

MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY



Implementace fulltextového vyhledávání v systému správy požadavků

BAKALÁŘSKÁ PRÁCE

Jiří Holuša

Brno, Jaro 2014

Prohlášení

Prohlašuji, že tato bakalářská práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Jiří Holuša

Vedoucí práce: Mgr. Filip Nguyen

Poděkování

TODO: poděkování

Shrnutí

TODO: abstrakt

Klíčová slova

TODO: klíčová slova

Obsah

1	Úvod	2
2	Vyhledávání	3
2.1	Vyhledávání v relačních databázích	3
2.2	Problémy vyhledávání v relačních databázích	4
2.2.1	Vyhledávání přes několik tabulek	4
2.2.2	Vyhledávání jednotlivých slov	4
2.2.3	Filtrace šumu	5
2.2.4	Vyhledávání příbuzných slov	5
2.2.5	Oprava překlepů	5
2.2.6	Relevance	5
2.3	Fulltextové vyhledávání	6
2.3.1	Úvod do fulltextového vyhledávání	6
2.3.2	Indexace	6
2.3.3	Hledání	6
3	Dostupné technologie	7
3.1	Apache Lucene	7
3.2	Hibernate Search	7
3.3	Elasticsearch	8
4	Implementace	9
4.1	Výběr technologie	9
4.2	Indexace	9
4.3	Vyhledávání	10
4.4	Import dat	10
5	Závěr	11

1 Úvod

Úvod

2 Vyhledávání

Tato kapitola stručně popisuje způsob vyhledávání v nejčastějším datovém úložišti - relačních databázích - a uvádí jeho nedostatky. Poté se detailněji věnuje jednou z možností jejich řešení, a to fulltextovým vyhledáváním. Uvádí nezbytnou teorii k pochopení principů, jak fulltextové vyhledávání funguje, jeho výhody a nevýhody.

2.1 Vyhledávání v relačních databázích

Relační databáze poskytují vysoce výkonný přístup k datům a široké možnosti pro jejich správu. Díky svým schopnostem se staly nejpoužívanější technologií pro datové úložiště. Vyhledávat v datech lze přitom pouze dvěma způsoby: porovnání obsahu buňky a operátor LIKE.

Porovnání obsahu buňky funguje na velice jednoduchém principu úplné shody obsahu. V následujícím příkladu vidíme dotaz v jazyce SQL, který vybere právě ty záznamy z tabulky People, které mají hodnotu atributu name rovnou "Bruce Banner".

```
SELECT * FROM People WHERE name = 'Bruce Banner'
```

Nebudou tedy vybrány žádné jiné záznamy, přestože by obsah atributu name měly např. "Bruce Banners" či dokonce ani "Bruce Banner" (přebytečná mezera na konci). Výhodou tohoto řešení je efektivita a jednoduchost - jediná nutná operace je pouze porovnání dvou řetězců, žádné dodatečné zpracování není potřeba.

Trochu více sofistikovaným způsobem je operátor LIKE, který umožňuje (v omezené míře) používat pattern matching - vyhledávání pomocí vzoru. Podporovány jsou tzv. zástupné symboly, jež mohou mít v tomto kontextu jiný význam než jen právě daný znak, např. symbol % (procento) zastupuje libovolnou sekvenci znaků (třeba i žádnou) nebo znak . (tečka) libovolný, ale právě jeden znak. Níže vidíme příklad SQL dotazu, jenž nám vrátí všechny záznamy z tabulky People, které jejich jméno končí na "Banner".

```
SELECT * FROM People WHERE name LIKE '%Banner'
```

Nyní již dokážeme tímto dotazem získat jak lidi se jménem "Bruce Banner", tak i "Richard Banner".

2.2 Problémy vyhledávání v relačních databázích

V předchozí kapitole jsme si představili základní způsoby vyhledávání v relačních databázích. Nyní se podíváme na případy, kde nám tyto způsoby nestačí nebo si s danou situací nedokáží poradit buď vůbec, nebo pouze neefektivně.

Abychom si mohli tyto nedostatky demonstrovat na příkladech, uvažujme existenci jednoduché relační databáze s následujícím schématem.

