn z datového typu proměnné.
- **@NodeFrom** — Anotace pro proměnnou třídy. Definuje uzel, ze kterého vychází hrana.
- **@NodeTo** — Anotace pro proměnnou třídy. Definuje uzel, do kterého hrana vstupuje.

Příklad použití anotací implementovaných knihovnou je představen na třídě `GraphUser`, která reprezentuje uzel a třídě `PageViewRelationship`, která představuje hranu datového modelu. Obě ukázky zdrojového kódu lze nalézt v příhách A.1 a A.2.

2.6.3 Implementované operace

Základní metody potřebné pro ukládání a získávání dat z databáze zajišťuje třída `GenericGraphRepo`, která je implementovaná jako abstraktní generická třída. Pouhým rozšířením této třídy s definováním doménového typu a datového typu jeho identifikátoru získá uživatel následující CRUD operace pro práci se zvoleným doménovým typem.

- *T save(T entity)* — Instance třídy, která je anotovaná na úrovni třídy anotací (`@Node`), je uložena nebo upravena v databázi. Pokud v databázi již existuje uzel se shodným identifikátorem (proměnná třídy anotovaná `@Unique`), je tento uzel aktualizován hodnotami ukládaného objektu. Návratovou hodnotou je uložená instance.
- *Iterable<T> save(Iterable<T> entities)* — Hromadné uložení kolekce uzlů do databáze (objektů anotovaných na úrovni třídy anotací `@Node`). Již existující uzly jsou upraveny. Implementace metody používá optimalizaci pro hromadné vkládání dat, kterou poskytuje API databáze OrientDB [24]. Porovnání výkonnosti této optimalizace vůči iterování kolekce a postupnému ukládání shrnuje tabulka 2.2[39].
- *void saveRelationship(Object entity)* — Uloží do databáze libovolnou hranu (třídu s anotací `@Relationship`). Pokud je hrana definovaná jako unikátní a odpovídající hrana již v databázi existuje, tato hrana se pouze aktualizuje.
- *T get(ID id)* — Získá z databáze instanci na základě identifikátoru *id* z parametru a vrátí ji jako návratovou hodnotu. V případě, že nebyl nalezen uzel s odpovídajícím identifikátorem, pro návratovou hodnotu je použito *null*.

[39] Experiment byl spouštěn se základní konfigurací bez dalších optimalizací.

- *void delete(ID id)* — Odstraní z databáze uzel s identifikátorem *id* z parametru. V případě, že uzel s identifikátorem *id* nebyl nalezen, neprovede žádnou akci.
- *void delete(Iterable<ID> ids)* — Hromadné odstranění uzlů z databáze dle kolekce identifikátorů v parametru.
- *List<ID> listVertexIds()* — Získá seznam všech identifikátorů uzlů uložených v databázi od dané třídy. V případě, že žádný uzel nebyl nalezen, návratová hodnota je *null*.
- *long count()* — Vrátí číslo reprezentující počet uzlů v databázi dané třídy.

Všechny operace jsou implementované jako transakce. Je pouze na uživateli knihovny, zda se rozhodne implementovat další metody, které mu v implementaci třídy `GenericGraphRepo` chybí. Příklad rozšířené třídy `GraphUserRepo` lze najít mezi přílohami (příloha A.3).

Při implementaci knihovny byla použita reflexe, schopnost programovacího jazyka získat za běhu programu informace o typu a vlastnostech objektu, se kterým pracuje [25]. Programovací jazyk Java tuto vlastnost podporuje.

Počet uzlů	Čas s optimalizací (*ms*)	Čas při iterování kolekce (*ms*)	Zrychlení
10000	20153204, 555	23712419, 446	17, 66%
5000	11564426, 298	13420411, 626	16, 04%
1000	3522869, 749	4090110, 868	16, 10%
100	271428, 260	337477, 724	24, 33%

Tabulka 2.2: Porovnání hromadného vkládání dat s optimalizací a bez optimalizace

2.6.4 Konfigurace knihovny

Pro použití knihovny je nutné uložit soubor orientdb-mapper.jar mezi závislé knihovny projektu. Druhým krokem je konfigurace třídy `OrientDbSchemaChecker`, kde je nutné do proměnné *scannedPackage* uložit název Java balíčku s třídami reprezentujícími datový model (uzly a hrany). Posledním krokem je konfigurace třídy `OrientDbManager`, kde je nutné nastavit připojení k databázi.

2.7 Sběr dat pro doporučování

Data, která jsou použita pro doporučování produktů lze rozdělit do dvou kategorií:

- Data z relační databáze internetového obchodu
- Data získaná na základě sledování chování návštěvníků

2.7.1 Získávání dat z databáze internetového obchodu

Databáze internetového obchodu je spolehlivý zdroj dat o produktech, kategoriích nebo objednávkách. Jedná se o data z tisíců řádků relační databáze. Tyto informace se často mění (například zařazení nových produktů do katalogu obchodu nebo vytváření nových kategorií), a proto je nutné zajistit pravidelnou aktualizaci těchto dat v grafové databázi doporučovacího systému. Pro administrátora internetového obchodu by tento proces měl znamenat žádnou nebo minimální administraci.

2.7.1.1 Inicializační import dat

Před prvním spuštěním aplikace je nutné importovat do grafové databáze aktuální data z relační databáze. Pro tento případ jsem se rozhodl použít komponentu OrientDB-ETL[40], která je součástí databázového systému OrientDB. Tento nástroj slouží k importování dat do databáze OrientDB z několika zdrojů.

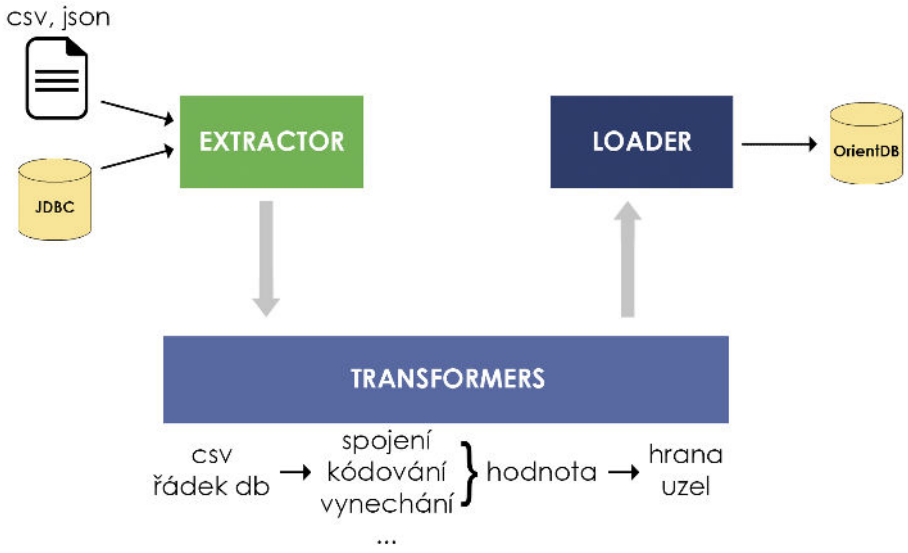
Proces ETL obecně slouží pro přenos dat mezi dvěma datovými úložišti s možnou transformací dat během přenosu. Proces ETL zahrnuje sled tří operací:[26]

- Extrakce — První část ETL procesu, která definuje množinu a strukturu načítaných dat. Zdrojem dat můžou být například dokumenty, soubory nebo databáze. Data jsou načtena a odeslána do další fáze.
- Transformace — Druhá část ETL procesu, která definuje pravidla a funkce pro úpravu vstupních dat z původního formátu do struktury požadované cílovým úložištěm. Příkladem transformací může být agregace, slučování buněk nebo kódovaní.
- Uložení — Poslední fází ETL procesu je uložení transformovaných dat do cílového datového úložiště.

