ulm university universität
uulm

Fakultät für Ingenieurwissenschaften und Informatik
Institut für Datenbanken und Informationssysteme

Bachelorarbeit
im Studiengang Informatik

Apache Solr

Entwicklung von Szenarien zu typischen Suchanfragen und praktische Untersuchung der Szenarien am Beispiel des Apache Lucene - Solr Frameworks

vorgelegt von

Tim Mohring

Februar 2015

1. Gutachter	Prof. Dr. Manfred Reichert
Betreuer:	Rüdiger Pryss
Matrikelnummer	731890
Arbeit vorgelegt am:	19.02.2015

Kurzfassung

In dieser Arbeit werden einige Szenarien der Volltextsuche, wie sie im realen Betrieb in Unternehmen zum Einsatz kommen, anhand von *Solr*, untersucht. Dabei wird betrachtet, inwieweit diese sich realisieren lassen und welche Konfigurationen nötig sind, um sie zu realisieren.
Suchsysteme sind für jede moderne IT-Infrastruktur wichtig [BBP09]. Dazu ist ein leistungsfähiges Programm zur Volltextsuche nötig. Solr ist ein Open Source Server für diesen Zweck. Er wurde von Apache als ein Unterprojekt von Apache Lucene entwickelt und ist komplett in Java geschrieben. Als Grundlage dient ein anderes Projekt aus Lucene, Lucene Core [Foub]. Dies ist eine Bibliothek für die Volltextindizierung und bietet einige Funktionen für die Indizierung und Suche an. Solr baut darauf auf, bietet jedoch zusätzlich weitere Funktionen, wie zum Beispiel ein grafisches Administrationsinterface, APIs für verschiedenen Programmiersprachen und eine Facettensuche.
In dieser Arbeit werden außerdem auch andere Open Source Server für die Volltextsuche betrachtet. Weiter wird die Architektur von Solr, also die Funktionen und Realisierungen der einzelnen Komponenten, sowie das Zusammenspiel dieser, betrachtet. Außerdem wird die Einbindung in externe Programme, also die API, betrachtet. Solr bietet eine REST-API über JSON und XML. Dabei werden drei verschiedene Arten von APIs unterschieden. Die Schema API, die zur Ausgabe der aktuellen Konfiguration von Solr verwendet wird. Die CoreAdmin API, die verwendet wird, wenn Solr in der Cloud betrieben wird. Die "Standard"-API, die für die Suche und Indizierung verwendet wird. Für die Suche und Indizierung stehen außerdem verschiedene Client APIs zur Verfügung.

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich bei der Fertigstellung dieser Arbeit unterstützt und motiviert haben. Zunächst möchte ich mich bei Herrn Prof. Dr. Manfred Reichert (Universität Ulm, Institut für Datenbanken und Informationssysteme) für die Begutachtung dieser Bachelorarbeit bedanken.
Mein besonderer Dank gilt meinem Betreuer Herrn Rüdiger Pryss (Universität Ulm, Institut für Datenbanken und Informationssysteme), der mich bei der Anfertigung dieser Bachelorarbeit sehr gut unterstützt hat und viel Zeit in die Korrektur der Arbeit gesteckt hat.
Dann möchte ich mich noch bei meinen Freunden, meiner Familie und meinen Kommilitonen bedanken, die mich während des Studiums motiviert und unterstützt haben.

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sinngemäße Übernahmen aus anderen Werken sind als solche kenntlich gemacht und mit genauer Quellenangabe (auch aus elektronischen Medien) versehen.

Ulm, den 19.Februar.2015 Tim Mohring

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Motivation . . . 1
- 1.2 Ziel der Arbeit . . . 2
- 1.3 Aufbau der Arbeit . . . 2

2 Verwandte Arbeiten **3**

3 Architektur **6**
- 3.1 Verarbeitungseinheiten . . . 7
- 3.2 Anwendungsschicht . . . 8
- 3.3 Lucene Core . . . 9
- 3.4 IndexReplicator . . . 10
- 3.5 Fazit . . . 10

4 API **11**
- 4.1 Schema API . . . 11
- 4.2 Core Admin API . . . 14
- 4.3 "Standard"-API . . . 14
  - 4.3.1 Suche . . . 14
  - 4.3.2 Indizierung . . . 16
  - 4.3.3 Löschen . . . 19
  - 4.3.4 Optimierung und Commit . . . 19
  - 4.3.5 Client APIs . . . 20

5 Anforderungsanalyse **21**

6 Realisierung mit Solr **22**
- 6.1 Indizierung verschiedener Dateiformate . . . 22
  - 6.1.1 Indizierung über das AdminUI . . . 23
  - 6.1.2 Indizierung über das SimplePostTool . . . 25
  - 6.1.3 Indizierung über die API . . . 26
  - 6.1.4 Indizierung von Datenbanken . . . 26
- 6.2 Tägliche/Dynamische Indizierung der Datenbasis . . . 31
- 6.3 Separate Indizierung eines passwortgeschützen Bereichs . . . 32
- 6.4 Einschränkung von Suchen . . . 33
- 6.5 Plausibilität von Suchen . . . 34
- 6.6 Suchvarianten . . . 35
- 6.7 Formatierung der Suchergebnisse . . . 36

6.7.1 Query Highlighting 36
6.7.2 Ausgabe der Suchgeschwindigkeit 37
6.7.3 Anzahl der Ergebnisse 37
6.7.4 Ausgabe von Synonymen 38
6.7.5 Sortierung nach Relevanz 39
6.7.6 Vermeidung doppelter Einträge 39
6.7.7 Anzeige von aussagekräftigen Namen 40
6.8 Fazit 40

7 Zusammenfassung und Ausblick **43**

Abbildungsverzeichnis

3.1 Architektur von Solr. Quelle: [Kar14] 6
3.2 Ablauf einer Indizierung, bzw. Suche. Quelle: [Kar14] 9

6.1 AdminUI - Indizierung 23
6.2 AdminUI - Parameter für die Indizierung 24
6.3 AdminUI - Datenbank 28
6.4 AdminUI - Import der Datenbank 29
6.5 AdminUI - Ergebnisse des Datenbankimports 30
6.6 AdminUI - Suche 34
6.7 AdminUI - Highlighting 36
6.8 AdminUI - Response Header 37

Tabellenverzeichnis

2.1 Related Work 4

3.1 Architektur - Hauptkomponenten 7
3.2 Architektur - Verarbeitungseinheiten 8
3.3 Architektur - Anwendungsschicht 8
3.4 Architektur - Lucene Core 9

4.1 GET-Parameter der Schema API, Quelle: [Foua] 12
4.2 PUT-Parameter der Schema API, Quelle: [Foua] 12
4.3 Parameter einer Felddefinition, Quelle: [Foua] 13
4.4 Parameter des QueryParsers, Quelle: [Foua] 15
4.5 Parameter des Standard QueryParser, Quelle: [Foua] 15
4.6 Parameter des DisMax QueryParsers, Quelle: [Foua] 15
4.7 Parameter des eDisMax Query Parsers, Quelle: [Foua] 16
4.8 Parameter für die Indizierung von CSV-Dateien, Quelle: [Foua] 17
4.9 Parameter für Apache Tika, Quelle: [Foua] 18
4.10 Befehle für die Indizierung von Datenbanken, Quelle: [Foua] 18
4.11 Parameter für einen full-import, Quelle: [Foua] 19

6.1 Optionen des SimplePostTools, Quelle: [Foua] 25
6.2 Wildcards, Quelle: [Foua] 35
6.3 Anforderungsauswertung 42

Zu einem guten Ende gehört auch ein guter Beginn.

Konfuzius, (551 - 479 v. Chr.)

1 Einleitung

In diesem Abschnitt wird die Motivation für die Verwendung eines Servers für die Volltextsuche, insbesondere Solr, dargestellt und die Ziele, sowie der Aufbau der Arbeit betrachtet.

1.1 Motivation

Eine heute wesentliche Funktion für nahezu alle Websites ist die Volltextsuche. Besonders bei großen Websites ist es dabei wichtig, aus großen Datenmengen schnell die gesuchten Informationen zu erhalten. Dazu ist ein leistungsfähiges Programm zur Volltextsuche nötig.
Für eine effiziente Volltextsuche ist eine effiziente Volltextindizierung nötig. *Solr* [Foua] setzt dabei auf *Apache Lucene* [Foub]. Dieses ist ein bewährtes Programm zur Volltextindizierung. Die erste Version wurde 1999 entwickelt und aktuell gibt es Lucene in der Version 4.10.3. Auch Solr existiert schon über 10 Jahre. Es wurde 2004 entwickelt und die aktuelle Version ist 4.10.3. Weiter ist Solr Open Source [Foua] und komplett in Java geschrieben, was eine gute Portabilität zur Folge hat. Außerdem bietet Solr auch eine große Funktionalität, wie zum Beispiel Rechtschreibkorrektur, Tokenizer [Foua], etc. Des Weiteren bietet Solr auch APIs in verschiedenen Programmiersprachen und unterstützt Caching und den Einsatz in der Cloud. Außerdem bietet Solr noch ein grafisches Administrationsinterface, über das aktuelle Konfigurationen abgefragt, Dateien indiziert und Anfragen gestellt werden können.

1.2 Ziel der Arbeit

Ziel dieser Arbeit ist es, die Funktionsweise und den Funktionsumfang von Solr zu betrachten. Es wird untersucht, welche Funktionen Solr unterstützt und welche nicht. Bei Funktionen, die Solr unterstützt, werden die Konfigurationen, die für die Verwendung dieser nötig sind, betrachtet. Außerdem wird die Verwendung der Funktionen, zum Beispiel über die API, betrachtet. Bei Funktionen, die Solr nicht unterstützt, wird untersucht, ob der Benutzer diese über die API in einem externen Programm selbst realisieren kann. Dabei werden Mechanismen von Solr betrachtet, die die Realisierung der Funktionen aus einem externen Programm unterstützen können.

1.3 Aufbau der Arbeit

Dieses Kapitel umfasst eine Motivation für die Verwendung von Servern für die Volltextsuche und speziell von Solr. Ebenso sind die Ziele der Arbeit sowie der Aufbau der Arbeit enthalten.
In Kapitel 2 wird der Funktionsumfang einiger anderer Open Source Server für die Volltextsuche betrachtet.
Kapitel 3 beschreibt die Architektur von Solr. Dabei werden die Funktionen der einzelnen Komponenten, sowie das Zusammenspiel der einzelnen Komponenten beschrieben.
In Kapitel 4 wird die API von Solr betrachtet. Solr hat eine REST-API für JSON und XML. Diese wird in drei Arten unterschieden. Diese sind die Schema API, die CoreAdmin API und die API zur Suche und Indizierung, die im Folgenden als "Standard-API" bezeichnet wird. Die Schema API, dient der Konfiguration von Solr aus anderen Programmen. Die CoreAdmin API wird verwendet, wenn Solr als Cloud betrieben wird. Die "Standard-API" dient der Suche und der Indizierung. Dafür stehen außerdem ClientAPIs für verschiedene Programmiersprachen zur Verfügung. Es gibt auch noch einen *EmbeddedSolrServer* [Foua] für Java, der keine HTTP-Verbindung benötigt. Dieser ist jedoch in seinem Funktionsumfang sehr beschränkt und sollte deshalb nur für Testzwecke verwendet werden. Dabei wird auf der Funktionsumfang der APIs, sowie die Verwendung dieser Funktionen, aus einem Anwendungsprogramm untersucht.
In Kapitel 5 werden die Anforderungen an das System formuliert, die dann in Kapitel 6 untersucht werden.
Diese sind die Indizierung verschiedener Dateiformate, eine täglich neue Indizierung der Datenbasis, die Einschränkung der Suchergebnisse, sowie eine übersichtliche und plausible Ausgabe der Suchergebnisse mit allen gewünschten Zusatzinformationen, wie zum Beispiel der Suchgeschwindigkeit. Dabei wird untersucht, inwieweit die Szenarien mit Solr realisierbar sind und herausgearbeitet, welche Schritte nötig sind, um die Szenarien zu realisieren.
Kapitel 7 beinhaltet eine Zusammenfassung und einen Ausblick.

2 Verwandte Arbeiten

Im Folgenden werden einige weitere Open Source Server für die Volltextsuche betrachtet, Sphinx [Incb], OpenSearchServer [INCa] und Elasticsearch [Ela].
Dabei werden in Tabelle 2.1 die wesentlichen Merkmale mit Solr verglichen.