TODO: obrázek

2.2.1 Vyhledávání přes několik tabulek

Představme si, že uživatel zadá do vyhledávacího políčka nějaký řetězec, na jehož základě očekává odpovídající výsledky. Vyvstává otázka, kde bychom měli zadanou frázi hledat. V našem případě pravděpodobně v nadpisu, popisu, ve jméně a příjmení autora, tam všude by se mohly nacházet informace, které uživatel hledal.

SQL nyní musí prohledat všechny zadané sloupce, které se však mohou nacházet v různých tabulkách, což vede ke spojování tabulek. Výsledný dotaz by mohl vypadat například takto:

```
SELECT * FROM Book book LEFT JOIN book.authors author WHERE  
book.title = ? OR book.description = ?  
OR author.firstname = ? OR author.lastname = ?
```

Je vidět, že i při relativně jednoduchém požadavku (vyhledáváme pouze ve čtyřech sloupcích) je výsledný dotaz poměrně složitý. Pokud bychom chtěli uživateli dát možnost využívat komplexnější dotazy, je otázka generování odpovídajících SQL dotazu netriviální. Navíc uvažme, že musíme spojit více tabulek, což může vést k problémům s efektivitou.

2.2.2 Vyhledávání jednotlivých slov

Jak jsme si řekli, SQL dokáže vyhledat v jednotlivých sloupcích přesně zadanou frázi. Je ovšem velice nepravděpodobné, že sloupce v databázi budou obsahovat přesně stejnou danou frázi, hledání jednotlivých slov by nám velice zvýšilo pravděpodobnost nálezu. SQL však žádnou takovou funkcionalitu na dělení vět neposkytuje, je tedy nutné si větu předpřipravit explicitně (tj. rozdělit na slova), a poté spouštět vyhledávací dotaz pro každé slovo zvlášť. Následně výsledky nějakým způsobem sloučit.

2.2.3 Filtrace šumu

Některá slova ve větách nenesou vzhledem k vyhledávání žádnou informační hodnotu, např. spojky či předložky či ještě lepším příkladem mohou být anglické neurčité členy. Taková slova se nazývají šum (noise). Dále se pak některá slova v určitém kontextu šumem stávají, např. slovo kniha v našem internetovém knihkupectví. SQL nám opět neposkytuje žádný prostředek k řešení takového problému, je tedy nutné data externě předpřipravit a případná filtrace šumu musí být vyřešena jinak.

2.2.4 Vyhledávání příbuzných slov

Je velice žádoucí, abychom se při vyhledávání mohli zaměřit pouze na význam hledaného slova, nikoliv na jeho tvar. Nemělo by záležet na tom, zda hledáme frázi fulltextové hledání nebo fulltextových vyhledávání, význam těchto frází je stejný. Jinak řečeno, vyhledávání by mělo brát v potaz i slova odvozená, se stejným kořenem. Ještě pokročilejším požadavkem by mohla být možnost zaměňovat slova s jejich synonymy, např. upravit a editovat.

SQL nám opět nenabízí možnost k řešení těchto požadavků, klíčem by mohl být slovník příbuzných slov a synonym a pokusit se vyhledávat i podle něj. Takové řešení však přináší nezanedbatelné množství práce, nehledě na nutnost existence takového slovníku.

2.2.5 Oprava překlepů

Uživatel je člověk a jako člověk je to tvor omylný a dělá chyby. Vyhledávání by to mělo brát v potaz a snažit se tyto překlepy opravit či uhodnout, co měl uživatel na mysli. Když v našem internetovém knihkupectví uživatel hledá knihu "Fulltextové vyhledávání" a omylem zadá do vyhledávacího pole "Fulltetové vyhledávání", je žádoucí, aby i přes tento překlep knihu našel.

2.2.6 Relevance

Pravděpodobně největším problémem v SQL je absence jakéhokoliv mechanismu pro určení míry shody (relevance) záznamu se zadaným dotazem. Předpokládejme, že v našem knihkupectví napsal autor "John Smith" 100 knih, jednu o fulltextovém vyhledávání a zbytek naprosto nesouvisející s informatikou. Dále několik dalších autorů rovněž napsalo publikace na téma fulltextového vyhledávání.

Pokud jako uživatel víme, že je autorem John Smith a kniha je o fulltextovém vyhledávání, očekáváme, že na vyhledávací dotaz "John Smith full-

textové vyhledávání"obdržíme nejdříve právě chtěnou knihu, a poté teprve knihy ostatní od našeho autora či další knihy o fulltextovém vyhledávání, jelikož naše kniha "nejvíce"odpovídala položenému dotazu.