Každý ETL proces je při použití nástroje OrientDB-ETL popsán konfiguračním souborem ve formátu JSON. Konfigurační soubor je složen z několika

[40]ETL je zkratkou anglických slov Extractor, Transformers, Loader

sekcí z nichž nejdůležitější jsou Extractor (popisuje fázi extrakce dat – definuje zdroj dat), Transformers (definuje fázi transformace – za pomocí implementovaných funkcí transformuje data) a Loader (fáze uložení – konfiguruje připojení k databázi OrientDB, do které mají být získaná data uložena). Sekce Transformers může obsahovat více funkcí, které se aplikují dle pořadí, v jakém jsou zapsány. Obrázek 2.5 popisuje ETL proces implementovaný databázovým systémem OrientDB.



Obrázek 2.5: Popis ETL procesu v databázi OrientDB

Pro potřeby doporučovacího systému jsem ETL proces použil pouze pro inicializační import dat do grafové databáze. Extractor ETL procesu jsem nakonfiguroval jako JDBC připojení do relační databáze[41]. Pomocí SQL dotazů jsem specifikoval data, která měla být přenesena. Sekce Transformers se lišila dle typu importovaných dat. Vazby (hrany) mezi entitami lze importovat souběžně s uzly, pouze byly upraveny SQL dotazy připojením odpovídajících tabulek. Pokud je entita součástí více než jedné vazby many-to-many (v mém případě například entita *Product*), bylo nutné importovat druhou a další vazbu M:N odděleně[42]. Sekce Loader byla nakonfigurovaná dle požadované cílové grafové databáze. Všechny ETL procesy byly spuštěny před prvním spuštěním doporučovacího systému.

[41]Byl použit externí JDBC driver MySQL Connector/J stažený z webu http://dev.mysql.com/downloads/connector/j/

[42]Spojením více spojovacích tabulek vzniknou ve výsledku duplicitní hodnoty, což vede k duplicitnímu vytvoření hran.

Použité konfigurace ETL procesů jsou dostupné na přiloženém CD. Příklad jedné z konfigurací (import uzlu *Category* včetně odpovídajících hran) byl zároveň vložen mezi přílohy této práce. Viz A.4.

2.7.1.2 Pravidelná aktualizace dat

Po dobu běhu aplikace je nutné průběžně aktualizovat data v grafové databázi. Z tohoto důvodu byla implementovaná služba `GraphDbSynchronizationService`, která zajišťuje pravidelný import aktualizovaných nebo nově vytvořených produktů, kategorií, značek, výrobců, produktových typů a objednávek do grafové databáze. Každý import obsahuje mnoho položek a proto bylo nutné načítání a ukládání dat do databáze optimalizovat.

Načítání kompletních objektů z relační databáze pomocí použitého ORM frameworku Hibernate je nevhodné, neboť k objektům je navázané množství kolekcí, jejichž načtení vyžaduje dodatečné dotazy do dalších tabulek. Tyto kolekce dat nejsou pro import do doporučovacího systému vždy potřeba a jejich načítání by zpomalovalo celý proces importu. Z tohoto důvodů jsou z relační databáze získávána pouze data potřebná pro vytvoření objektů doménového modelu pro grafovou databázi. Vytváření instancí DTO[43] objektů během načítání dat z databáze umožňuje Java Persistence API. Následuje ukázka zdrojového kódu, který tento mechanismus zajišťuje:

```
1  public List<GraphOrder> findGraphOrderSince(LocalDateTime sinceDateTime) {
2      CriteriaBuilder cb = getEntityManager().getCriteriaBuilder();
3      CriteriaQuery<GraphOrder> query = cb.createQuery(GraphOrder.class);
4      Root<Order> orderRoot = query.from(Order.class);
5      query.select(
6              cb.construct(
7                  GraphOrder.class,
8                  orderRoot.get("orderId"),
9                  orderRoot.get("insertTime"),
10                 orderRoot.get("itemsTotalPriceWithVat")
11         )
12     );
13     query.where(cb.equal(orderRoot.get("insertTime"), sincelDateTime));
14     TypedQuery<GraphOrder> typedQuery = getEntityManager().createQuery(query);
15     return typedQuery.getResultList();
16 }
```

Zdrojový kód 2.1: Načtení pouze potřebných dat z relační databáze do DTO objektu, který je uložen do grafové databáze.

[43] Typ objektu, který seskupuje data do požadované struktury (data můžou být z různých tabulek nebo transformována).

Na straně grafové databáze byla použita optimalizace pro hromadné vkládání dat, kterou implementuje naprogramovaná knihovna.

Služba byla zakomponovaná do existujícího plánovače internetového obchodu, který každých 24 hodin spouští import dat. Importují se pouze data vytvořená případně upravená za dobu od poslední synchronizace. Informace o vytvoření nebo editaci dat poskytuje internetový obchod ve sloupcích databáze určených pro audit.

2.7.2 Získávání dat o návštěvnících webových stránek

Na základě dat o chování návštěvníků na webových stránkách obchodu (tabulka 2.1) bylo rozhodnuto, že pro potřeby personifikovaného doporučování bude vhodné získávat informace o uživateli již od jeho prvního vstupu na webové stránky.

Informace o chování návštěvníků na webových stránkách nebyly dosud obchodem získávány. Tato data je nutná ukládat a zpracovávat online, protože jsou ihned používána pro specifikování uživatelského profilu návštěvníka. Z tohoto důvodu byla implementována metoda pro sběr takových dat.

Na straně klientské aplikace byla naprogramovaná nová AngularJS služba *RecommendService*, která zasílá asynchronní požadavky se získanými daty na server. Vzhledem k povaze single-page aplikace nelze data odesílat po načtení stránek jako je tomu obvyklé u běžných webových aplikací. Načtení stránky je simulováno AngularJS událostí *$viewContentLoaded*, jež je vyvolána po načtení všech závislostí spojených s aktuálním stavem stránky. Na vyvolání této události reaguje nově vytvořená AngularJS služba *RecommendService* a pomocí funkce *trackVisit()* zašle přes HTTP protokol požadavek typu GET na serverovou část.

HTTP požadavek obsahuje v parametrech data, která byla o uživateli zjištěna. Jedná se o následující informace:

- Identifikátor uživatele (pokud se nejedná o první návštěvu)
- URL, kterou uživatel navštívil
- Informace o stavu přihlášení uživatele
- Informace o nastaveném řazení produktů

Na straně serveru bylo implementováno REST rozhraní, které přijímá HTTP požadavky. Za zpracování a uložení dat do databáze zodpovídá controller `CollectDataController`. V případě, že požadavek neobsahoval identifikátor uživatele, třída `RecommenderCookieResolver` vygeneruje nový identifikátor a připojí ho do odpovědi zasílané serverem. Na straně klienta je tento řetězec uložen v prohlížeči do cookie. Controller informace o návštěvách přiřazuje k uživatelům v grafové databázi. Pokud existuje předchozí návštěva

stránky spojená se stejným uživatelem, která není starší než 24 hodin, aktualizuje se doba návštěvy na této stránce.

Ze statistik návštěvnosti webových stránek obchodu bylo zjištěno, že přibližně 11% návštěvníků opustí stránky ihned po zobrazení vstupní stránky. Tyto návštěvy nejsou pro doporučování relevantní. Z tohoto důvodu byla implementovaná pravidelně spouštěná úloha, která vyhledá uzly návštěvníků s jednou navštívenou stránkou a tyto uzly odstraní z databáze.

2.8 Doporučování produktů

Pomocí grafové databáze s dostatkem relevantních dat lze implementovat velké množství různých dotazů vhodných pro různé situace doporučování. Pro řešení tohoto problému byl zvolen návrhový vzor Strategy.

Návrhový vzor Strategy umožňuje definovat skupinu algoritmů (strategií), které lze za běhu aplikace zaměňovat. Jednotlivé strategie nejsou závislé na prostředí, ze kterého jsou volány.[27]

Pomocí návrhového vzoru Strategy byly implementovány jednotlivé metody doporučování produktů nezávisle na jejich dalším použití. Společné rozhraní `RecommendationStrategyInterface` definuje metodu *getProducts()*. Parametrem této metody je instance třídy `BasicRecommendationMetadata`, která obsahuje parametry pro doporučování. Návratovou hodnotou metody je seznam identifikátorů produktů seřazený dle preference pro doporučení.