Merkmal	**Solr 4.10.2** [Foua]	**Sphinx 2.2.6** [Incb]	**OpenSearch-Server 1.5.8** [INCa]	**Elasticsearch 1.4.2** [Ela]
Kompatibilität	Linux, OS X und Windows	Windows 2000 und neuer, FreeBSD 4.x, 5.x, 6.x, 7.x, 8.x, NetBSD 1.6, 3.0, Solaris 9, 11, Mac OS X und nahezu alle Linux-Distributionen	Debian, Linux, Mac OS X und Windows	auf allen Systemen, auf denen Java 8 update 20 oder Java 7 update 55 läuft
Programmier-sprache	Java	C++	Java	Java
Programm-bibliothek	Apache Lucene	Sphinx Search Library	Apache Lucene	Apache Lucene
RESTAPI	JSON, XML, CSV	keine	JSON	JSON

ClientAPIs	Ruby, Rails, PHP, Java, Python, Perl, Forrest, C#, ColdFusion, .NET, Ajax	PHP, Python, Java	PHP, Ruby, NodeJS, Perl	PHP, Ruby, Perl, Python, .NET, Java, Javascript, Groovy, Rivers
Integrationen in andere Programme	ColdFusion, Typo3, Django, Drupal, WordPress, OpenCMS und weitere	Laravel, Monitoring, Drupal, Joomla, WordPress und weitere	Drupal, WordPress	Drupal, Django, WordPress, CouchBase und weitere
Document-Parsing	alle gängigen Formate [Fouc]	nein	MS Office, OpenOffice, HTML/XHTMK, XML, PDF, RTF, TXT, MP3/4, WAV und weitere	alle gängigen Formate [Fouc]
Datenbanken	alle JDBC-Datenbanken, xml-Datenquellen	MySQL, MS SQL, Oracle, sowie xml-Datenquellen (extra APIs SphinxQL und SphinxSE)	alle JDBC-Datenbanken, wie Oracle, MySQL und Microsoft SQL Server	alle JDBC-Datenbanken
Ausgabe-formate	CSV, JSON, PHP, PHPS, Python, Ruby, Velocity, XML, XSLT	JSON, XML	XML	XML, JSON
grafisches Interface	ja	nein	ja	nur über Plugins
Highlighting	ja	ja	ja	ja
Boosting	ja	ja	ja	ja
Filter	ja	ja	ja	ja
laufende Beispiele	eHarmony, DuckDuckGo, Sears, StubHub!, Zappos, BestBuy, Aol	Infegy, Boardreader, Craiglist.org, Avito.ru, BBC Archive Development, tumblr, Gutefrage.net, The Pirate Bay		Quizlet, The Guardian, Xing, stumbleupon, fog creek, github, soundcloud, foursquare

Tabelle 2.1: Related Work

Die vier Server zur Volltextsuche unterscheiden sich in der Kompatibilität kaum. Alle vier Programme laufen unter den gängigen Systemen, Linux, Windows und OS X. Auch Suchtools, wie Highlighting [Foua], Boosting [Foua] und Filter [Foua] bieten alle Vier.
Sphinx ist der einzige Server, der nicht auf der Programmbibliothek *Apache Lucene* basiert. Des Weiteren ist Sphinx eher auf Datenbanken ausgerichtet. Es bietet extra APIs für Datenbanken, unterstützt aber kein *Document Parsing*. Weiter bietet Sphinx auch kein grafisches Administrationsinterface.
Die anderen drei Programme sind sich von den Merkmalen ähnlich. Solr bietet dabei vorallem bei den Ausgabeformaten und der REST-API größere Flexibilität. Zum Beispiel bietet Solr die Ausgabeformate JSON, XML, PHP, CSV und weitere, wohingegen der OpenSearchServer nur XML und Elasticsearch nur XML und JSON unterstützt. Eine REST-API bietet Solr für JSON, XML und CSV, die anderen beiden bieten dagegen nur eine REST-API für JSON. Außerdem bietet Solr ein grafisches Administrationsinterface. Der OpenSearchServer besitzt ebenfalls ein grafisches Administrationsinterface, allerdings existieren für ihn wenig Integrationen in andere Programme und laufende Beispiele. Elasticsearch, Solr und Sphinx laufen unter anderem auf bekannten und großen Websites.

3 Architektur

Dieser Abschnitt betrachtet die Architektur von Solr. Dabei werden die einzelnen Komponenten, deren Funktionen und das Zusammenspiel der einzelnen Komponenten untereinander untersucht. Abbildung 3.1 zeigt eine Übersicht der Komponenten einer Solr Installation.

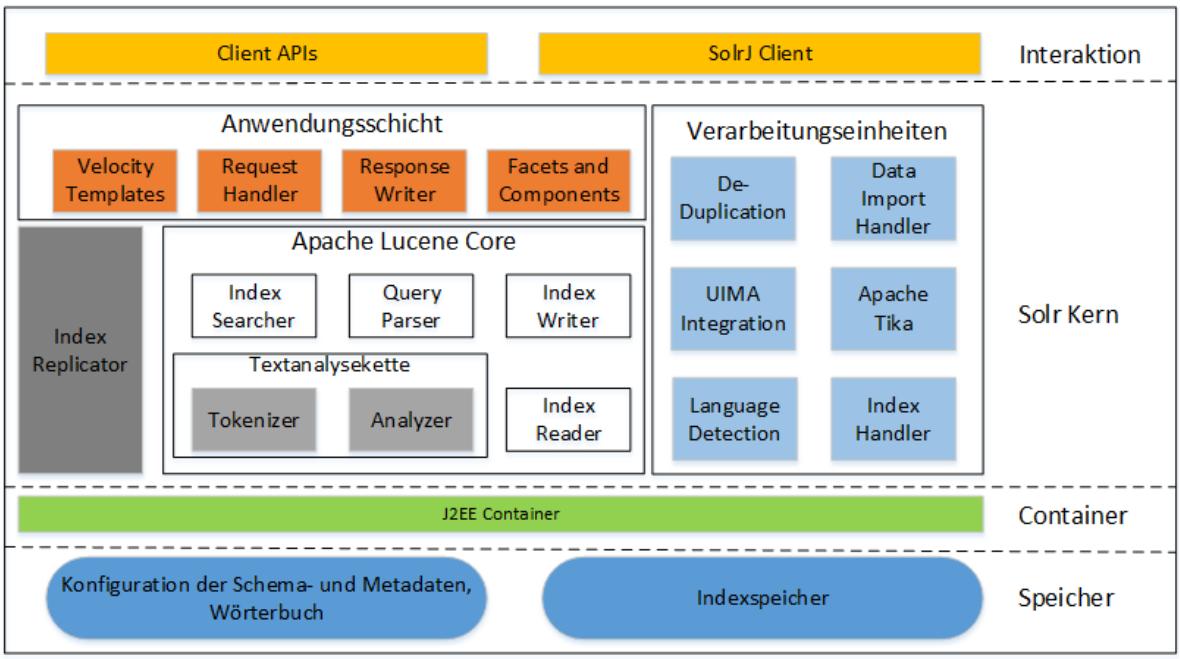

Abbildung 3.1: Architektur von Solr. Quelle: [Kar14]

Man kann die Architektur, wie in Abbildung 3.1 angedeutet, in vier Schichten aufteilen. Diese sind die Interaktion mit dem Benutzer oder Anwendungsprogrammen, die Solr-Anwendung, den *Servlet-Container* und der physische Speicher. Tabelle 3.1 beschreibt diese vier Schichten genauer.

Interaction	Interaktion mit Anwendungsprogrammen (siehe Abschnitt API)
SolrApplication	Kern von Solr lässt sich in die Processing Units, eine Anwendungsschicht, den integrierten Lucene Core [Foub] und einen Index Replicator aufteilen.
Container	Solr benötigt einen Servlet-Container. Dieser muss separat installiert werden. Unterstützt werden zum Beispiel Jetty, Tomcat, Resin. Eine kleine Version von Jetty ist bereits in Solr enthalten. Diese unterstützt jedoch nicht alle Funktionen.
Storage	Speicher auf der Platte, beinhaltet die Konfigurationsdateien, sowie die Indizes.

Tabelle 3.1: Architektur - Hauptkomponenten

Die Interaktion mit Anwendungsprogrammen wird in Abschnitt API betrachtet. Auf die Architektur des *Containers* und des Speichers wird in dieser Arbeit nicht eingegangen. Im Folgenden werden die einzelnen Komponenten Solr-Anwendung genauer betrachtet.

3.1 Verarbeitungseinheiten

Die Verarbeitungseinheiten sind die Komponenten, die die Indizierungsanfragen bearbeiten, bevor sie an den *Lucene Core* übergeben werden. Tabelle 3.2 zeigt die verschiedenen Verarbeitungseinheiten.

De-Duplication	Berechnet bei der Indizierung einen eindeutigen Hashwert, über den doppelte Indizes verhindert werden.
UIMA Integration	Mittels UIMA kann einem Dokument beim Indizieren Metainformationen hinzufügt werden.
Language Detection	Erkennt die Sprache des Dokuments vor der Indizierung und ermöglicht so eine effektivere Analyse. Es gibt zwei Implementierungen, die Tika Language Detection und LangDetect, wobei LangDetect mehr Sprachen unterstützt.
DataImport-Handler	Indiziert strukturierte Daten, wie zum Beispiel RSS und ATOM-Feeds, E-Mail Repositories oder Datenbanken. Dabei muss die Datenquelle in der Konfiguration festgelegt werden. Danach können die Befehle zur Indizierung der Daten wieder über *HTTP-POSTs* gesendet werden.

Apache Tika	Integriertes Plugin, das die Indizierung von vielen Dateiformaten, die Solr nicht direkt unterstützt, ermöglicht. Die Kommunikation erfolgt dabei ebenfalls über *HTTP-POSTs*. Tika erkennt den Dateityp der übergebenen Datei automatisch und produziert daraus eine XHTML-Datei. Diese wird dann an den *SAXContentHandler* übergeben, der *SAX-Events* an eine Schnittstelle sendet. Solr indiziert die Datei dann mittels dieser Events.
IndexHandler	Einfacher *RequestHandler*, der Dateien über einen *HTTP-POST* empfängt und zur Indizierung an den *IndexWriter* übergibt. Unterstützt werden XML, JSON und CSV. Des weiteren unterstützt der *IndexHandler* Updates und Löschoperationen auf den Indizes. Die Löschoperation ist dabei unabhängig vom Dateiformat.

Tabelle 3.2: Architektur - Verarbeitungseinheiten

3.2 Anwendungsschicht

Die Anwendungsschicht enthält Komponenten, die die Anfragen von der API entgegennehmen und die Antworten für die API bereitstellen. Außerdem sind in der Anwendungsschicht auch Komponenten für die Formatierung der Suchergebnisse enthalten. Tabelle 3.3 zeigt die verschiedenen Komponenten der Anwendungsschicht.

VelocityTemplate	Grafisches Interface, mit dem einfache Suchabfragen und deren Resultatausgabe simuliert werden können, ohne Solr in eine Anwendung integrieren zu müssen (Highlighting, Facettierung, Autovervollständigung und einige weitere Konzepte werden unterstützt)
RequestHandler	Komponente, die die Logik definiert, die für eine Abfrage ausgeführt wird. Es gibt verschiedene *Handler* für die Suche, Updates und weitere Aufgaben, wie Ausgabe der Systeminformationen oder Administration des Systems. Ein weiterer *Handler* ist der *RealTimeGetHandler*, der es ermöglicht, Dokumente in einer Suche auszugeben, ohne dass diese zuvor per *commit* ins Dateisystem geschrieben wurden. So muss nach der Indizierung nicht erst der Searcher gestartet werden, um nach dem Dokument zu suchen.
ResponseWriter	Liefert die Ergebnisse, die er vom IndexSearcher erhält im gewünschten Format zurück (Formate: CSV, JSON, PHP, PHPS, Python, Ruby, Velocity, XML ,XSLT)
Facets and Components	Beinhaltet Komponenten, die die Suchergebnisse formatieren. Diese sind unter anderem Facettierung, Highlighter, SpellChecker, MoreLikeThis.