2.3 Fulltextové vyhledávání

V předchozí kapitole jsme si uvedli, jaké problémy má vyhledávání pouze pomocí SQL. Nyní si představíme možné řešení - fulltextové vyhledávání.

2.3.1 Úvod do fulltextového vyhledávání

Fulltextové vyhledávání (někdy také fulltext nebo full-text) je speciální způsob vyhledávání informací v textu. Vyhledávání probíhá porovnáváním s každým slovem v hledaném textu. Jelikož počet slov v textu může teoreticky neomezený a jelikož je nutné, aby vyhledávání bylo co nejrychlejší, funguje fulltextové vyhledávání ve dvou fázích: indexace a hledání.

2.3.2 Indexace

Indexace je proces předpřipravení dat, ve kterých se následně bude hledat. Spočívá ve zpracování textu předem, ve sběru takových informací o struktuře textu, s jejichž využitím následně dokážeme vlastní hledání enormně zrychlit, a tak zachovat rychlost hledání i nad velkým množstvím textu.

Hlavním úkolem indexace je rozdělit text do slov a následně vybudovat datovou strukturu, v níž jde efektivně nalézt hledané slovo. Této datové struktuře se říká index, ve které si fulltextové vyhledávání poznačí pro každé slovo všechny pozice, na kterých se vyskytuje.

Pouhým rozdělením do slov však možnosti předpřipravení textu nekončí a může být zapojena složitá analýza. V praxi je pak každé slovo předáno analyzátoru, který může index libovolně upravovat, a tím ho lépe připravit na nadcházející dotazování, a umožnit mu odpovídat na složitější dotazy. Typickým příkladem možné analýzy je úprava podstatných jmen do základního tvaru (např. z množného čísla na jednotné), přidání do indexu synonym či slov odvozených.

2.3.3 Hledání

Samotné vyhledávání v textu je ve fulltextovém vyhledávání realizováno nikoliv nad textem samotným, ale nad předpřipraveným indexem z procesu indexace. Vyhledávací nástroj tedy může využít doplňkových informací o textu, které dokáží vyhledávání podstatně zrychlit.

3 Dostupné technologie

Na platformě Java existuje řada dostupných volně širitelných vyhledávacích technologií. Nyní si představíme tři z nich: Apache Lucene, Hibernate Search a následující kapitole trochu podrobněji Elasticsearch.

3.1 Apache Lucene

Lucene je vysoce výkonná, škálovatelná, volně širitelná vyhledávací knihovna napsána v jazyce Java. Autorem projektu, který vznikl v roce 1997, je Doug Cutting. V roce 2000 zveřejnil Lucene na stránkách serveru SourceForge.com a uvolnil ji tak zdarma pro komunitu. O rok později byla adoptována organizací Apache Software Foundation. Od té doby se knihovna neustále vyvíjela a v dubnu roku 2014 je aktuálně dostupná ve své nejnovější verzi 4.7.1.

Již několik let je Lucene nejpopulárnější vyhledávací technologií zdarma. Díky své popularitě se však dočkala i přepsání do dalších jazyků než je Java jako například Perl, Python, Ruby, C/C++, PHP a C# (.NET). Projekt je stále aktivně vyvíjen s širokou komunitní základnou.

Apache Lucene není hotová vyhledávací aplikace, je to knihovna, nástroj, poskytující všechny potřebné prostředky, aby mohla být taková aplikace pro vyhledávání naprogramována. Nabízí rozhraní pro vytváření, úpravu indexu, zpracování dat před indexací a tvorbu, úpravu dotazů. O zbytek úkonů se musí programátor postarat sám, z čehož vyplývají hlavní výhoda (robustnost, univerzálnost použití), ale také hlavní nevýhoda (složitost nasazení).

3.2 Hibernate Search

Po rozmachu technologie ORM na platformě Java a její nejznámější implementace Hibernate ORM bylo nutné rovněž dát tomuto nástroji možnosti fulltextového vyhledávání, o což se postarala právě knihovna Hibernate Search, která umožňuje Hibernate ORM fulltextové vyhledávání. Autorem je Emmanuel Bernard. Svou popularitu získala především, jak již název napovídá, perfektní integrací s Hibernate ORM. Integrace do již existujícího systému na bázi Hibernate ORM je velice jednoduchá a přináší do aplikace možnost fulltextového vyhledávání s plnou silou Apache Lucene, neboť je na této technologii postavena.