Jednotlivé strategie doporučování se liší vstupními parametry, dle kterých se řídí průchody grafem. Tento problém jsem vyřešil za pomoci návrhového vzoru Abstract Factory.

Návrhový vzor Abstract Factory poskytuje obecné rozhraní pro inicializování skupin objektů. Pro vytváření těchto objektů není nutné znát jejich třídu. O vytvoření správného objektu se stará implementace rozhraní (konkrétní factory).[27].

Bylo implementováno rozhraní `MetadataFactoryInterface`, které definuje metodu *createMetadata()*. Pro každou strategii doporučování byla vytvořena třída konkrétní Factory implementující popsané rozhraní, která vygeneruje objekt s požadovanými parametry. Vytvářené objekty jsou instancemi tříd, jež jsou potomky třídy `BasicRecommendationMetadata`.

Analýzou struktury stránek internetového obchodu byla navržena následující tři místa, kde budou použity výsledky doporučovacího systému:

- Stránka s detailem produktu
- Dialogové okno po přidání položky do košíku
- Dialogové okno s nabídkou dokoupení produktů pro získání bezplatné dopravy

Každá z těchto situací vyžaduje jinou metodu doporučování a proto byly navrženy a implementovány tři různé strategie implementující obecné rozhraní `RecommendationStrategyInterface`.

Pro dotazování do databáze byl zvolen jazyk Gremlin. Dotazy jsou implementované pomocí frameworku Pipes[44] z Tinkerpop stacku. Každý krok dotazu v jazyku Gremlin odpovídá jednomu Pipe objektu. Složením několika Pipe objektů lze vytvořit požadované dotazy. Při implementaci byly použity již připravené Pipe objekty, ale zároveň musely být implementovány vlastní Pipe objekty pro specifické případy použití.

2.8.1 Detail produktu

Stránka detailu produktu obsahuje podrobný popis produktu, veškeré vlastnosti, parametry a fotografie. V případě, že uživatel navštíví stránku s detailem produktu, lze předpokládat, že o tento produkt má zájem. Nicméně může to také znamenat, že ještě není přesvědčen o jeho zakoupení, neboť v takovém případě by produkt do košíku mohl vložit přímo z výpisu kategorie. Pro takovou situaci je vhodné návštěvníkovi nabídnout produkty, které jsou podobné s právě prohlíženou položkou a je šance, že by lépe odpovídaly jeho potřebám.

Tento požadavek odpovídá metodě doporučování na základě obsahu, neboť cílem je nabídnout produkty s podobnými parametry a vlastnostmi. Zároveň je vhodné zohlednit získaný uživatelský profil, pomocí kterého lze výběr navrhovaných produktů lépe zaměřit.

Postup pro tuto metodu doporučování implementuje samostatná strategie ze třídy `ProductDetailRecommendationStrategy`. Tato strategie má za cíl nalézt 12 produktů podobných s právě prohlíženým produktem, u kterých je předpoklad, že by mohly uživatele zaujmout.

Prvním krokem je analýza uživatelského profilu, do které vstupují jako parametry identifikátor uživatele (dále jako *userId*) a identifikátor prohlíženého produktu (dále jako *productId*). Dotaz je definován jako následující průchod grafem:

1. Z grafu získám uzel uživatele (dále jako U) na základě *userId*.

2. Najdu všechny hrany vycházející z U označené štítkem *viewed_product*. Filtruji pouze hrany, které mají atribut $visitLength > 5$. Návštěvy stránky s detailem produktu kratší než 5 sekund nebudou považovány za relevantní. Množina uzlů, ve kterých tyto hrany končí, reprezentuje produkty navštívené uživatelem (množina P).

3. Následováním hran se štítkem *belong_to_category* vycházejících z uzlů množiny P získám kategorie již navštívených produktů (dále množina C).

[44] https://github.com/tinkerpop/pipes

4. Zpětným průchodem hran *belong_to_category* získám množinu produktů zařazených do těchto kategorií. Tento výběr omezím pouze na produkty s identifikátorem *productId*. Tím se zároveň zredukuje množina C i P na kategorie a produkty relevantní pouze s právě prohlíženým produktem.

5. Vrátím seznam produktů reprezentovaný množinou P

Poté se provede podobný průchod grafem. Pouze se zamění původní vztah *belong_to_category* za vztah *has_advisor_category*. Obě výsledné množiny produktů jsou sjednoceny. Získaná množina obsahuje produkty, o které se uživatel zajímal a jsou podobné právě prohlíženému produktu. Následně se vypočítá průměrná cena těchto produktů. Tato hodnota (*avgUserPrice*) je jedním z parametrů pro alogitmus hledání produktů, které budou uživateli doporučeny.

Dalším parametrem je metoda řazení produktů, kterou má uživatel aktuálně nastavenou. Nejhodnotnějším měřítkem při odhadování preferencí uživatele byla zvolena metoda řazení dle ceny. V případě řazení dle ceny sestupně lze předpokládat zvýšený zájem o dražší produkty. Naopak při zvoleném řazení od nejlevnějšího produktu je pravděpodobné, že pro zákazníka je důležitá nízká cena zboží.

Produkty, které budou uživateli doporučeny jsou vybrány na základě následujícího postupu:

1. Z grafu získám uzel právě prohlíženého produktu na základě *productId* (dále jako P).

2. Následováním incidujících hran *belong_to_category*, *has_advisor_category*, *is_product_type* najdu množinu parametrů produktu P.

3. Zpětným průchodem hran *belong_to_category*, *has_advisor_category* a *is_product_type* získám seznam produktů SP, které sdílí s produktem P alespoň jeden parametr (produkt, který sdílí dva parametry je v seznamu obsažen dvakrát).

4. Odstraním položky, které nejsou aktuálně v prodeji.

5. Ze seznamu SP vytvořím seřazenou množinu produktů MP, kde řadícím kriteriem je výskyt jednotlivých položek v seznamu.

6. Položky z MP se stejným počtem výskytů seřadím jedním z následujících způsobů:

   a) V případě, že má uživatel nastaveno řazení produktů v internetovém obchodu dle ceny vzestupně, z množiny MP jsou vybrány produkty, pro které platí výraz $totalPriceWithVat - avgUserPrice < 0$. Dle této hodnoty jsou seřazeny sestupně. Zbytek produktů je seřazen sestupně dle ceny na konec množiny MP.

b) V případě, že má uživatel nastaveno řazení produktů v internetovém obchodu dle ceny sestupně, z množiny *MP* jsou vybrány produkty, pro které platí výraz $totalPriceWithVat - avgUserPrice > 0$. Dle této hodnoty jsou seřazeny vzestupně. Zbytek produktů je seřazen sestupně dle ceny na konec množiny *MP*.

c) Pokud má uživatel nastaveno řazení dle jiného kritéria, množina *MP* je seřazena vzestupně dle výrazu $|totalPriceWithVat - avgUserPrice|$.

7. Vrátím seřazenou množinu produktů *MP*.

Na základě množiny identifikátorů produktů *MP* získám dotazem do relační databáze informace o produktech ve struktuře, kterou očekává klientská webová aplikace. Ve stejném dotazu jsou odfiltrovány produkty, které nejsou aktuálně skladem. Pokud má množina načtených produktů více než 12 prvků, přebytečné položky jsou odstraněny. Výsledná seřazená množina je následně odeslána v těle HTTP odpovědi ve formátu JSON.

2.8.2 Dialogové okno po přidání položky do košíku

V případě, že si návštěvník internetového obchodu přidá za pomocí tlačítka zboží do košíku (nezávisle na sekci obchodu), je mu zobrazeno dialogové okno s potvrzením jeho akce. Toto okno slouží jako zpětná vazba na provedenou akci a zároveň umožňuje uživateli přejít ihned do platebního procesu. Pro provozovatele obchodu je to také jedno z posledních míst, kde může nabídnout návštěvníkovi produkty před samotným provedením nákupu.