Tabelle 3.3: Architektur - Anwendungsschicht

3.3 Lucene Core

Der *Lucene Core* ist hauptsächlich für die effiziente Speicherung der Indizes und eine schnelle Suche auf den Indizes verantwortlich. Tabelle 3.4 zeigt die Komponenten des *Lucene Core*.

IndexSearcher	Gibt die gefundenen Treffer nach einem berechneten Wert sortiert zurück
QueryParser	Parst die Suchabfragen der Benutzer, es werden mehrere Arten von Abfragen unterstützt, wie zum Beispiel Bereichsabfragen
IndexWriter	Schreibt Indizes in das Dateisystem
IndexReader	Liest die im Dateisystem gespeicherten Indizes
Analysatoren	Bestehen aus einer Klasse, oder einer Sequenz aus Tokenizern und Filtern
Tokenizer	Teilt einen Textstream anhand bestimmter Regeln in Tokens auf
Filter	Schaut sich jedes Token sequentiell an und entscheidet, ob es geändert wird, unverändert gelassen wird, oder entfernt wird

Tabelle 3.4: Architektur - Lucene Core

Abbildung 3.2 zeigt wie die Komponenten bei der Suche bzw. Indizierung zusammenarbeiten.

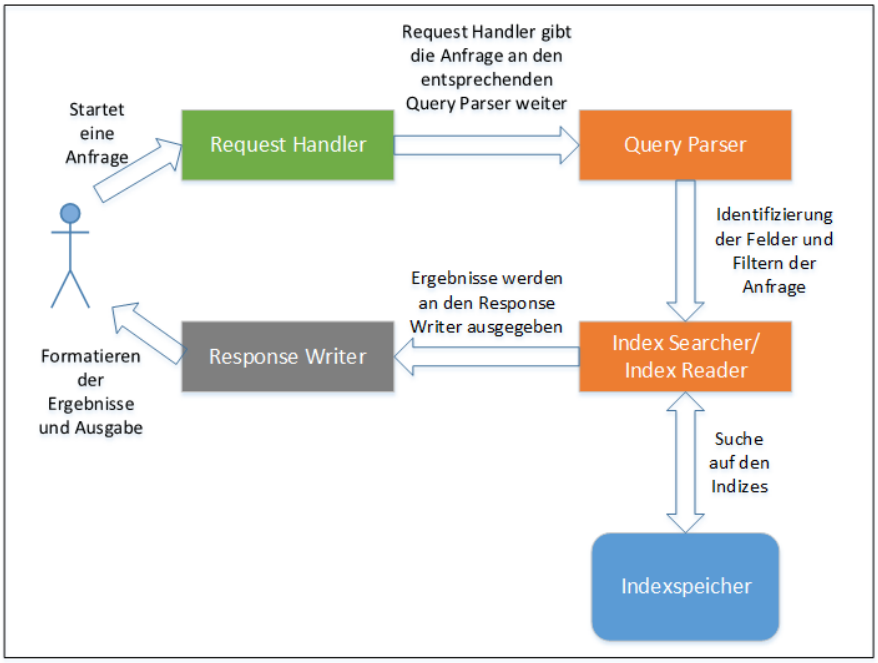

Abbildung 3.2: Ablauf einer Indizierung, bzw. Suche. Quelle: [Kar14]

3.4 IndexReplicator

Der *IndexReplicator* kopiert einen Index auf mehrere Server, um eine schnelle Antwortzeit auch bei hohem Abfragevolumen zu garantieren. Dabei werden ein Master und mehrere Clients definiert. Die Clients enthalten alle den selben Index. Ein Update des Index wird dabei vom Master an alle Clients weitergeleitet und eine Suchabfrage wird immer nur von einem der Clients bearbeitet.

3.5 Fazit

Die wesentliche Komponente der Architektur von Solr ist der *Lucene Core*, der für die Speicherung von Indizes und die Suche auf den Indizes zuständig ist. Die anderen Komponenten bauen darauf auf. Einige bieten zusätzliche Funktionen, wie zum Beispiel die *De-Duplication* oder Tika. Andere erleichtern die Bedienung des Programms, wie zum Beispiel die API oder der *Response Writer*. Solr ist nicht direkt auf dem Betriebssystem lauffähig, sondern benötigt einen *Container*.

4 API

Im Folgenden Abschnitt wird die API von Apache Solr betrachtet. Grundsätzlich bietet Solr REST APIs in XML und JSON. Dabei unterscheidet Apache drei Arten: Schema API, CoreAdmin API und die "Standard"-API. Die Schema API liefert Informationen über die aktuelle Konfiguration eines Kerns und ermöglicht es, einige Konfigurationen zu ändern. Die CoreAdmin API wird benutzt, um Informationen über alle Kerne einer Solr-Instanz zu erhalten und diese zu bearbeiten. Die "Standard"-API wird zur Indizierung und zur Suche verwendet. Für sie gibt es auch einige Client APIs in speziellen Programmiersprachen, die die Formulierung der HTTP-Requests vereinfachen.

4.1 Schema API

Bei der Schema API werden die Abfragen über HTTP-Request gesendet. Diese haben dabei immer das folgenden Grundformat:

http://<host>:<port>/<context-path>.

Danach folgen die *Path-Parameter*, diese bestehen aus dem Kern der angesprochen werden soll und dem Befehl, der auf dem Kern ausgeführt werden soll.
Um Informationen zu erhalten, wird ein *GET-Request* gesendet. Soll das Schema verändert werden, muss eine *POST-Request* gesendet werden. Tabelle 4.1 zeigt die verfügbaren *GET-Requests*.

/schema	Liefert das komplette Schema zurück
/schema/fields	Liefert Informationen zu allen definierten Feldern
/schema/fields/name	Liefert Informationen über das Feld mit diesem Namen
/schema/dynamicfields	Liefert Informationen über dynamische Felder

/schema/dynamicfields/name	Liefert Informationen über das dynamische Feld mit diesem Namen
/schema/fieldtypes	Liefert Informationen zu allen Feldtypen
/schema/fieldtypes/name	Liefert Informationen zu dem Feldtyp mit diesem Namen
/schema/copyfields	Liefert Informationen über "copyFields"
/schema/name	Liefert den Namen des Schemas
/schema/version	Liefert die Version des Schemas
/schema/uniquekey	Liefert den definierten Primärschlüssel
/schema/similarity	Liefert die globale "Similaritydefinition"
/schema/solrqueryparser/defaultoperator	Liefert den Standardoperator

Tabelle 4.1: GET-Parameter der Schema API, Quelle: [Foua]

Eine Beispielabfrage wäre:

`http://localhost:8080/solr/collection1/schema/fields?wt=json` [Foua].

Diese Abfrage liefert Informationen zu allen definierten Feldern des Kerns *collection1* im JSON-Format zurückliefern.
Es gibt zwei Rückgabeformate, JSON und XML. Das Standardrückgabeformat ist JSON, soll die Ausgabe im XML-Format erfolgen, muss ans Ende des HTTP-Request ?wt=xml hinzufügt werden. Bei der Abfrage des kompletten Schemas gibt es noch ein weiteres Rückgabeformat, wt=schema.xml. Dabei wird die Datei im Format der Datei schema.xml zurückgegeben.
Bei einer Konfigurationsänderung muss der Body des Requests die nötigen Informationen, zum Beispiel Felder, enthalten. Dabei wird im Body jedoch nur das JSON-Format akzeptiert.
Zur Konfigurationsänderung stehen die in Tabelle 4.2 aufgelisteten Befehle zur Verfügung:

/schema/fields	Felder hinzufügen
/schema/fields/name	Ein Feld mit diesem Namen hinzufügen
/schema/copyfields	CopyFields hinzufügen
/schema/managed/resource/paths	"Managed Resource Data" (Plugins) bearbeiten

Tabelle 4.2: PUT-Parameter der Schema API, Quelle: [Foua]

Der Befehl /schema/fields/name muss als *PUT-Request*, anstatt als *POST-Request* formuliert werden.
Eine Felddefinition muss den Namen, den Typ und den Defaultwert enthalten. Die Parameter dafür sind *name*, *type* und *default*. Zusätzlich können optional noch weitere Parameter hinzugefügt werden, die entweder *true* oder *false* gesetzt werden können (siehe Tabelle 4.3).

indexed	Wert kann für die Suche verwendet werden
stored	Wert kann abgerufen werden
docValues	Wert wird in "DocValues" abgelegt
sortMissingFirst / sortMissingLast	Sortiert das Dokument ein, wenn keine Sortierung bei der Suche eingegeben wird

multiValued	Dokument kann mehrere Werte für den Feldtyp enthalten
omitNorms	Normen werden abgeschaltet
omitTermFreqAndPositions	Schaltet Häufigkeit und Position von Nachrichten dieses Felds aus
omitPositions	Schaltet nur die Position aus
termVectors / *termPositions* / *termOffsets*	Solr behält die kompletten Terme, optional mit Position und Offset
required	Alle Felder dieses Typs müssen einen Wert haben

Tabelle 4.3: Parameter einer Felddefinition, Quelle: [Foua]

Listing 4.1 zeigt ein Beispiel mit dem cURL-Tool, das zwei neue Felder anlegt.

```
curl http://localhost:8080/solr/collection1/schema/fields -X POST -H
'Content-type:application/json' --data-binary '
[
  {
    "name":"sell-by",
    "type":"tdate",
    "stored":true
  },
  {
    "name":"catchall",
    "type":"text_general",
    "stored":false
  }
]'
```

Listing 4.1: Quelle: [Foua]

CopyFields haben eine *source*, eine *dest* und *maxChars*. Die Source ist der Name des Feldes, das kopiert werden soll, die Destination ist der Name der Kopie und die *maxChars* ist die Obergrenze an Buchstaben, die kopiert werden. *ManagedResources* stellt ein Mechanismus dar, der CRUD Operationen für Solr Plugins unterstützt.

4.2 Core Admin API

Die Core Admin API funktioniert ebenfalls über HTTP-Requests. Über sie können Informationen über mehrere Kerne einer Instanz abgerufen, Kerne erzeugt und einige weitere Operationen auf den Kernen ausgeführt werden. Diese sind STATUS, CREATE, RELOAD, RENAME, SWAP, UNLOAD, MERGEINDEXES, SPLIT und REQUESTSTATUS. Ein Beispiel wäre:
`http://localhost:8080/solr/admin/cores?action=STATUS&core=core0` [Foua].
Diese Abfrage gibt die Statusinformationen von *core0* aus.

4.3 "Standard"-API

Die "Standard"-API wird für die Benutzung von Solr aus einem anderen Programm genutzt. Dabei werden im Wesentlichen 5 Operationen unterstützt. Diese sind query, index, delete, commit und optimize. Die Kommunikation mit Solr erfolgt dabei über HTTP-Requests. Es können *GET*- und *POST-Requests* mit optionalen Parametern gesendet werden. Für die Rückgabe der Treffer gibt es verschieden *ResponseWriter*, die die Suchergebnisse in unterschiedlichen Formaten zurückgeben. Unterstützt werden die Formate CSV, JSON, PHP, PHPS, Python, Ruby, Velocity, XML und XSLT. Außerdem gibt es Client APIs für viele Programmiersprachen, die die Formulierung von HTTP-Requests vereinfachen. Diese gibt es für Ruby, Rails, PHP, Java, Python, Perl, Forrest/Cocoon, C#, ColdFusion, .NET und Ajax. Des Weiteren gibt es einen *EmbeddedSolrServer* [Foua], der eine Client API in Java bereitstellt, die keine HTTP-Verbindung benötigt. Dieser ist jedoch in seinem Funktionsumfang sehr beschränkt und sollte deshalb nur für Testzwecke verwendet werden.
Die Schnittstelle über HTTP wird über folgenden Befehl aufgerufen,
http://<host>:<port>/<context-path>/<kern>/<Abfrage mit Parametern>,
dabei ist der <context-path> meist solr und <kern> ist der Kern auf dem die Suchabfrage durchgeführt werden soll. Die Abfrage beginnt mit dem *RequestHandler*.