3.3 Elasticsearch

Elasticsearch je realtime distribuovaný vyhledávací a analytický nástroj. Historie této technologie se začala psát v roce 2004, kdy Shay Banon vytvořil Compass. Postupným vývojem a změnou požadavků však dospěl k názoru, že aby se mohl Compass stát distribuovanou technologií, bylo by zapotřebí ho značnou část přepsat. Rozhodl se proto naprogramovat zcela nový nástroj, který měl být již od počátku distribuovaný. První verze Elasticsearch byla vydána v únoru 2010.

Elasticsearch je postaven na dříve představené technologii Apache Lucene, k níž však přidává další klíčové vlastnosti. Řeč je zejména o celé architektuře. Elasticsearch není na rozdíl od Apache Lucene knihovnou, Elasticsearch tvoří samostatný distribuovaný systém serverů, které na pozadí používají Apache Lucene, ovšem skrývají její složitost a poskytují služby v mnohem jednodušším uživatelském API. Dalším rozdílem je pak to, že tyto služby jsou vystavují skrze REST API, čímž získáváme naprostou nezávislost na programovacím jazyku na rozdíl od Apache Lucene.

4 Implementace

V předchozích kapitolách jsme si představili fulltextové vyhledávání a dostupné technologie pro jeho implementaci na platformě Java.

Nyní se věnujeme skutečné implementaci fulltextového vyhledávání v systému správy požadavků eShoe. Celou implementaci lze rozdělit na několik částí: výběr technologie, indexace, vyhledávání a import dat.

4.1 Výběr technologie

Pro vlastní implementaci byla zvolena technologie Elasticsearch.

4.2 Indexace

První z problémů, který je potřeba vyřešit, je zvládnutí procesu indexace, tedy denormalizaci entit do formátu, jež se dá přímo předat Elasticsearch serveru. Jak již víme, Elasticsearch přijímá JSON-like objekty, tedy dvojice klíč - hodnota. Jedním z prvních možných řešení indexace je prostá manuální tvorba indexu z entity pomocí get metod, tedy pro každou třídu je vytvořen mechanismus, který v předem daném pořadí předem dané atributy získá a vytvoří z nich index.

Nevýhoda tohoto řešení je zjevná - nulová flexibilita. Při každé úpravě entity je nutné dopsat odpovídající mechanismus, který upravený atribut denormalizuje. Navíc je toto řešení udělané přesně na míru tomuto projektu, resp. přesně danému modelu entit, tudíž není znovupoužitelné do budoucna. Výhoda je ovšem rovněž zřejmá - jednoduchost. K naprogramování takového mechanismu není potřeba víc než základní znalost jazyka Java.

Rozhodli jsme se však ubírat jinou, sofistikovanější, cestou. Denormalizace je řešena pomocí anotací velice podobně jako v Hibernate Search. Myšlenkou bylo vytvořit samostatný projekt, který má ambice se později začlenit přímo do projektu Elasticsearch. To však vyžaduje vytvořit robustní a široce použitelný nástroj, nikoliv na míru vytvořený pouze pro potřeby projektu eShoe.

Základem je použití anotací, což zaručuje vysokou eleganci a jednoduchost použití. Inspirace vzešla právě z Hibernate Search, tedy oannotovat datový model (entity) nově vytvořenými anotacemi tak, aby po předání takové entity našemu indexačnímu manažeru nástroj sám vytvořil příslušný JSON dokument, který by následně automaticky odeslat do Elasticsearch serveru. Tím se z pohledu klienta redukuje proces tvorby indexu na pouhé vybrání

vlastností entit, podle kterých se následně bude vyhledávat a zavolání jediné metody, o vše ostatní se již nástroj postará sám.

Přesně podle této myšlenky byl založen nový projekt `elasticsearch-annotations`.
Elasticsearch-annotations poskytuje velice malé veřejné API, což usnadňuje jeho použití. Jádrem projektu je třída `IndexManager`.

4.3 Vyhledávání

4.4 Import dat

5 Závěr

Závěr

Literatura

- [1] Douglas Adams. *The Restaurant at the End of the Universe*. The Hitchhiker's Guide to the Galaxy. Pan Macmillan, 1980.
- [2] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, 1 edition, 1994.