Produkty nabízené v dialogovém okně nemusí být podobné právě přidanému produktu do košíku (dále pouze *P*), neboť lze předpokládat, že návštěvník je již rozhodnut o jeho zakoupení. Pravděpodobnější je přikoupení příslušenství nebo jiného levnějšího produktu. V tomto případě je vhodné čerpat podklady pro doporučování z chování ostatních návštěvníků internetového obchodu. Tento postup je typem kolaborativního doporučování a je implementován pomocí strategie `AfterAddingToBasketRecommendationStrategy`.

Cílem této strategie je nalézt uživatele podobné aktuálnímu návštěvníkovi webových stránek. Na základě chování těchto uživatelů navrhnout 12 produktů, které mají šanci zaujmout aktuálního návštěvníka.

Prvním krokem je identifikace „podobných uživatelů“, jejichž chování bude sledováno. Podobní návštěvníci jsou v implementovaném systému rozpoznáváni na základě dvou parametrů:

- Návštěvnící, kteří již provedli objednávku, ve které byl obsažen produkt *P*.
- Návštěvnící, kteří na stránky internetového obchodu přišli se stejnou vstupní stránkou.

V prvním případě se provede následující průchod grafem:

1. V grafu najdu produkt P na základě *productId*.
2. Následováním hran *contain_item* vstupujících do uzlu P získám množinu objednávek (dále pouze O), ve kterých je produkt P obsažen.
3. Průchodem hran *contain_item* vycházejících z množiny objednávek O získám seznam produktů, které byly zakoupeny spolu s produktem P.
4. Seskupím seznam produktů získaný v předchozím kroku dle *productId*.
5. Vrátím množinu identifikátorů spolu s číslem reprezentujícím počet výskytů produktu v seznamu.

Následně se provede druhý průchod grafem dle kritéria stejné vstupní stránky:

1. V grafu najdu uzel aktivního uživatele na základě jeho *userId*.
2. Najdu nejnovější hranu *viewed_page* s atributem *enter_visit*=`true` a uzel webové stránky, do které vede (dále uzel *WS*).
3. Od uzlu *WS* následuji všechny vstupující hrany *viewed_page* s atributem *enter_visit* = `true`. Tím získám uzly uživatelů se stejnou vstupní stránkou.
4. Z uzlů uživatelů pokračuji průchodem hran *created_order* k uzlům objednávek a dále pak hranami *contain_item* k seznamu produktů v těchto objednávkách.
5. Seskupím seznam produktů ze 4. kroku dle *productId*.
6. Vrátím množinu identifikátorů s počtem výskytů produktu v seznamu.

Množiny získané z výše uvedených průchodů grafem jsou následně sjednoceny a seřazeny dle počtu výskytů produktu.

Na základě výsledné seřazené množiny identifikátorů jsou z relační databáze načtena data o produktech a transformována do DTO objektů, které očekává webová aplikace. Dotaz do relační databáze také zohledňuje podmínku na aktuální dostupnost produktu. Pokud je načteno více než 12 produktů, přebytečné produkty jsou odstraněny a konečná seřazená množina je odeslaná v těle serverové HTTP odpovědi ve formátu JSON.

2.8.3 Dialogové okno s nabídkou dokoupení produktů pro získání bezplatné dopravy

Jednou z metod pro zvýšení obratu internetového obchodu je přislíbení výhod po dosažení určité hodnoty objednávky. Zákazník získá motivaci k nákupu dalších produktů, které původně neměl v plánu kupovat. Předpokládám, že efektivitu této metody zvýším, pokud návštěvníkovi nabídnu produkty, které pro něj budou relevantní a zároveň se jejich cena blíží k překročení požadovaného limitu hodnoty objednávky. V internetovém obchodu nazuby.cz je tímto limitem 1000 Kč a po jeho překročení je zákazníkovi odpuštěn poplatek za dopravné a balné.

Tento typ doporučování je implementován strategií, kterou popisuje třída `FreeDeliveryRecommendationStrategy`. Nejdůležitějším parametrem této strategie je částka, která zbývá zákazníkovi do získání výhody (označena proměnnou *leftToFreeDelivery*). V druhé řadě je brán ohled na produkty, které má zákazník v košíku, ale také na výši marže, kterou obchodník produktům nastaví. Jedná se o metodu hybridního doporučování.

Byly navrženy průchody grafem jejichž výsledky jsou poté sjednoceny. Filtrování produktů dle ceny a jejich seřazení probíhá v následující fázi načítání produktů z relační databáze.

Průchod grafem je podobný kolaborativnímu filtrování dle objednávek ostatních zákazníků v kapitole 2.8.2. Jediným rozdílem je úprava prvního kroku, kde je produkt P zaměněn za množinu produktů, které má aktuálně zákazník v nákupním košíku.

Další fází je nalezení množiny produktů s atributem *basketOffer* nebo *inAction* nastaveným na hodnotu `true` a vrácení jejich unikátních identifikátorů.

Následně se provede dotaz do relační databáze, který provedé následující operace:

1. Načte produkty dle získaných identifikátorů
2. Filtruje získané položky jejichž aktuální cena je vyšší než hodnota proměnné *leftToFreeDelivery*.
3. Seřadí výběr dle aktuální ceny[45] a marže.
4. Odstraní přebytečné produkty.

Načtená data jsou transformována do DTO objektů určených pro klientskou webovou aplikaci a odeslaná v těle HTTP odpovědi ve formátu JSON.

[45] Aktuální cena se vyhodnocuje v okamžiku zobrazení produktu.

2.9 Dokumentace důležitých komponent

Hledání produktů pro doporučení probíhá asynchronně a tím pádem nezpomaluje načítání nové obrazovky. Každá doporučovací strategie má svou unikátní URL, která přijímá HTTP metodou GET parametry.

Na straně klienta komunikuci zajišťuje implementovaná AngularJS služba `RecommendService`, která kromě již zmíněné funkce *trackVisit()* definuje následující funkce:

- *getSimilarProductRecommendation()* — zasílá na server HTTP požadavek metodou GET s parametrem *productId*. Ze serveru získá odpověď s produkty určenými pro zobrazení na stránce detailu produktu.
- *getAfterAddingToBasketRecommendation()* — zasílá na server HTTP požadavek metodou GET s parametrem *productId*. Ze serveru získá odpověď s produkty určenými pro dialogové okno zobrazené po přidání produktu do košíku.
- *getTipsToFreeDeliveryRecommendation()* — zasílá na server HTTP požadavek metodou GET s parametrem *leftToFreeDelivery* a parametrem *basket* reprezentující položky košíku. Ze serveru získá odpověď obsahující produkty určené pro zobrazení v dialogovém okně s nabídkou dokoupení produktů.

Serverová část aplikace zpracovává HTTP požadavky za pomocí controllerů jejichž mapování má prefix `/rest/recommendation` a následuje:

Controller	URL
`ProductDetailRecommendationController`	`/product-detail/list.json`
`FreeDeliveryRecommendationController`	`/free-delivery/list.json`
`AfterAddingToBasketRecommendationController`	`/after-adding/list.json`

Tabulka 2.3: Mapování controllerů

Kapitola 3

Ladění výkonu a testování

Tato kapitola se věnuje metodám a postupům zajišťujícím stabilitu a výkonnost systému. Jedná se zejména o úkony spojené s pravidelným testováním jednotlivých komponent systému a sledováním výkonnosti grafové databáze OrientDB při provádění operací. Na základě získaných informací jsem se snažil upravit implementovaný systém a konfiguraci databáze a tím získat rychlejší odezvy a spolehlivější fungování.

3.1 Ladění a testování databáze OrientDB

3.1.1 Ladění dotazů do databáze

Pro fázi vývoje je vývojářům bezplatně dostupná verze OrientDB Enterprise, která obsahuje kromě dalších i nástroje pro ladění výkonu. Následující data byla získána pomocí nástroje Enterprise Workbench.

První fází bylo odhalení nejfrekventovanějších dotazů do databáze. Monitorováním souborů s logováním frekvence dotazů bylo zjištěno, že nejčastěji[46] je volán dotaz pro nalezení uživatele dle jeho identifikátoru. Dokumentace databáze OrientDB uvádí několik doporučení pro zvýšení výkonu dotazů [28]. V následujícím textu provedu experimenty, které prokáží reálný dopad na výkonnost databáze při použití těchto doporučení.