4.3.1 Suche

Der *RequestHandler* für die Suche, also die Operation *query*, ist *select*. Danach folgen in HTTP-Syntax die Suchparameter mit den jeweiligen Werten nach einem **?**. Die Suchparameter werden mittels **&** getrennt und können in beliebiger Reihenfolge eingegeben werden.
Es gibt drei *QueryParser*, die unterschiedliche Parameter unterstützten. Diese sind Standard, *DisMax* und *eDisMax* Parser. Tabelle 4.4 listet die Parameter, die von allen drei Parsern unterstützt werden, auf.

defType	Gibt an welcher Parser verwendet werden soll (Standard: dismax)
sort	Sortiert das Ergebnis entweder auf- oder absteigend
start	Gibt an ab dem wievielten Eintrag in den Suchergebnissen Solr den Inhalt anzeigen soll
rows	Gibt die Anzahl der Zeilen an, die angezeigt werden sollen

fq	Wendet einen Filter auf die Ergebnisse an
fl	Gibt an welche Felder angezeigt werden sollen
debug	Liefert zusätzliche Informationen in der Antwort (mögliche Werte sind *timing*, *results*, *query parameter*).
explainOther	Liefert Zusatzinformationen zu den Dokumenten
timeAllowed	Gibt die Zeit an, die eine Abfrage maximal dauern darf, eventuell wird nach Ablauf dieser Zeit ein partielles Ergebnis ausgegeben
omitHeader	Ergebnis wird ohne Header ausgegeben
wt	Gibt den *ResponseWriter* an, also das Ausgabeformat
logParamsList	Gibt an welche Parameter geloggt werden sollen, per Default alle

Tabelle 4.4: Parameter des QueryParsers, Quelle: [Foua]

Tabelle 4.5 zeigt die Parameter des Standard Query Parsers.

q	Der Suchbegriff
q.op	Standardoperator für Abfragen (OR oder AND)
df	Standardfeld

Tabelle 4.5: Parameter des Standard QueryParser, Quelle: [Foua]

Tabelle 4.6 zeigt die Parameter, die vom DisMax Parser unterstützt werden.

q	Der Suchbegriff
q.alt	Suche mit dem StandardQueryParser, wenn q nicht benutzt wird
qf	Definiert das Feld, auf dem die Suche ausgeführt wird
mm	Gibt die minimale Anzahl an Feldern an, die übereinstimmen müssen
pf	Der Score von Dokumenten, bei denen die Suchbegriffe in unmittelbarer Nähe sind, wird erhöht
ps	Gibt an, wann zwei Terme in unmittelbarer Nähe sind
tie	Wert um den Score mehrerer Treffer auszugleichen
bq	Gibt einen Faktor an, um den die Wichtigkeit eines Treffers erhöht wird
bf	Gibt die Funktion an, die bei dieser Erhöhung angewendet wird

Tabelle 4.6: Parameter des DisMax QueryParsers, Quelle: [Foua]

Der *eDisMax Parser* unterstützt alle Parameter des *DisMax Parser* und zusätzlich noch die Parameter, die in Tabelle 4.7 aufgelistet sind.

boost	Eine Liste von Suchbegriffen, deren Score mit dem Score der Abfrage multipliziert wird
lowercaseOperators	Gibt an, ob "and" und "or" wie "AND" und "OR" behandelt werden

ps	Ungenauigkeitsfaktor für pf und falls pf2 und pf3 nicht angegeben sind auch für pf2 und pf3
pf2	Liste von Wortpaaren und optionalen Gewichten
ps2	Ungenauigkeitsfaktor für pf2
pf3	Liste von Worttripeln und optionalen Gewichten
ps3	Ungenauigkeitsfaktor für pf3
stopwords	Gibt an, ob die StopFilterFactory verwendet werden soll
uf	Gibt an welche Felder der Endnutzer zur Suche verwenden darf, default sind alle

Tabelle 4.7: Parameter des eDisMax Query Parsers, Quelle: [Foua]

Die Antwort besteht aus mindestens zwei Sektionen. Dem *ResponseHeader*, der den Status, die Ausführungszeit und die Suchparameter enthält, und der eigentlichen Antwort, die die Treffer, nach den gewählten Parametern formatiert, enthält.

4.3.2 Indizierung

Die Indizierung funktioniert über den *IndexHandler*.
Dieser wird über http://<host>:<port>/<context-path>/update aufgerufen. Mit ihm können XML, JSON und CSV Dateien importiert werden. Andere Formate, wie pdf, Word, etc., müssen über das Plugin Apache Tika importiert werden. Dieses wird über

http://<host>:<port>/<context-path>/update/extract

aufgerufen. Strukturierte Daten werden über den *Data Import Handler* importiert.
Bei der Indizierung von XML gibt es 3 Tags. Das äußerste ist <add>, dessen Inhalt wird in den Index eingefügt. Das Zweite ist <doc>, es fügt die Felder zu einem Dokument zusammen. Das Dritte ist <field>, es beschreibt die Felder eines Dokuments. Bei JSON wird das <add> durch [] und das <doc> durch {} ersetzt und die Felddefinition wird als Paar im Format "name" : "Wert" angegeben.
Listing 4.2 zeigt eine Beispielabfrage mit cURL.

```
curl -X POST -H 'Content-Type: application/json'
'http://localhost:8080/solr/collection1/update' --data-binary '
[
  {
    "id": "1",
    "title": "Doc 1"
  },
  {
    "id": "2",
    "title": "Doc 2"
  }
]'
```

Listing 4.2: Quelle: [Foua]

Mit @<Dokumentenname> kann ein Dokument auch direkt indiziert werden (siehe Listing 4.3).

```
curl 'http://localhost:8080/solr/collection1/update?commit=true'
--data-binary @books.json -H 'Content-type:application/json'}
```

Listing 4.3: Quelle: [Foua]

Dabei muss sich *books.json* im aktuellen Arbeitsverzeichnis befinden.
Bei CSV muss die Datei direkt angegeben werden. Außerdem werden bei der Indizierung einer CSV-Datei die in Tabelle 4.8 aufgelisteten Parameter unterstützt.

seperator	Gibt das Trennsymbol an
trim	Entfernt die Leerzeichen am Anfang und Ende der Werte
header	Gibt an, ob die erste Zeile die Feldnamen enthält
field_names	Aufzählung der Feldnamen
literal.<field_names>	Aufzählung der Feldnamen für Literalwerte
skip	Aufzählung von Feldnamen, die übersprungen werden
skipLines	Anzahl der Zeilen, die übersprungen werden sollen
encapsulator	Zeichen, das optional verwendet wird um Werte mit csv-Trennzeichen zu schachteln
escape	Zeichen, das angibt dass das folgende Zeichen als Wert interpretiert wird
keepEmpty	Leere Felder mitindizieren
map	Werte zusammenfügen
split	Ein Feld in mehrere aufteilen
overwrite	Duplikate werden überschrieben
commit	Ein *commit* wird nach der Indizierung ausgeführt
commitWithin	Ein *commit* wird in angegebener Zeit ausgeführt
rowid	Zeilennummer kann einem Feld zugewiesen werden
rowidOffset	Offset wird auf Zeilennummer addiert

Tabelle 4.8: Parameter für die Indizierung von CSV-Dateien, Quelle: [Foua]

Bei der Indizierung mit dem *Tika Plugin* [Foud] wird die Datei im Body einfach über @ angegeben. Tika erkennt dann den Dateityp selbst. Tabelle 4.9 zeigt die Parameter, die von Tika unterstützt werden.

boost	Erhöht die Wichtigkeit des Feldes
capture	Erfasst XHTML-Elemente um Zusatzinformationen hinzuzufügen
captureAttr	Bezieht XHTML-Attribute in die Indizierung ein
commitWithin	Ein *commit* wird in angegebener Zeit ausgeführt
date.formats	Gibt das Datumsformat an
defaultField	Wenn Tika das aktuelle Feld nicht bestimmen kann, wird dieses Feld verwendet
extractOnly	Dokument wird nur extrahiert, nicht indiziert

extractFormat	Format zum Extrahieren, xml oder text
fmap.<source_field>	Feldnamen werden zusammengefügt
literal.<fieldname>	Wert wird für alle Dokumente in das Feld mit diesem Namen eingefügt
literalsOverride	Literalwerte überschreiben andere Werte des selben Feldes
lowernames	Alle Feldnamen werden in Kleinbuchstaben umgewandelt
multipartUploadLimitInKB	Limit für Größe eines Dokuments
passwordFile	Gibt den Pfad zur Passwortdatei an
ressource.name	Optionaler Name für die Datei (für MIME-Erkennung)
ressource.password	Passwort für passwortgeschützte Dateien
tika.config	Gibt den Pfad zur Konfigurationsdatei an
uprefix	Versieht alle nicht definierten Felder mit dem angegebenen Präfix
xpath	Beim extrahieren wird nur der XHTML-Inhalt zurückgegeben, der dem gegebenen xpath genügt

Tabelle 4.9: Parameter für Apache Tika, Quelle: [Foua]

Listing 4.4 zeigt ein Beispiel, indem die Datei *tutorial.html* mit dem Primärschlüssel *doc1* indiziert wird und anschließend ein *commit* durchgeführt wird.

```
curl 'http://localhost:8080/solr/update/extract?literal.id=doc1&commit=true' -F
"myfile=@tutorial.html"
```

Listing 4.4: Quelle: [Foua]

Für die Indizierung mit dem *Data Import Handler*, muss dieser zuerst in *solrconfig.xml* konfiguriert werden. Bei dieser Konfiguration wird auch die Datenquelle angegeben.
Aufgerufen werden die Befehle auf dieser Quelle dann über
http://<host>:<port>/<context-path>/<kern>/dataimport?command=<befehl>.
Dabei stehen für <*befehl*> verschiedene Befehle zur Verfügung (siehe Tabelle 4.10).

abort	Bricht eine laufende Operation ab
delta-import	Inkrementeller Import mit der Erkennung von Änderungen
full-import	In einem eigenen Thread werden alle Daten importiert
reload-config	Konfiguration wird neu geladen
status	Statistiken werden ausgegeben

Tabelle 4.10: Befehle für die Indizierung von Datenbanken, Quelle: [Foua]

Tabelle 4.11 zeigt die Parameter für einen *full-import*.

clean	Löscht den Index vor der Indizierung
commit	Führt einen *commit* nach der Indizierung durch
debug	Führt die Indizierung im Debuggingmodus durch, dabei muss *commit* explizit gesetzt werden
entity	Es können die Entitys angegeben werden, die ausgeführt werden sollen (Standard: alle)
optimize	Führt eine Optimierung des Indizes nach der Indizierung durch

Tabelle 4.11: Parameter für einen full-import, Quelle: [Foua]

4.3.3 Löschen

Löschvorgänge können mittels JSON und XML an Solr gesendet werden und löschen alle Indizes mit den angegebenen Eigenschaften, unabhängig vom Format. Bei XML können die Indizes über zwei Wege gelöscht werden, zum einen über den Primärschlüssel und zum anderen über eine Abfrage. Listing 4.5 zeigt einen beispielhaften Body eines Requests.

```
<delete>
  <id>3956</id>
  <id>4361</id>
  <query>title:solr</query>
  <query>publisher:timmohring</query>
</delete>
```

Listing 4.5

Bei JSON kann nur über den Primärschlüssel gelöscht werden und es kann jedem Löschvorgang eine Version zugewiesen werden (siehe Listing 4.6).

```
{
  "delete":"id":3956,
  "_version_":28456
}
```

Listing 4.6

4.3.4 Optimierung und Commit

Die Befehle *optimize* und *commit* können in der URL als Parameter, übergeben werden. Bei *optimize* verändert Solr die Indexstruktur derart, dass zukünftige Indizierungen und Suchvorgänge schneller ablaufen. Bei einem *commit* werden alle Daten seit dem letzten *commit* auf die Platte geschoben. Daten die noch nicht mittels *commit* auf die Platte geschrieben wurden, werden beim Suchen nicht gefunden. Ein *commit* ist nur nötig, wenn in den Einstellungen *autocommit* nicht aktiviert ist.

4.3.5 Client APIs

Client APIs erleichtern die Suche und Indizierung aus bestimmten Programmiersprachen. Zum Beispiel SolrJ für Java (siehe Listing 4.7).

```
// SolrServer initialisieren
String urlString = "http://localhost:8080/solr";
SolrServer solr = new HttpSolrServer(urlString);
// Java arbeitet normalerweise mit Binaries, umstellen auf xml
server.setParser(new XMLResponseParser());
// Query initialisieren und Parameter setzen
SolrQuery parameters = new SolrQuery();
parameters.set("q", mQueryString);
// Query ausfuehren
QueryResponse response = solr.query(parameters);
// Dokmunete der Antwort in Liste speichern
SolrDocumentList list = response.getResults();
```

Listing 4.7: Quelle: [Foua]

SolrJ enthält auch noch Methoden, mit denen die Resultatliste weiter bearbeitet werden kann. Listing 4.8 zeigt ein Beispiel einer Indizierung mit SolrJ.

```
// SolrServer initialisieren
String urlString = "http://localhost:8080/solr";
SolrServer solr = new HttpSolrServer(urlString);
// Dokument erstellen
SolrInputDocument document = new SolrInputDocument();
// Felder hinzufuegen
document.addField("id", "552199");
document.addField("name", "Gouda cheese wheel");
document.addField("price", "49.99");
// Datei indizieren
UpdateResponse response = solr.add(document);
// commit
solr.commit();
```

Listing 4.8: Quelle: [Foua]

Statt des *SolrServer* kann auch der *EmbeddedSolrServer* [Foua] verwendet werden, dieser kommuniziert dann mit einer Mikroinstanz von Solr, die direkt in der Javaanwendung läuft. Allerdings ist er in seiner Funktionalität sehr beschränkt und sollte vorwiegend zum Testen verwendet werden.

5
Anforderungsanalyse

In diesem Abschnitt werden die Anforderungen an das System, die in Kapitel 6 untersucht werden sollen, beschrieben. Es werden verschiedene Dateiformate indiziert, HTML, TYPO3, dynamisches HTML(PHP), PDF, Microsoft Word, Microsoft Excel, Microsoft Powerpoint, Text, RSS, XML, RTF und eine mysql-Datenbank. Die Datenbasis wird täglich neu indiziert, bzw. dynamisch an das Aktualisierungsintervall der Homepage angepasst. Weiter werden passwortgeschützte Bereiche separat indiziert. Suchen müssen bezüglich Datum, Dateiformaten, Suchbegriffen oder Gruppen eingeschränkt werden können. Dazu wird die Bildung von Suchgruppen benötigt. Suchergebnisse sollen plausibel sein, das heißt zum Beispiel bei leerem Volltextsuchfeld sollte nach anderen eingegebenen Kriterien gesucht werden, oder eine Fehlermeldung ausgegeben werden, dass das Volltextsuchfeld ausgefüllt werden muss. Verschiedene Sucharten werden benötigt. Die Suche nach allen Suchwörtern, der exakten Suchwortreihenfolge, einem der Suchworte oder keinem der Suchworte. Zuletzt sollen Ergebnisse in einer gewünschten Formatierung ausgegeben werden. Dabei sind gewünschte Formatierungseigenschaften das *Query Highlighting*, bei dem die Suchworte hervorgehoben werden, die Vermeidung doppelter Einträge, die Einbeziehung von Synonymen, die Sortierung nach der Relevanz der Einträge, die Anzeige der Suchgeschwindigkeit, die Anzahl der Ergebnisse pro Seite, sowie die Ausgabe aussagekräftiger Namen und Überschriften.

Zusammenfassung der Anforderungen:

- Indizierung verschiedener Dateiformate
- Tägliche/Dynamische Indizierung der Datenbasis
- Separate Indizierung eines passwortgeschützten Bereichs
- Einschränkung von Suchen
- Plausibilität von Suchen
- Suchvarianten
- Formatierung von Suchergebnissen

6 Realisierung mit Solr

In diesem Abschnitt wird getestet, inwiefern und auf welche Weise sich die Anforderungen umsetzten lassen. Dazu verwenden wir Solr 4.10.2 [Foua] unter Jetty 9.2.3 [Int] auf Debian 7.6. [Aok]. Es werden folgende Anforderungen untersucht:

- Indizierung verschiedener Dateiformate
- Tägliche/Dynamische Indizierung der Datenbasis
- Separate Indizierung eines passwortgeschützten Bereichs
- Einschränkung von Suchen
- Plausibilität von Suchen
- Suchvarianten
- Formatierung von Suchergebnissen

6.1 Indizierung verschiedener Dateiformate

Die Indizierung von Dateien läuft bei Solr über einen *HTTP-POST*. Dazu liefert die Solr Distribution zwei integrierte Möglichkeiten. Zum einen über das *AdminUI* und zum anderen über das *SimplePostTool*. Des Weiteren können die *POSTs* auch aus externen Programmen über die API an Solr gesendet werden. Für die Indizierung von Datenbanken muss zuerst die Konfiguration angepasst werden.

6.1.1 Indizierung über das AdminUI

Zuerst wird die Indizierung über das *AdminUI* betrachtet. Im *AdminUI* wird, wie Abbildung 6.1 zeigt, zuerst der Kern ausgewählt, in den die Dateien importiert werden sollen und dann Documents. Dort kann der Inhalt von XML, CSV und JSON-Dateien direkt eingegeben werden oder eine Datei vom Dateisystem ausgewählt werden, die indiziert werden soll.

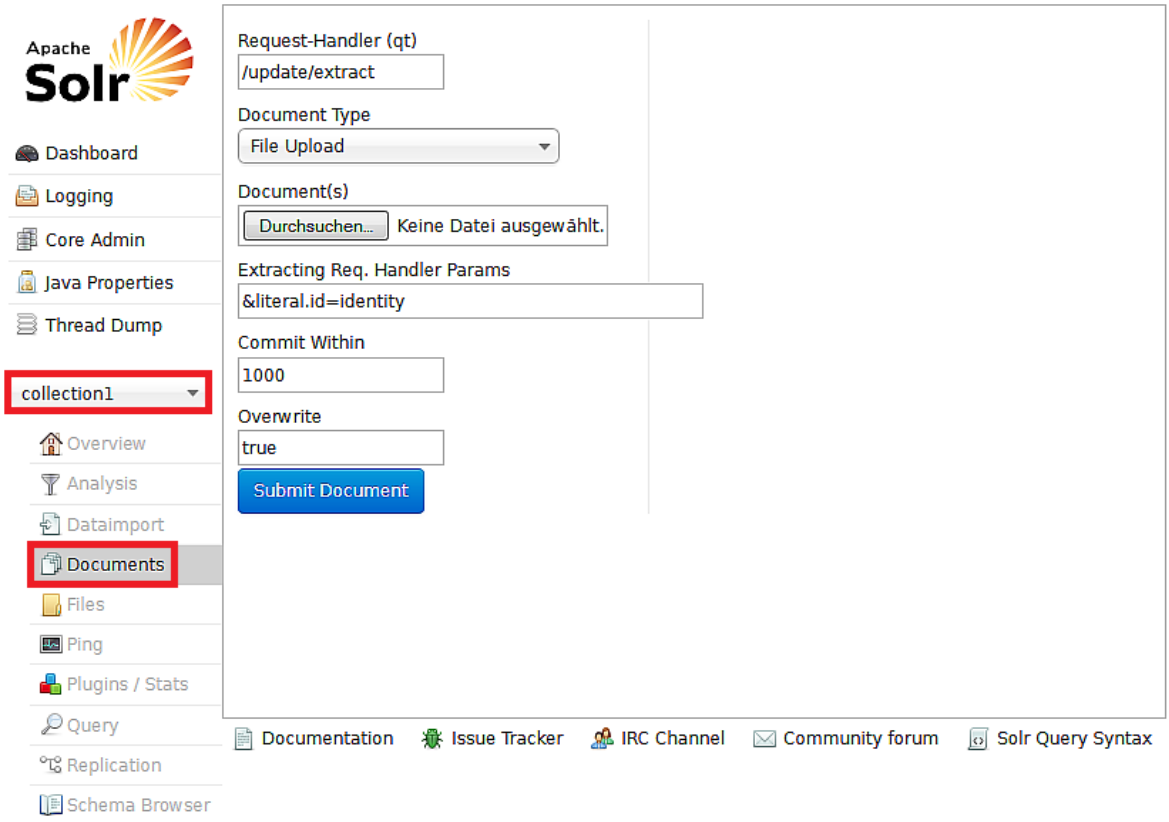

Abbildung 6.1: AdminUI - Indizierung

Zur Indizierung einer Datei wird als Document Type **"File Upload"** ausgewählt und als *Handler* Parameter eine ID eingegeben (siehe Abbildung 6.2). Diese dient zur eindeutigen Identifizierung des Indexes und wird bei der Durchführung von Updates verwendet. Als *Request-Handler* sollte */update/extract* angegeben werden, damit die Datei vor der Indizierung an das Tika Plugin übergeben wird, welches die Datei parst und dann an Solr übergibt. Ist *Overwrite* als *true* angegeben, wird ein Index mit der selben ID überschrieben, ansonsten wird das ausgewählte Dokument nicht indiziert, wenn ein Index mit der selben ID existiert.

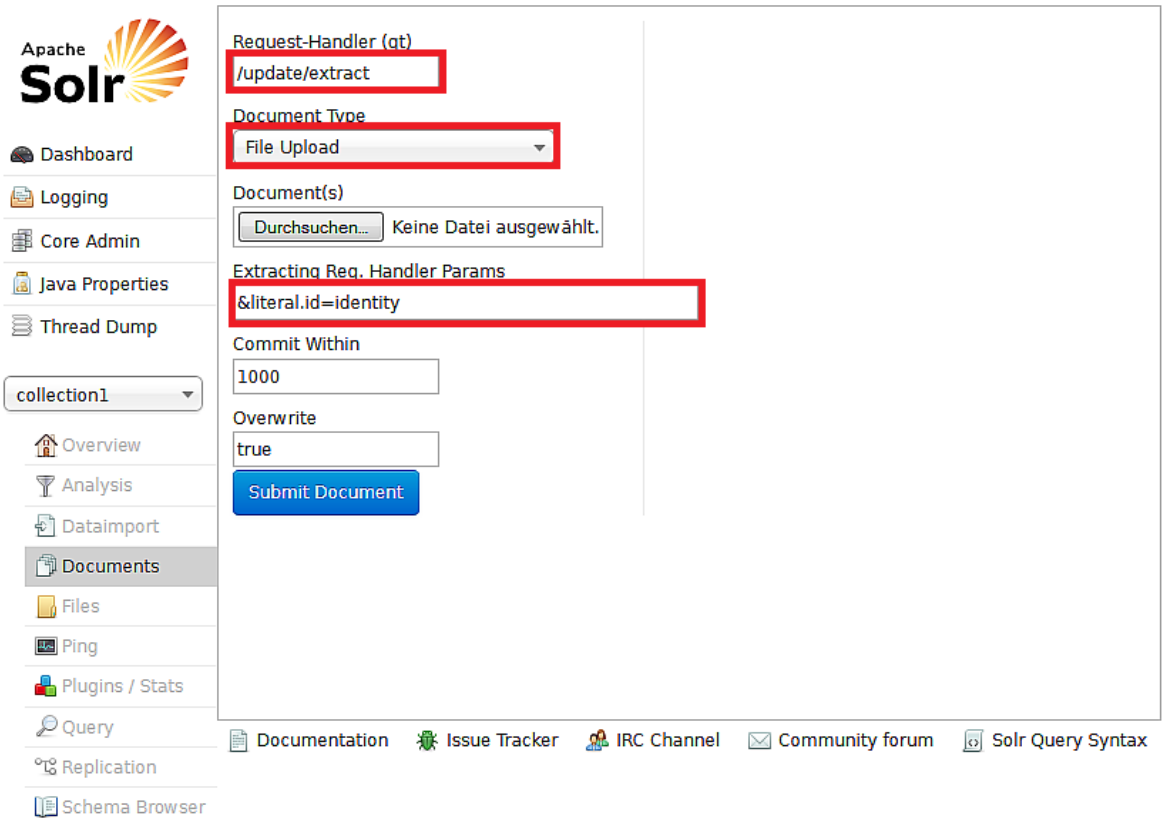

Abbildung 6.2: AdminUI - Parameter für die Indizierung

Die Indizierung über das *AdminUI* ist eher für Testzwecke geeignet, da jede Datei einzeln indiziert werden muss.