Experiment bude uskutečněn nad testovací databází se 100 000 uzly uživatelů, 10 000 uzly objednávek a 10 000 uzly produktů. V prvním testovacím případě nebylo vytvořeno schéma databáze. Všechny uzly jsou jednoho typu (V). Byl proveden následující dotaz:

```
select from V where userId = ‘5cf856e0-1e31-4cec-9ef3-93ae8675ba0e‘
```

Z exekučního plánu 3.1 lze vypozorovat, že přestože vlastnost *userId* existuje jen u některých uzlů, databázový systém musel prohledat uzly všechny,

[46] 53,67% všech dotazů

@ver	docReads	current	docAnalyzedCompatibleClass	recordReads	resultType	resultSize	time
0	120000	#9:119999	120000	120000	collection	1	2,303 sec

Tabulka 3.1: Exekuční plán dotazu bez optimalizací

neboť jsou uloženy v jednom clusteru. První použitou optimalizací dle doporučení bylo zavedení schématu pro různé typy dat, které zajistí seskupení souvisejících dat. Po vytvoření třídy *User* s odpovídajícími vlastnostmi se dotaz změní na:

```
select from User where userId = ‘5cf856e0-1e31-4cec-9ef3-93ae8675ba0e‘
```

@ver	docReads	current	docAnalyzedCompatibleClass	recordReads	resultType	resultSize	time
0	100000	#12:99	100000	100000	collection	1	1,848 sec

Tabulka 3.2: Exekuční plán dotazu po vytvoření schématu

Nyní je dotaz proveden pouze nad podmnožinou všech uzlů, která reprezentuje uživatele. Tím je vysvětlen kratší čas vykonávání dotazu než v prvním případě. Další optimalizací dotazu bude přidání SB-Tree unikátního indexu na vlastnost *userId* třídy *User*. OrientDB v případě existence odpovídající index použije, takže dotaz zůstává stejný jako v předchozím případě:

```
select from User where userId = ‘5cf856e0-1e31-4cec-9ef3-93ae8675ba0e‘
```

Výsledky dle exekučního plánu 3.3 vykazují rapidní zrychlení dotazu při použití unikátního SB-Tree indexu. Databáze dle indexu vyhledala konkrétní uzel a pouze tento načetla.

Provedená měření potvrdily, že vytvoření schématu a aplikování indexů rapidně zvyšují rychlost dotazů do databáze. Na základě těchto výsledků byly pro všechny typy uzlů datového modelu vytvořeny třídy a na identifikátory aplikovány unikátní SB-Tree indexy.

3.1.2 Testování kapacity databáze

V databázi OrientDB lze dle dokumentace uchovávat až 2^{63} záznamů. Množství uzlů a hran, se kterým bude pracovat vyvinutý systém pro doporučování produktů bylo odhadnuto na hodnotu v řádu stovek tisíc záznamů a tím pádem je kapacita databáze dostatečná.

Během implemenetace řešení této práce databáze OrientDB obsahovala data v řádu stovek tisíců náhodných záznamů. Za tuto dobu nebylo pozorováno rapidní zhoršení výkonu databázového systému závislé na počtu záznamů.

@ver	docReads	compositeIndexUsed	current	docAnalyzedCompatibleClass	involvedIndexes	resultType	resultSize	time
0	1	1	#12:99	1	[‘User.userId‘]	collection	1	0.079 sec

Tabulka 3.3: Exekuční plán dotazu po založení schématu a vytvoření SB-Tree indexu

3.2 Jednotkové testování

Testování je nezbytnou součástí vývojového cyklu aplikace. Pravidelné testování odhaluje chyby již v průběhu implementace a umožňuje vývojáři jejich okamžitou opravu. Vzhledem k rozsáhlosti zdrojových kódů byla spolehlivost aplikace zajišťována pomocí jednotkového testování. Jednotkové testy jsou nástrojem pro udržování konzistence aplikace po celou dobu vývoje. Testy byly implementované pomocí frameworku JUnit[47].

Jednotkovým testováním byly testovány důležité komponenty knihovny zajišťující ukládání a načítání dat z grafové databáze, která byla popsána v kapitole 2.6. Třídy s testy `GenericRepositoryTest` a `SchemaTest` vyžadují komunikaci s inicializovanou databází. K tomuto účelu byla využita možnost spuštění grafové databáze OrientDB s datovým úložištěm v operační paměti. Pro případ testování je toto datové úložiště více než vhodné, neboť automatické smazání dat z operační paměti po skončení testů šetří práci s ručním mazáním databáze.

Všechny jednotkové testy jsou součástí zveřejněných zdrojových kódů této knihovny (package *cz.cvut.palislub.tests*).

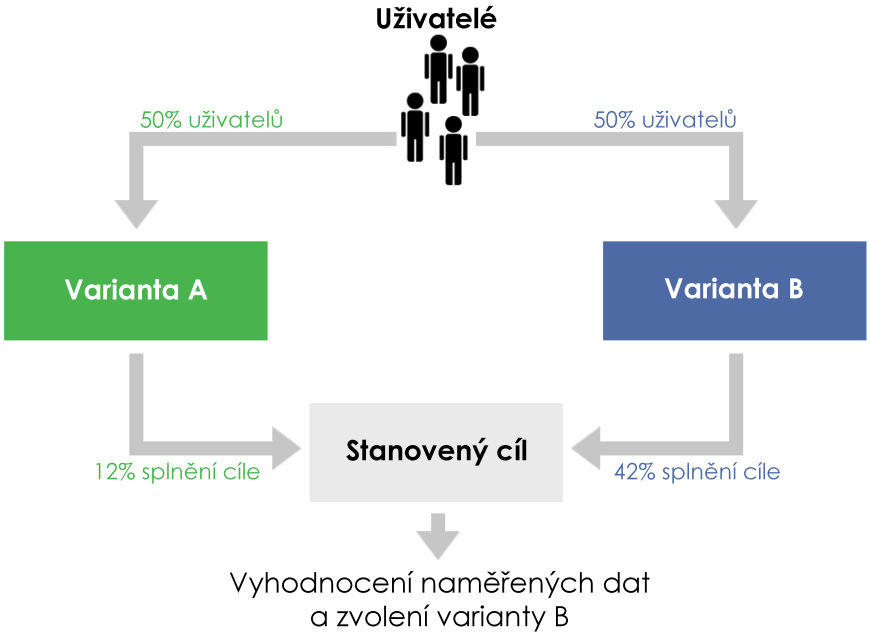
3.3 Testování kvality doporučování

Kvalita doporučování nelze jednoduše shrnout pod jednu metriku. Cílem doporučovacího systému je zvýšení tržeb internetového obchodu, ale v neposlední řadě také zlepšit použitelnost a uživatelský prožitek z webových stránek. Pro porovnání původní metody doporučování a nově implementovaného systému bylo použito A/B testování.

3.3.1 A/B testování

A/B testování je metoda, jak experimentálně porovnat efekt provedených změn. Princip A/B testování je založen na implementaci dvou variant (varianta A a varianta B) řešení stejného problému. Uživatelé jsou rozděleni do dvou skupin a každé skupině je zobrazena jedna z variant. Rozdělení uživatelů musí být náhodné a neovlivněné jinými faktory. Důležitou vlastností procesu A/B testování je zobrazování stále stejné varianty jednomu konkrétnímu uživateli.[29]

[47] http://junit.org/



Obrázek 3.1: Schéma metody A/B testování.

Pro každou variantu jsou sledovány předem definované cíle (například počet kliknutí nebo procento konverze). Testování probíhá na dostatečně velkém testovacím vzorku návštěvníků a po dostatečně dlouhou dobu, aby byly vyloučeny vedlejší vlivy. Po skončení experimentu jsou data vyhodnocena a verze s lepšími výsledky může být nasazena do produkčního prostředí pro všechny uživatele.