6.1.2 Indizierung über das SimplePostTool

Das *SimplePostTool* ist ein Kommandozeilenprogramm, dass die Dateien vom Dateisystem über einen *HTTP-POST* an Solr sendet. Es wird über die bei Solr mitgelieferte Datei *post.jar* aufgerufen. Ein Beispiel wäre **java -jar post.jar *.xml**. Dabei müssen sich die XML-Dateien, sowie die *post.jar* Datei im aktuellen Verzeichnis befinden. Es können aber auch Pfadausdrücke angegeben werden. Außerdem unterstützt das *SimplePostTool* verschiedene Optionen (siehe Tabelle 6.1)

Parameter	**Werte**	**Standardwert**	**Beschreibung**
-Ddata	args, stdin, files, web	files	Art der Übergabe, args = Argumente(z.B. löschen), stdin=Standardeingabe, files=Dateipfad, web=simpler Web-Crawler
-Dtype	<Dateityp>	application/xml	Gibt den Dateityp an, wenn -Dauto nicht angegeben ist
-Durl	<solr-update-url>	`http://localhost:8080/solr/update`	Die URL, an die der *POST* gesendet wird
-Dauto	yes, no	no	Dateityp wird automatisch anhand des Namenssuffixes bestimmt
-Drecursive	yes, no	no	Unterorder werden rekursiv ebenfalls indiziert
-Dfiletypes	<typ>[,<typ>,..]	xml, json, csv, pdf, doc, docx, ppt, pptx, xls, xlsx, odt, odp, ods, rtf, htm, html	Dateiformate, die berücksichtig werden sollen
-Dparams	"<Schlüssel>=<Wert> [&<Schlüssel>=<Wert>...]"	keine	Zusätzliche Parameter
-Dcommit	yes, no	yes	Ein *commit* wird nach der Indizierung ausgeführt
-Doptimize	yes, no	no	Optimierung nach der Indizierung
-Dout	yes, no	no	Antwort in Outputdatei schreiben

Tabelle 6.1: Optionen des SimplePostTools, Quelle: [Foua]

Wird keine ID angegeben, wird der Dateipfad als ID übergeben.

6.1.3 Indizierung über die API

Bei der Indizierung aus einem externen Programm muss dieses die Datei in einem *HTTP-POST* an Solr senden. Dieser muss dann das im Abschnitt API beschriebene Format haben. Listing 6.1 zeigt dies zum Beispiel für eine *JSON-Datei* und Listing 6.2 für eine *HTML-Datei*.

```
curl 'http://localhost:8080/solr/collection1/update?commit=true'
--data-binary @books.json -H 'Content-type:application/json'
```

Listing 6.1: Quelle: [Foua]

```
curl 'http://localhost:8080/solr/update/extract?literal.id=doc1&commit=true' -F
"myfile=@tutorial.html"
```

Listing 6.2: Quelle: [Foua]

Bei Verwendung von *update/extract* ist die ID eine Pflichtangabe, ansonsten wird die Indizierung der Datei abgewiesen. Auch bei dieser Methode kann immer nur ein Dokument pro *POST* indiziert werden.

6.1.4 Indizierung von Datenbanken

Bei der Indizierung von Datenbanken müssen vor der Indizierung zuerst einige Konfigurationen vorgenommen werden. Zuerst muss der *Request-Handler* in *solrconfig.xml* registriert werden (siehe Listing 6.3). Dabei gibt *config* den Pfad zur Konfigurationsdatei des *Request-Handlers* an, also in diesem Beispiel im selben Verzeichnis, wie *solrconfig.xml*.

```
<requestHandler name="/dataimport" class="solr.DataImportHandler">
  <lst name="defaults">
    <str name="config">data-config.xml</str>
  </lst>
</requestHandler>
```

Listing 6.3: Quelle: [DIH]

Außerdem muss der Pfad zur Java-Datei des Solr *DataImportHandlers*, wie in Listing 6.4, angegeben werden.

```
<lib dir="../../../dist/" regex="solr-dataimporthandler-.*\.jar" />
```

Listing 6.4: Quelle: [DIH]

Dann muss die Konfigurationsdatei für den *Request-Handler* erstellt werden. Diese muss unter dem in *config* registrierten Pfad abgespeichert werden. Listing 6.5 zeigt ein Beispiel für eine mysql-Datenbank.

```
<dataConfig>
  <dataSource type="JdbcDataSource"
              driver="com.mysql.jdbc.Driver"
              url="jdbc:mysql://localhost/test"
              user="testuser"
              password="test"/>
  <document>
    <entity name="id"
           query="select id,name,description from testtable">
    </entity>
  </document>
</dataConfig>
```

Listing 6.5: Quelle: [DIH]

Dabei muss die URL, der Benutzername und das Passwort der Datenbank eingegeben werden. Im *Entitytag* muss eine gültige Anfrage an die Datenbank stehen, außerdem müssen die Attribute als Felder in der *schema.xml* Datei existieren.
Dann muss der entsprechende JDBC-Treiber heruntergeladen werden und der Pfad der .jar Datei des Treibers in **solrconfig.xml** angegeben werden (siehe Listing 6.6).

```
<lib dir="../../../dist/" regex="mysql-connector-java-\d.*\.jar" />
```

Listing 6.6: Quelle: [DIH]

Dann gibt es zwei Möglichkeiten, die Datenbank zu indizieren, zum einen über das *AdminUI* und zum anderen über die API.

6.1.4.1 Indizierung über das AdminUI

Zur Indizierung über das *AdminUI* wird, wie Abbildung 6.3 zeigt, wieder zuerst der Kern ausgewählt und dann *Dataimport*.

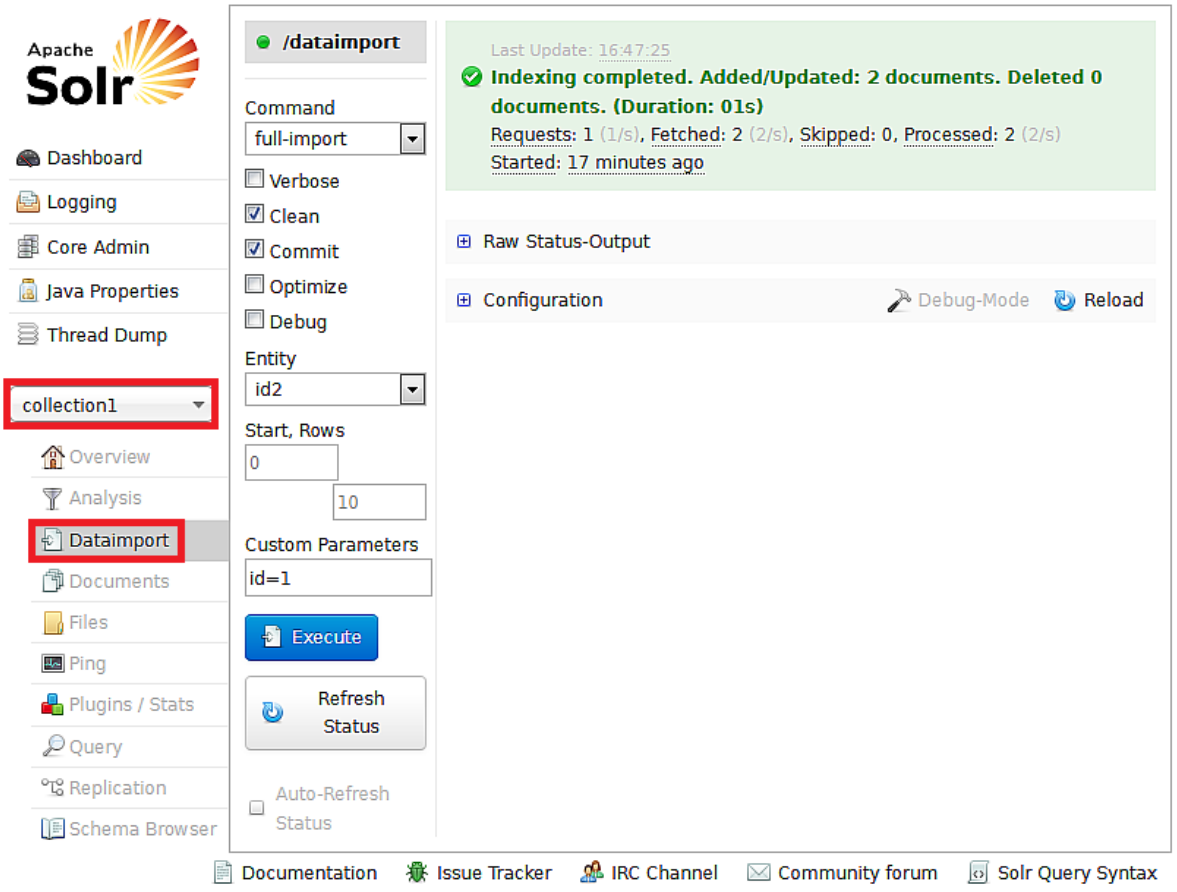

Abbildung 6.3: AdminUI - Datenbank

Dann gibt es, wie Abbildung 6.4 zeigt, in der linken Hälfte verschiedene Einstellungsmöglichkeiten. Es kann zwischen einem *full-import* oder *delta-import* gewählt werden. Beim *full-import* werden alle vorhandenen Indizes gelöscht und die Datenbank komplett neu indiziert, beim delta-import werden nur die geänderten Daten geladen. Weitere Optionen sind *Verbose* (es werden detaillierte Informationen über die Zwischenschritte der Indizierung ausgegeben), *Clean* (der Index wird vor der Indizierung gelöscht), *Commit* (führt einen *commit* nach der Indizierung durch), *Optimize* (optimiert den Index nach der Indizierung), *Debug* (führt die Indizierung im Debuggingmodus durch). Weiter kann das Entity gewählt werden, das ausgeführt werden soll. Es kann ausgewählt werden, ab welcher Zeile und wie viele Zeilen indiziert werden sollen und es können Parameter für die SQL-Query übergeben werden, zum Beispiel id=1. Diese können dann mittels **$dataimporter.request.id** in der Konfigurationsdatei des *Request-Handlers* abgerufen werden. Der Auto-Refresh Status kann ausgewählt werden. Wird dieser nicht ausgewählt, muss nach der Ausführung des Imports auf Refresh Status geklickt werden, ansonsten wird das Ergebnis nicht angezeigt, sondern der Status bleibt *Indexing*.

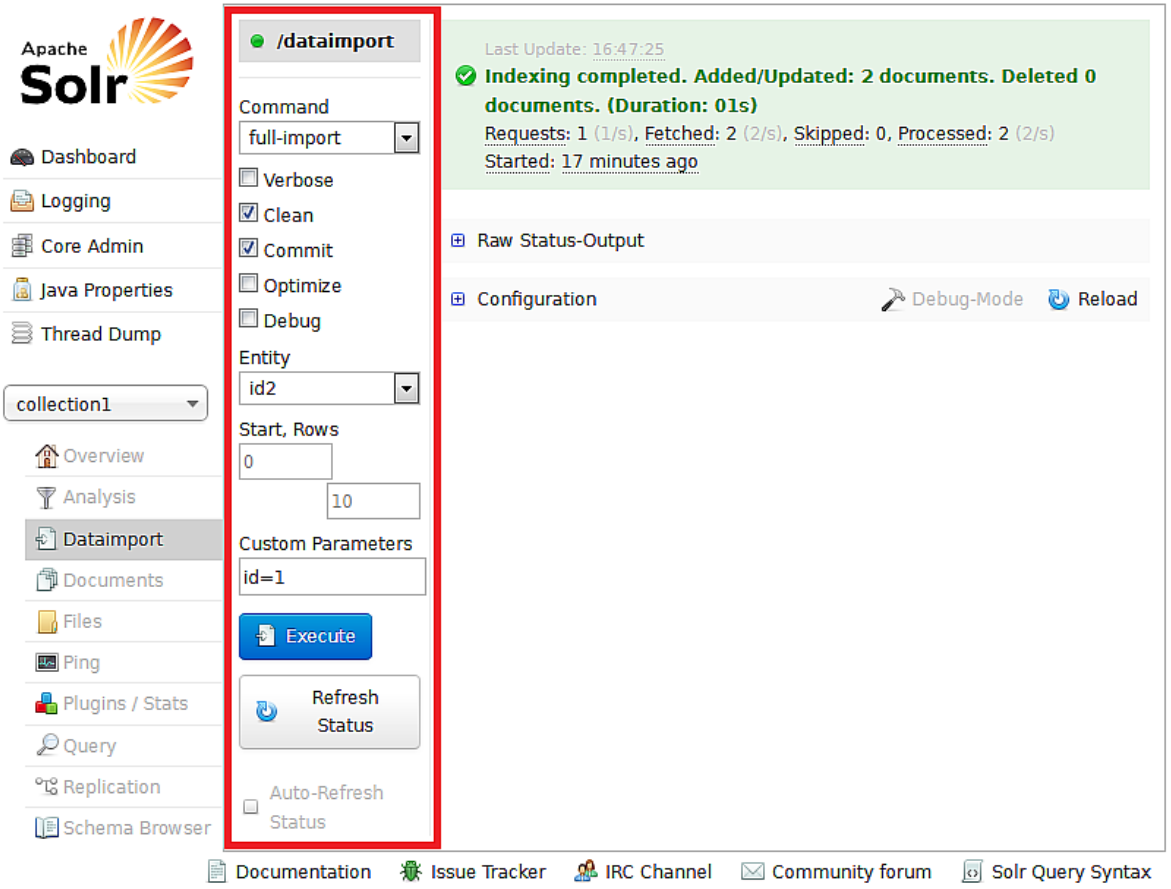

Abbildung 6.4: AdminUI - Import der Datenbank

Auf der rechten Seite können die Informationen über die Indizierung, Konfiguration und das Debugging abgerufen werden (siehe Abbildung 6.5).