Při správném použití metody A/B testování získáme výsledky z praxe na reálných uživatelích a tím pádem je jejich vypovídající hodnota vysoká a relevantní [30].

Vzhledem k vysoké popularitě A/B testování existují nástroje, které proces testování a jeho vyhodnocování zjednodušují. Jmenuji například Optimizely[48] nebo Visual Website Optimizer[49].

[48] https://www.optimizely.com/
[49] https://vwo.com/

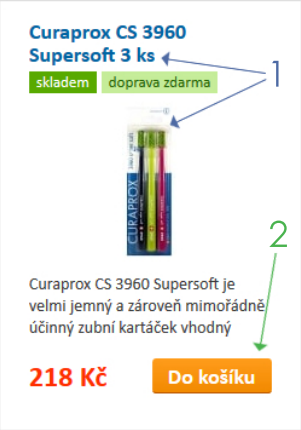
3.3.2 Implementace A/B testování

Součástí implementace doporučovacího systému bylo nasazení A/B testování. Varianta A je reprezentovaná původní metodou doporučování produktů, zatímco varianta B představuje doporučování produktů pomocí implementovaného doporučovacího systému.

Každá varianta má vlastní skupinu URL, které jsou asynchronně volány klientskou webovou aplikací. Náhodné rozhodnutí, zda se při požadavku na doporučení produktů volá URL varianty A nebo varianty B, je implementováno pomocí javascriptové metody *Math.random()*. Tato metoda vrací náhodné číslo v intervalu $\langle 0, 1)$ [31]. Pro vygenerované hodnoty nižší než 0,5 je zvolena varianta A, v opačném případě je zvolena varianta B.

Byly definované dva cíle, jejichž hodnota se bude měřit a analyzovat:

- Počet kliknutí na doporučený produkt (cíl označen číslem 1 na obrázku 3.2)
- Počet přímých vložení do košíku doporučeného produktu (cíl označen číslem 2 na obrázku 3.2)



Obrázek 3.2: Grafická podoba doporučené položky s vyznačenými cíly, které jsou předmětem měření A/B testování.

Samotné měření bylo implementováno pomocí nástroje Google Analytics[50] s rozšířením Ecommerce[51]. Využil jsem možnost tohoto nástroje měřit interakce uživatelů se seznamy produktů.

[50] http://www.google.com/analytics/ — Online nástroj poskytující informace o používání internetových stránek.

[51] https://developers.google.com/analytics/devguides/collection/analyticsjs/ecommerce — Rozšíření nástroje Google Analytics zaměřené na internetové obchody.

Pro všechna tři implementovaná místa doporučování bylo nakonfigurováno sledování počtu zobrazení produktů, měření jednotlivých kliknutí na produkt a zaznamenávání počtu přidání produktu do košíku. Každé místo bylo označeno štítkem a k tomuto štítku doplněna informace, zda se jednalo o seznam produktů získaný doporučovanou variantou A nebo variantou B.

Pomocí získaných dat je možné v nástroji Google Analytics vytvářet reporty se statistikami úspěšnosti definovaných cílů pro každou variantu. Tyto reporty budou podkladem pro analytika, aby vyhodnotil kvality jednotlivých přístupů doporučování a zvolil ten nejvhodnější.

Závěr

Cílem této diplomové práce bylo seznámit se s odvětvím grafových databází a analyzovat jejich použití v prostředí internetového obchodu. Vhodným příkladem aplikace grafové databáze byl shledán systém na doporučování produktů. Po důkladném studiu teorie doporučovacích systémů byl implementován systém pro kolaborativní a obsahové doporučování s použitím databáze OrientDB. Tento systém byl integrován do fungujícího internetového obchodu. Testování kvality doporučování bylo zajištěno metodikou A/B testování, jejíž výstupy budou v budoucnu analyzovány. Zadání tím bylo splněno ve všech bodech.

Z mého pohledu je hlavním přínosem pro čtenáře praktická ukázka vhodné aplikace grafové databáze na určitý typ problému. Práce popisuje důvody, které vedly ke zvolení databáze OrientDB. Následně je zveřejněna analýza řešení, nástroje a postupy použité při implementaci a pozdějším testování řešení.

Během implementace byl kladen důraz na oddělení programového kódu pracujícího s databází od zbytku aplikace. Z tohoto důvodu byla vyvinuta knihovna v jazyce Java pro databázový systém OrientDB, která zajišťuje objektově grafové mapování entity datového modelu. Knihovna zároveň implementuje základní CRUD operace pro práci s anotovanými entitami. Při vývoji byla snaha o podporu specifických vlastností databáze OrientDB jako jsou indexy nebo schémata. Tato knihovna vznikla jako vedlejší produkt práce a byla zveřejněna jako opensource projekt na serveru github.com[52].

Pro mě osobně byla práce velkým přínosem, neboť jsem měl možnost důkladně se seznámit s relativně novou technologií grafových databází. Ověřil jsem si její praktické využití na reálném projektu, který je použit v existujícím internetovém obchodu.

[52] https://github.com/shobull/orientdb-mapper

Budoucí práce

Přestože je výsledkem této práce plně funkční systém nasazený do produkčního prostředí internetového obchodu, nepochybně lze na dosud provedené poznatky navázat. Na základě A/B testování bude nutné pravidelně získané statistiky o doporučování vyhodnocovat a případně přizpůsobovat definované průchody grafem.

Vývoj databáze OrientDB je v současné době velmi aktivní a tento produkt je stále obohacován o nové funkce. Vzhledem k tomu by bylo vhodné tyto změny sledovat a průběžně přizpůsobovat naprogramovanou knihovnu pro ukládání a načítání dat z databáze.

Literatura

[1] CIO BUSINESS WORLD: *Internetový obchod v ČR zatím stále roste, na prvním místě je elektronika*. [online]. [cit. 2015-03-16]. Dostupné z: `http://businessworld.cz/analyzy/id-11247`

[2] TRUDEAU, R.: *Introduction to Graph Theory*. Dover Books on Mathematics Series, Dover Pub., 1993, ISBN 9780486678702.

[3] POKORNÝ, J.; HALAŠKA, I.: *Databázové systémy*. Praha: Vydavatelství ČVUT, vyd. 2. přeprac. vydání, 2003, ISBN ISBN 80-01-02789-9.

[4] LITH, A.; MATTSSON, J.: *Investigating storage solutions for large data - A comparison of well performing and scalable data storage solutions for real time extraction and batch insertion of data*. Diplomová práce, Department of Computer Science and Engineering, Chalmers University of Technology, Göteborg, 2010.

[5] BROWNE, J.: *Brewer's CAP Theorem*. [online]. [cit. 2015-04-25]. Dostupné z: `http://www.julianbrowne.com/article/viewer/brewers-cap-theorem`

[6] GILBERT, S.; LYNCH, N.: Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services. *SIGACT News*, ročník 33, č. 2, Červen 2002: s. 51–59, ISSN 0163-5700, doi:10.1145/564585.564601. Dostupné z: `http://doi.acm.org/10.1145/564585.564601`

[7] SALMINEN, A.: Introduction to NOSQL. In *NoSQL Seminar*, 2012. Dostupné z: `http://www.cs.tut.fi/~tjm/seminars/nosql2012/NoSQL-Intro.pdf`

[8] YU, E.; DOBBIE, G.; JARKE, M.; aj.: *Conceptual Modeling: 33rd International Conference, ER 2014, Atlanta, GA, USA, October 27-29,2014. Proceedings*. Lecture Notes in Computer Science / Information Systems

and Applications, incl. Internet/Web, and HCI, Springer International Publishing, 2014, ISBN 978-3-319-12206-9.

[9] ROBINSON, I.; WEBBER, J.; EIFREM, E.: *Graph Databases*. O'Reilly Media, 2013, ISBN 978-1-449-35626-2.