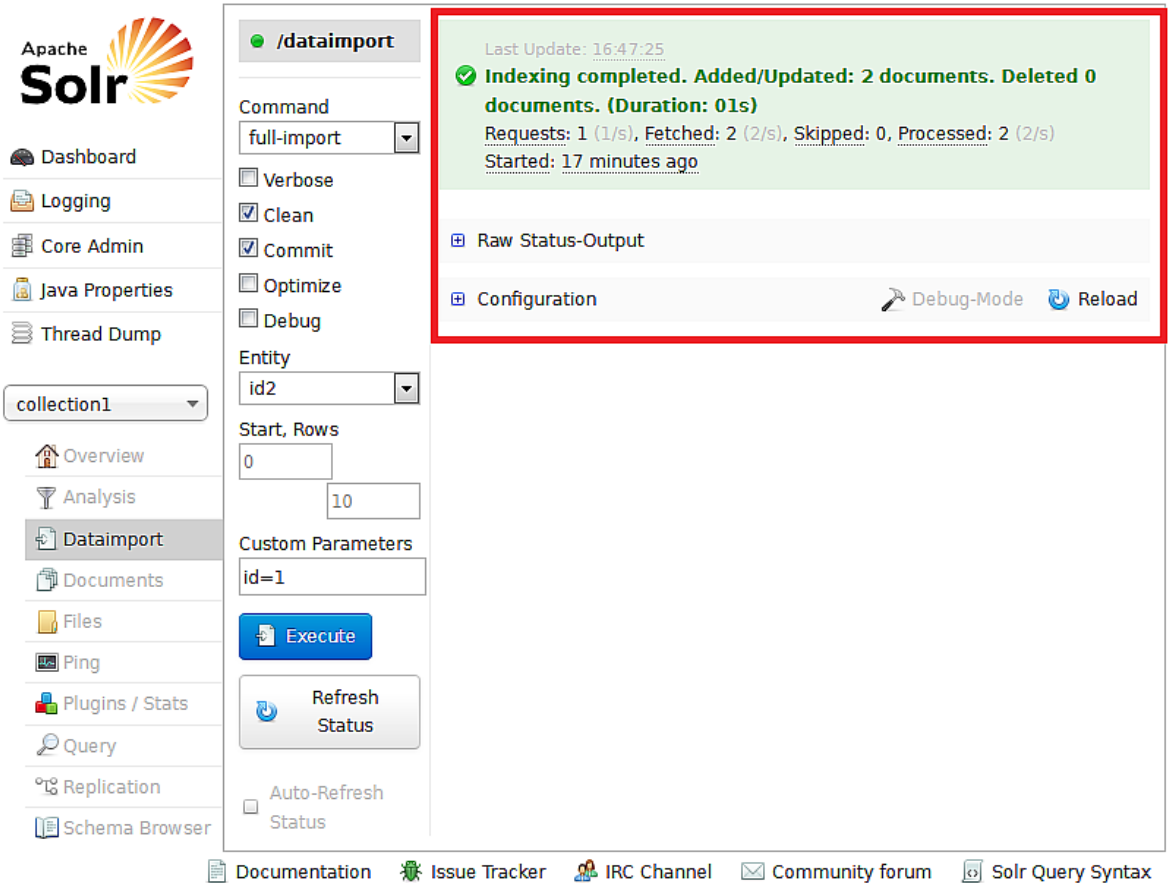

Abbildung 6.5: AdminUI - Ergebnisse des Datenbankimports

6.1.4.2 Indizierung über die API

Die Indizierung von Datenbanken über die API läuft wieder über einen HTTP-Request. Die verschiedenen Optionen zur Indizierung finden sich im Abschnitt API. Hier können im Gegensatz zum *AdminUI* mehrere Entitys auf einmal ausgeführt werden.

6.2 Tägliche/Dynamische Indizierung der Datenbasis

Solr bietet keine Möglichkeit, sich die Dateien selbst zu holen und zu indizieren. Stattdessen muss die tägliche Indizierung über das Anwendungsprogramm realisiert werden. Dabei gibt es zwei Möglichkeiten, ein Update durchzuführen. Die eine ist das Dokument nochmal zu indizieren. Dabei ist darauf zu achten, dass dieselbe ID, die bei der vorherigen Indizierung verwendet wurde, angegeben wird. Im Normalfall löscht Solr die Datei mit der selben ID dann und indiziert die neue Datei. Solr bietet jedoch auch eine weitere Strategie für Updates, die *atomic updates*. Bei *atomic updates* wird nur ein Feld des Dokuments verändert. Dabei werden die Operationen Wert ändern/setzen (**set**), Feld hinzufügen (**add**), Feld löschen (**remove**) und Wert inkrementieren (**inc**) unterstützt.
Listing 6.7, Listing 6.8 und Listing 6.9 zeigen ein Beispiel für ein *atomic update*. Dabei ist Listing 6.7 ein beispielhafter Ausgangsindex, Listing 6.8 ein mögliches *atomic update* und Listing 6.9 der daraus resultierende Index.

```
{
  "id":"Studium",
  "name":"Mohring Tim",
  "abschluss":"Bachelor",
  "hochschulsemester":9,
  "offenepruefungen":["Bachelorarbeit"]
}
```

Listing 6.7

```
{
  "id":"Studium",
  "abschluss":{"set":"Master"},
  "hochschulsemester":{"inc":1},
  "offene_pruefungen":{"remove":"Bachelorarbeit"},
  "akademischer_grad":{"add":"Bachelor"}
}
```

Listing 6.8

```
{
  "id":"Studium",
  "name":"Mohring Tim",
  "abschluss":"Master",
  "hochschulsemester":10,
  "akademischer_grad":["Bachelor"]
}
```

Listing 6.9

Ein weiteres Konzept, das Solr bei Updates unterstützt, ist *Optimistic Concurrency*. Es wird verwendet, damit Anwendungsprogramme feststellen können, ob das Dokument, das aktualisiert werden soll, bereits von einem anderen Programm verändert wurde. *Optimistic Concurrency* wird über ein Feld **_version_** realisiert. Dieses ist in *schema.xml* standardmäßig vorhanden und wird automatisch bei jeder Indizierung gesetzt. Die **_version_** kann dann bei einem

Update als Parameter angegeben werden und wird von Solr überprüft. Wenn die Version nicht den Anforderungen entspricht, wird das Update abgewiesen.
Dabei bedeutet eine **Version** > 1, dass die Versionsangaben genau übereinstimmen müssen, bei einer **Version** $= 1$, muss der Index für das Dokument nur existieren, die Versionsnummer wird nicht überprüft, bei einer **Version** < 0, darf kein Index für das Dokument existieren und bei einer **Version** $= 0$, wird überhaupt keine Überprüfung durchgeführt.
Dieses Konzept kann mit *atomic updates* und mit der Neuindizierung kombiniert werden.

6.3 Separate Indizierung eines passwortgeschützen Bereichs

Für die separate Indizierung von Dateien aus einem passwortgeschützten Bereich gibt es in Solr die Möglichkeit einen separaten Kern anzulegen, in dem diese Dateien indiziert werden. Bei der Indizierung dieser Dateien, wird dann über den Parameter *resource.password* das Passwort übergeben. Bei der Erzeugung eines neuen Kerns zum Beispiel über das *AdminUI* ist darauf zu achten, dass die *instanceDir* und *dataDir* Verzeichnisse, sowie die *schema.xml* und *solrconfig.xml* Dateien bereits vorher manuell erstellt werden müssen. Um den Zugriff auf den Kern zu beschränken, kann dieser mit einem Passwort geschützt werden. Dazu muss ein *security-constraint* und eine *login-config* in der **web.xml Datei vor** $</$**web-app**$>$ eingefügt werden (siehe Listing 6.10). Das *url-pattern* gibt an, welche URLs geschützt werden sollen. Der *realm-name* gibt den Namen des Realms [SEC] an. Der *role-name* gibt den Namen der Rolle an, die Zugriff auf das Realm erhalten soll.

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Solr authenticated application</web-resource-name>
    <url-pattern>/collection1/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>core1-role</role-name>
  </auth-constraint>
</security-constraint>

<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>Test Realm</realm-name>
</login-config>
```

Listing 6.10: Quelle: [SEC]

Des Weiteren muss man den Code aus Listing 6.11 in die Konfigurationsdatei des Servlet-Containers, also von jetty (jetty.xml) einfügen. Dabei wird in *config* der Pfad zur *properties-Datei* des Realms angegeben.

```
<Call name="addBean">
  <Arg>
    <New class="org.eclipse.jetty.security.HashLoginService">
      <Set name="name">Test Realm</Set>
      <Set name="config"><SystemProperty name="jetty.home"
          default="."/>/etc/realm.properties</Set>
      <Set name="refreshInterval">0</Set>
    </New>
  </Arg>
</Call>
```

Listing 6.11: Quelle: [SEC]

Die Datei (**realm.properties**) muss dann mit einem Namen, einem Passwort und dem Namen des Realms angelegt werden (siehe Listing 6.12).