[10] THE WORLD WIDE WEB CONSORTIUM (W3C): *Resource Description Framework (RDF):Concepts and Abstract Syntax*. [online]. [cit. 2015-03-22]. Dostupné z: `http://www.w3.org/TR/2004/REC-rdf-concepts-20040210`

[11] SOLID IT: *DB-Engines Ranking of Graph DBMS*. [online]. [cit. 2015-04-25]. Dostupné z: `http://db-engines.com/en/ranking/graph+dbms`

[12] RETAIL TECHNOLOGY: *Walmart and eBay adopt graph database*. [online]. [cit. 2015-03-22]. Dostupné z: `http://www.retailtechnology.co.uk/news/5187/walmart-and-ebay-adopt-graph-database/`

[13] HINZ, O.; ECKERT, J.: The Impact of Search and Recommendation Systems on Sales in Electronic Commerce. *Business & Information Systems Engineering*, ročník 2, č. 2, 2010: s. 67–77, doi:10.1007/s12599-010-0092-x. Dostupné z: `http://dx.doi.org/10.1007/s12599-010-0092-x`

[14] PAZZANI, M. J.; BILLSUS, D.: Content-Based Recommendation Systems. In *The Adaptive Web*, *Lecture Notes in Computer Science*, ročník 4321, editace P. BRUSILOVSKY; A. KOBSA; W. NEJDL, Springer Berlin Heidelberg, 2007, ISBN 978-3-540-72078-2, s. 325–341, doi:10.1007/978-3-540-72079-9_10. Dostupné z: `http://dx.doi.org/10.1007/978-3-540-72079-9_10`

[15] SARWAR, B.; KARYPIS, G.; KONSTAN, J.; aj.: Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, ACM, 2001, s. 285–295.

[16] PARTNER, J.; VUKOTIC, A.; WATT, N.: *Neo4j in Action*. Manning Publications Company, 2014, ISBN 9781617290763.

[17] SOLID IT: *Method of calculating the scores of the DB-Engines Ranking*. [online]. [cit. 2015-03-23]. Dostupné z: `http://db-engines.com/en/ranking_definition`

[18] NEO TECHNOLOGY, INC.: *Subscriptions*. [online]. [cit. 2015-04-26]. Dostupné z: `http://neo4j.com/subscriptions`

[19] SMALL, N.: *Neo4j Index Confusion*. [online]. [cit. 2015-03-23]. Dostupné z: `http://nigelsmall.com/neo4j/index-confusion`

[20] NEO TECHNOLOGY, INC.: *Transaction Management - The Neo4j Manual v2.1.7*. [online]. [cit. 2015-03-23]. Dostupné z: http://neo4j.com/docs/stable/transactions.html

[21] ORIENT TECHNOLOGIES LTD: *Released OrientDB Enterprise 2.0.3, now FREE for Development!* [online]. [cit. 2015-04-03]. Dostupné z: http://www.orientechnologies.com/released-orientdb-enterprise-2-0-3-now-free-development/

[22] INTERNET ENGINEERING TASK FORCE: *The JavaScript Object Notation (JSON) Data Interchange Format*. [online]. [cit. 2015-03-23]. Dostupné z: http://tools.ietf.org/html/rfc7159

[23] ANALYTICS NINJA LLC: *How "Unique" are Unique Visitors in Google Analytics*. [online]. [cit. 2015-03-25]. Dostupné z: http://www.analytics-ninja.com/blog/2011/12/how-unique-are-unique-visitors-in-google-analytics.html

[24] ORIENT TECHNOLOGIES LTD: *OrientDB Manual - version 2.0 - Performance Tuning*. [online]. [cit. 2015-03-27]. Dostupné z: http://www.orientechnologies.com/docs/last/Performance-Tuning.html#massive_insertion

[25] FORMAN, I. R.; FORMAN, N.: *Java Reflection in Action*. Manning Publications Co., 2004, ISBN 1932394184.

[26] VAVRUŠKA, J.: ETL a kvalita dat. *Časopis IT Systems*, , č. 3, 2003: str. PŘÍLOHA #2. Dostupné z: http://www.systemonline.cz/clanky/etl-a-kvalita-dat.htm

[27] GAMMA, E.; HELM, R.; JOHNSON, R.; aj.: *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, 1994, ISBN 9780321700698.

[28] ORIENT TECHNOLOGIES LTD: *Tuning the Graph API*. [online]. [cit. 2015-04-14]. Dostupné z: http://orientdb.com/docs/last/Performance-Tuning-Graph.html

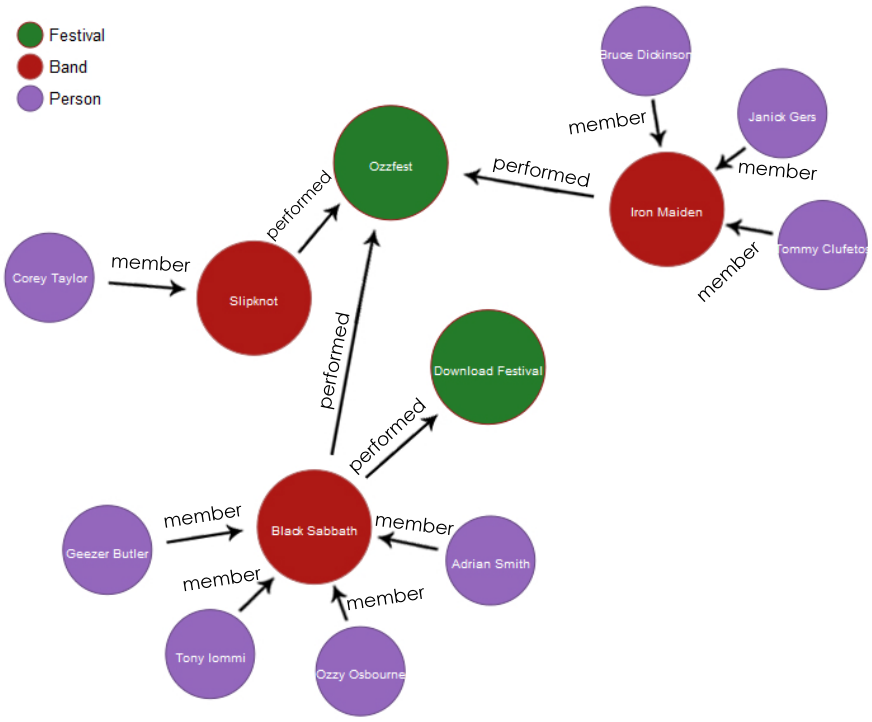
[29] KOHAVI, R.; LONGBOTHAM, R.; SOMMERFIELD, D.; aj.: Controlled Experiments on the Web: Survey and Practical Guide. *Data Min. Knowl. Discov.*, ročník 18, č. 1, únor 2009: str. 149, ISSN 1384-5810, doi:10.1007/s10618-008-0114-1.

[30] SNÍŽEK, M.: *A/B testování – kompletní průvodce*. [online]. [cit. 2015-04-07]. Dostupné z: http://www.optimics.cz/c/ab-testovani-kompletni-pruvodce

[31] W3SCHOOLS: *JavaScript random() Method*. [online]. [cit. 2015-04-07]. Dostupné z: `http://www.w3schools.com/jsref/jsref_random.asp`

Příloha **A**

Doplňkové materiály



Obrázek A.1: Doménový model pro příklad průchodu grafem z kapitoly1.4.2

```java
package cz.cvut.fit.example.domain.nodes;

import cz.cvut.palislub.annotations.Node;
import cz.cvut.palislub.annotations.NodeProperty;
import cz.cvut.palislub.annotations.Unique;

@Node(name = "User")
public class GraphUser {
        @NodeProperty
        @Unique
        public String userId;

        @NodeProperty
        @Indexed
        public String username;

        @NodeProperty
        public String email;

        public String getUserId() {
                return userId;
        }

        public String getUsername() {
                return username;
        }

        public String getEmail() {
                return email;
        }

        public void setUserId(String userId) {
                this.userId = userId;
        }

        public void setUsername(String username) {
                this.username = username;
        }

        public void setEmail(String email) {
                this.email = email;
        }
}
```

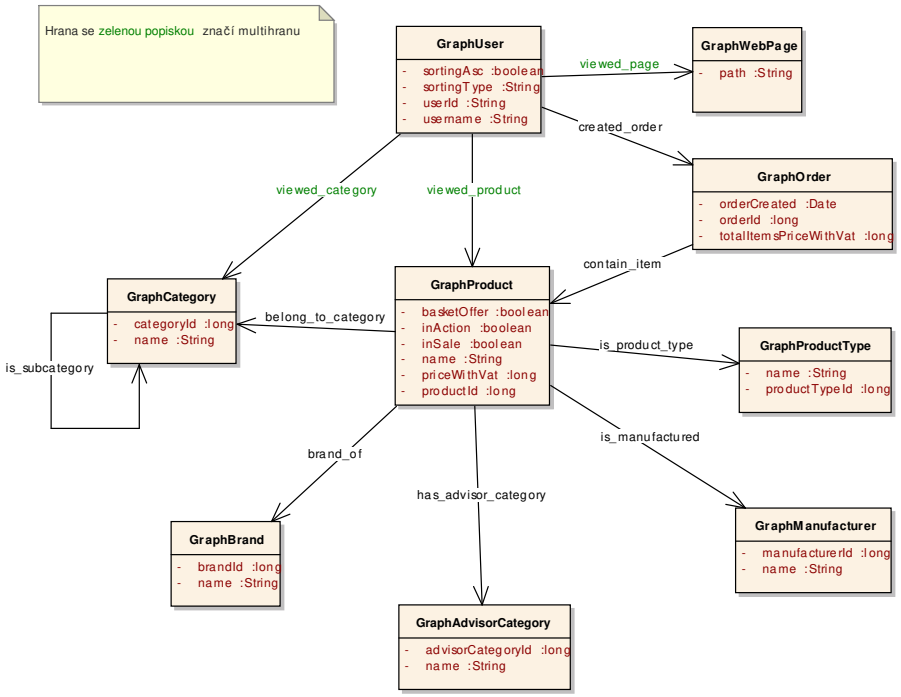
Zdrojový kód A.1: Objekt *User* (uzel) doménového modelu pro grafovou databázi s anotacemi implementovanými ve vlastní knihovně.

```java
package cz.cvut.fit.example.domain.relationships;

import cz.cvut.palislub.annotations.NodeFrom;
import cz.cvut.palislub.annotations.NodeTo;
import cz.cvut.palislub.annotations.Relationship;
import cz.cvut.palislub.annotations.RelationshipProperty;
import cz.cvut.palislub.example.domain.nodes.GraphUser;
import cz.cvut.palislub.example.domain.nodes.GraphWebPage;

import java.util.Date;

@Relationship(type = "viewed_page", unique = false)
public class PageViewRelationship {

        @NodeFrom
        private GraphUser user;

        @NodeTo
        private GraphWebPage webpage;

        @RelationshipProperty
        private boolean enterVisit = false;

        @RelationshipProperty
        private Date visitedTime;

        @RelationshipProperty
        private long visitLength;

        public PageViewRelationship(GraphUser user, GraphWebPage webpage) {
                this.user = user;
                this.webpage = webpage;
        }

        // setters and getters
}
```

Zdrojový kód A.2: Objekt *PageViewRelationship* (hrana) doménového modelu pro grafovou databázi s anotacemi implementovanými ve vlastní knihovně.

```java
package cz.cvut.fit.example.repo;

import com.tinkerpop.gremlin.java.GremlinPipeline;
import cz.cvut.palislub.repository.GenericGraphRepo;
import cz.cvut.fit.example.repo.domain.GraphUser;

import java.util.HashMap;

public class GraphUserRepo extends GenericGraphRepo<GraphUser, String> {

        public GraphUserRepo() {
                super(GraphUser.class);
        }

        public HashMap<Long, Long> getCollaborativeEnterPageRecommendation(String userId) {
                GremlinPipeline collaborativeEnterPagePipeline = new GremlinPipeline();
                collaborativeEnterPagePipeline.start(getVertex(userId))
                                .outE("viewed_page")
                                .has("enter_visit", true)
                                .inV()
                                .inE("viewed_page")
                                .has("enter_visit", true)
                                .outV()
                                .out("created_order")
                                .out("contain_item")
                                .property("productId")
                                .groupCount()
                                .cap();

                return (HashMap<Long, Long>) collaborativeEnterPagePipeline.next();
        }
}
```

Zdrojový kód A.3: Ukázka třídy implementující *GenericGraphRepo.class* pro operace nad daty grafové databáze.



Obrázek A.2: Datové schéma grafové databáze

```
{
        "config": { "verbose": true },
        "extractor": {
                "jdbc": {
                    "driver": "com.mysql.jdbc.Driver",
                    "url": "jdbc:mysql://localhost/database",
                    "userName": "username",
                    "userPassword": "password",
                    "query": "SELECT id, parent_id, name FROM shop_category WHERE shop_id = 1"
                }
        },
        "transformers": [
                { "merge": {
                                "joinFieldName": "categoryId",
                                "lookup": "Category.categoryId"
                } },
                { "vertex": { "class": "Category" } },
                { "edge": {
                        "class": "is_subcategory",
                        "joinFieldName": "parent_id",
                        "lookup": "Category.categoryId",
                        "unresolvedLinkAction": "CREATE",
                        "if": "parent_id is not null"
                } },
                { "field": { "fieldName": "categoryId": "expression": "id"  } },
                { "field": { "fieldName": "name": "expression": "name"  } },
                { "field": { "fieldName": "id": "operation": "remove"  } },
                { "field": { "fieldName": "parent_id": "operation": "remove"  } },
                { "log": { "prefix": "Created Vertex: " } }
        ],
        "loader": {
                "orientdb": {
                "dbURL": "remote:localhost/testdb",
                "dbUser": "admin",
                "dbPassword": "admin",
                "tx": true,
                "batchCommit": 1000,
                "dbType": "graph",
                "indexes": [
                        {"class":"Category", "fields":["categoryId:long"], "type":"UNIQUE" }
                ] }
        }
}
```

Zdrojový kód A.4: Příklad konfigurace ETL procesu pro import kategorií produktů.

Základní deska	Gigabyte GA-965P-DQ6
CPU	Intel®Core^{TM}2 Duo CPU E6400 (2M Cache, 2.13 GHz, 1066 MHz FSB)
RAM	6,00 GB DDR2, 800Mhz
Grafická karta	ATI Radeon x1950 Pro, 256MB DDR3
Pevný disk	WD Caviar WD3200KS 7200 RPM 16MB Cache SATA
Operační systém	Windows 7 Professional SP1 64-bit
IDE	IntelliJ Idea 14.0.2 x64

Tabulka A.1: Hardware a software konfigurace použitá pro testování

Příloha B

Seznam použitých zkratek

ACID Atomicity, Consistency, Isolation, Durability

API Application Programming Interface

BASE Basicaly Available, Soft state, Eventual consistency

CAP Consistency, Availability, Partition tolerance

CRUD Create, Read, Update, Delete

DBMS Database Managment System

DTO Data transfer object

ETL Extractor, Transformers, Loader

HTML HyperText Markup Language

JPA Java Persistence API

JSON JavaScript Object Notation

MVCC Multiversion concurrency control

NoSQL Not only SQL

OLTP Online Transaction Processing

OOP Objektově orientované programování

ORM Objektově relační mapování

PDF Portable Document Format

POJO Plain Old Java Object

RDBMS Relational Database Managment System

RDF Resource Description Framework

REST Representational State Transfer

RID Record ID

SQL Structured Query Language

XML Extensible Markup Language

Příloha C

Obsah přiloženého CD

- `domain` ..... adresář s Enterprise Architekt projektem datového modelu
- `etl` ..... skripty pro ETL procesy
- `orientdb-mapper` ..... zdrojové kódy knihovny pro práci s OrientDB
- `recommendation-system` ..... zdrojové kódy doporučovacího systému
- `text` ..... text práce
  - `latex` ..... zdrojová forma práce ve formátu LaTeX
  - `DP_Palisek_Lubos_2015.pdf` ..... text práce ve formátu PDF
- `orientdb-mapper.jar` ..... zkompilovaná knihovna pro práci s OrientDB
- `readme.txt` ..... stručný popis obsahu CD