```
name: password, core1-role
```

Listing 6.12: Quelle: [SEC]

Es wird ein Name, ein Passwort und die zuvor angegebene Rolle eingegeben.
Dann sind alle URLs, die mit

`localhost:8983/solr/collection1/` [Foua]

beginnen *passwortgeschützt*, also auch die *GET-Requests* an *collection1*.
Möchte man Solr nur über ein Suchfeld in seinem Anwendungsprogramm anbieten und nicht direkt über die Eingabe einer URL, dann kann man auch alle Seiten von Solr mit einem Passwort schützen. Dazu gibt man in der web.xml Datei einfach <**url-pattern**>/*</**url-pattern**> an.

6.4 Einschränkung von Suchen

Es wird die Einschränkung von Suchen bezüglich Datum, Dateiformat, Suchbegriffen und Gruppen betrachtet. Suchbegriffe werden bei Solr immer im *q-Parameter* angegeben. Einschränkungen werden dort direkt mitangegeben. Dabei gibt es zwei Möglichkeiten. Bei der ersten Variante **[wert1 TO wert2]** sind die zwei Werte *wert1* und *wert2* inklusive. Bei der anderen **{wert1 TO wert2}** sind die beiden Werte *wert1* und *wert2* sind nicht inklusive. Diese Varianten funktionieren mit Daten, Zahlen und *Strings*. Dabei muss das Datum als *String* mit folgendem Format **"2014-10-20T15:10:49Z"** angegeben werden. Außerdem werden für Daten die Angaben *NOW*, *YEAR* und *DATE* unterstützt. Bei Dateiformaten muss der MIME-Type als String angegeben werden und bei Suchbegriffen können *Strings* oder Zahlen angegeben werden. Passen die Typen nicht, werden keine Ergebnisse ausgegeben. Der *q-Parameter* ist dabei immer mit <**Feldname**>:<**Wert**> anzugeben. Gibt man keinen Feldnamen an, wird das definierte Standardfeld verwendet.
Um Einschränkungen bezüglich Gruppen anzugeben, müssen vorher Gruppen definiert werden.

Die Bildung von Gruppen wird über die Definition von Feldern realisiert. Dies können Felder sein, die in den Dateien enthalten sind, also von Tika bestimmt werden können, oder die bei der Indizierung explizit mit angegeben werden. Die Einschränkung bezüglich dieser Gruppen kann dann äquivalent zur Einschränkung von Suchbegriffen realisiert werden.

6.5 Plausibilität von Suchen

Als nächstes wird die Plausibilität von Suchen betrachtet. Dazu verwenden wir die Suchoption des *AdminUI*. Dieses ist für einen Kern unter Query zu finden (siehe Abbildung 6.6).
Dort werden auf der linken Seite die meisten Suchparameter von Solr aufgelistet und auf der rechten Seite wird die daraus generierte URL und darunter das Ergebnis der Suchanfrage angezeigt. Für Parameter, die im *AdminUI* nicht unterstützt werden, geben wir diese direkt über eine URL an.

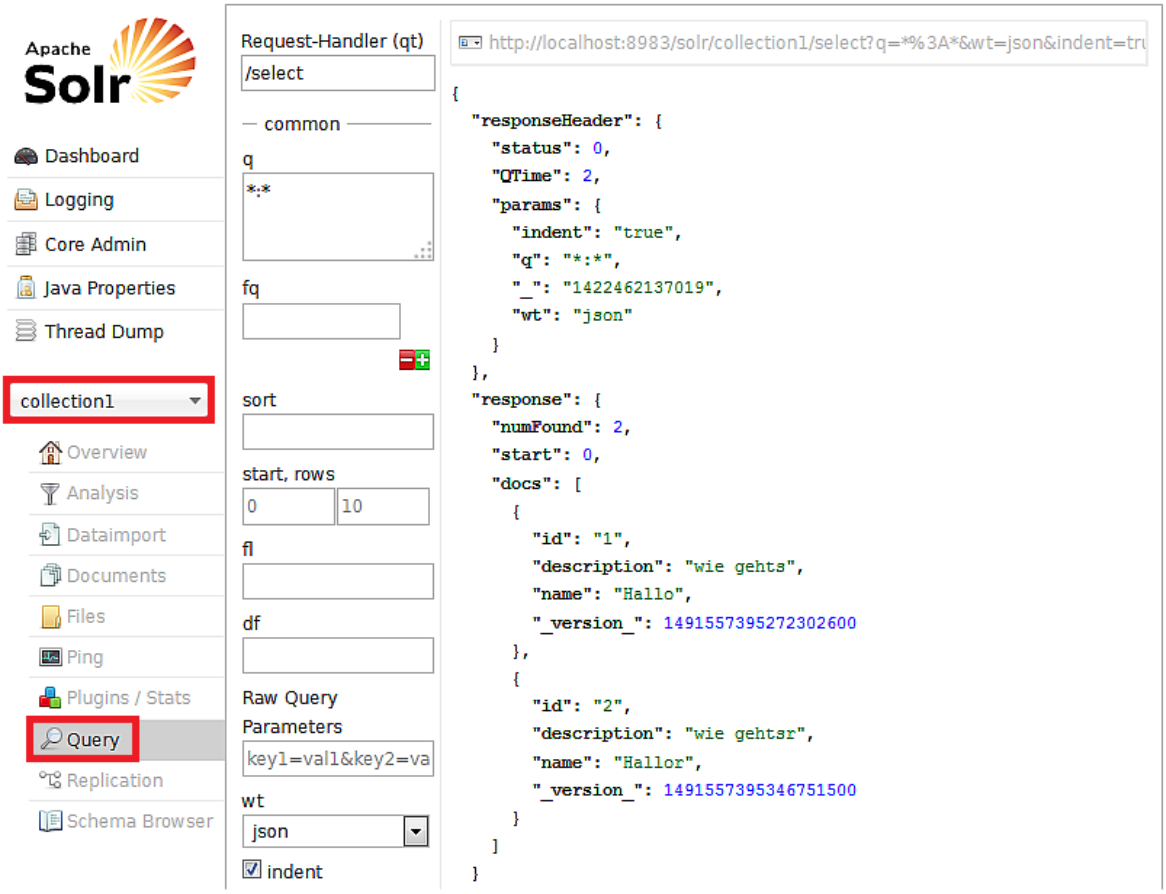

Abbildung 6.6: AdminUI - Suche

Alle Abfragen von Feldern laufen über den *q-Parameter*. Dabei wird eine Abfrage an ein bestimmtes Feld im *AdminUI* in folgendem Format formuliert **id:1**. Dies wird in der URL dann in folgendes Format geändert **id%3A**. Werden mehrere Felder angesprochen, werden diese im *AdminUI* über Komma oder Leerzeichen getrennt, was in der URL ein + ergibt. Dabei ist darauf

zu achten, dass in der *schema.xml-Datei* festgelegt ist, ob die einzelnen Suchbedingungen alle zutreffen müssen, oder nur eine. Wird kein Feld angegeben, so wird das definierte Standardfeld verwendet. Ist für den *q-Parameter* ein gültiger Ausdruck eingegeben und Ergebnisse zu dieser Anfrage existieren, liefert Solr diese Ergebnisse auch. Bei leerem oder fehlendem *q-Parameter*, wird ein Fehler, also eine *400er URL Response* gesendet.

6.6 Suchvarianten

Es werden die Suchen nach allen Suchwörtern, nach einem der Suchworte, nach keinem der Suchworte und nach der exakten Suchwortreihenfolge betrachtet.
Zu beachten ist, dass die Suchworte immer exakt mit den Feldwerten übereinstimmen müssen. Sucht man also nach *id:a*, wird ein Dokument, dessen ID *a* enthält, nicht ausgegeben. Sollen solche Dokumente ebenfalls ausgegeben werden, müssen Wildcards verwendet werden (siehe Tabelle 6.2).

?	Ein beliebiger Buchstabe
*****	Beliebig viele Buchstaben
∼	Fuzzy Search

Tabelle 6.2: Wildcards, Quelle: [Foua]

Beim *Fuzzy Search* [Foua] werden auch Werte gefunden, die ähnlich zum Suchbegriff sind. Diese Ähnlichkeit wird anhand des Damerau-Levenshtein Distance Algorithmus, von Fred J. Damerau [Dam64] und Vladimir I. Levenshtein [Lev66], bestimmt. Dabei kann nach ∼ die gewünschte Distanz für die zwei Algorithmen angegeben werden. Beispielsweise wird dann bei der Suche nach "roam" auch "foam" gefunden.
Für die Suche nach allen Suchwörtern und einem der Suchworte, gibt es zwei alternative Möglichkeiten. Eine Alternative für die Suche nach allen Suchwörtern und die Suche nach einem der Suchworte ist über die Angabe eines Parameters. Bei der Suche nach **allen Suchwörtern** muss **q.op=OR** angegeben werden und bei der Suche nach **einem der Suchwörter q.op=AND**. Eine weitere Möglichkeit ist im *AdminUI* zwischen zwei Suchworten ein **AND** oder ein **OR** einzufügen, also beispielsweise **id:1 OR id:2**. In der URL muss dafür **q=id%3A1 + OR + id%3A2** eingegeben werden. Für das **OR** kann in der URL auch **q=id%3A1 || id%3A2** verwendet werden.
Die Suche nach **keinem der Suchworte** kann im *AdminUI* über **NOT id:def** und in der URL entweder über **NOT + id%3A1** oder **!id%3A1** realisiert werden. Das NOT bezieht sich dabei nur auf den direkt folgenden Parameter. Wenn also die Suche nach keinem der Suchwörter realisiert werden soll, muss vor jedes Suchwort ein NOT gesetzt werden.
Soll nach der **exakten Wortreihenfolge** gesucht werden, müssen die Wörter in Hochkommatas gesetzt werden. Ein Beispiel wäre im *AdminUI* **id:"abc def"**, oder als URL **q=id%3A"abc+def"**.

6.7 Formatierung der Suchergebnisse

Es wird die Formatierung der Ausgabe betrachtet. Dies beinhaltet das *Query Highlighting*, die Ausgabe der Suchgeschwindigkeit, die Einschränkung der Anzahl der Ergebnisse, die Ausgabe von Synonymen, die Sortierung nach Relevanz, die Vermeidung doppelter Einträge und die Anzeige aussagekräftiger Namen.

6.7.1 Query Highlighting

Für das Query Highlighting, also das Hervorheben der Suchworte, muss der Parameter **hl=true** gesetzt werden, sowie beim Parameter **hl.fl** die Felder, in denen der Suchbegriff hervorgehoben werden soll, angegeben werden (siehe Abbildung 6.7). Ist **hl.fl=*** angegeben, so wird der Suchbegriff in allen Feldern hervorgehoben, in denen dies möglich ist.
Es kann außerdem **hl.requireFieldMatch=true** angegeben werden, dann wird der Begriff nur in den Feldern hervorgehoben, in denen die Query zutrifft. Außerdem kann ein Tag angegeben werden, das die hervorzuhebenden Elemente umschließt.

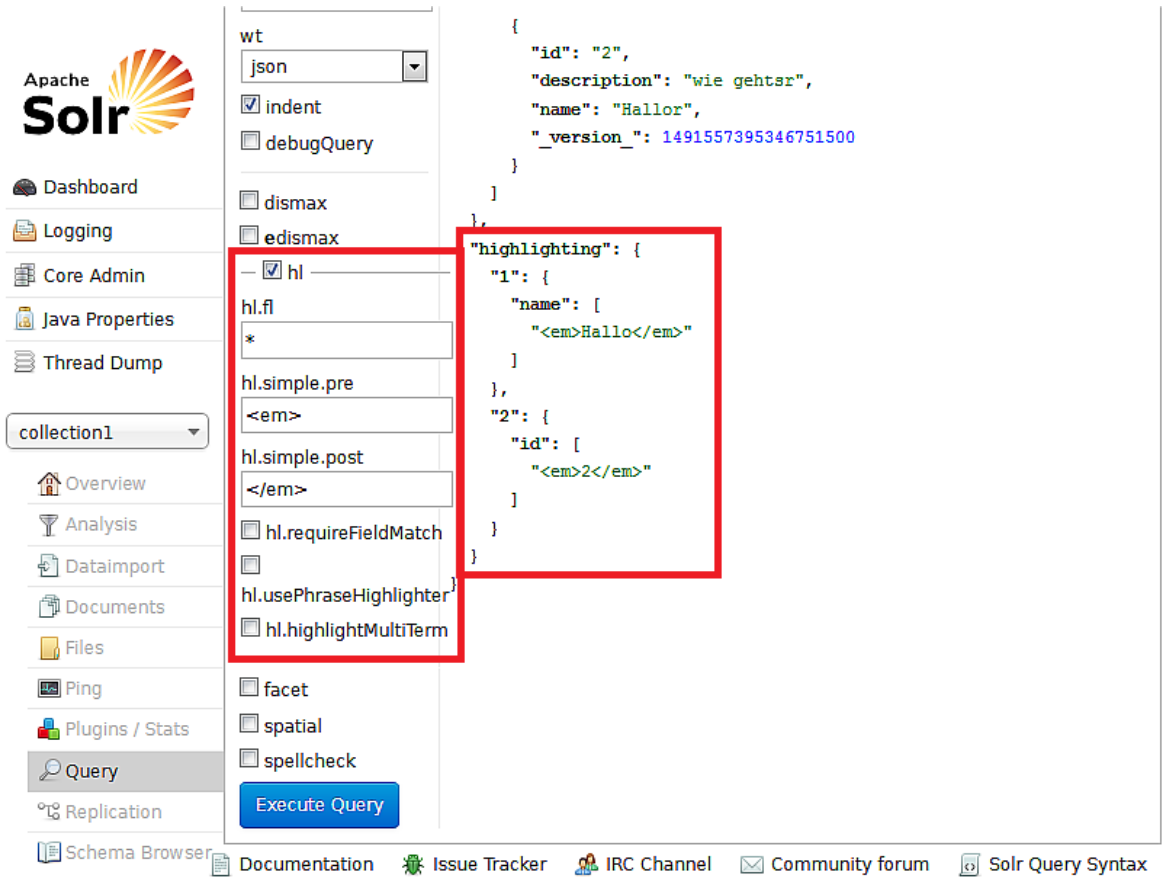

Abbildung 6.7: AdminUI - Highlighting

6.7.2 Ausgabe der Suchgeschwindigkeit

Die Suchgeschwindigkeit wird bei allen Ausgabeformaten, außer CSV im *ResponseHeader* im Parameter *QTime* mit ausgegeben (siehe Abbildung 6.8). Die Angabe der *QTime* ist in Millisekunden. Außerdem enthält der *ResponseHeader* auch noch die Parameter der Suchanfrage und einen *status*, der angibt, ob die Suchanfrage erfolgreich (**status=0**) ist.

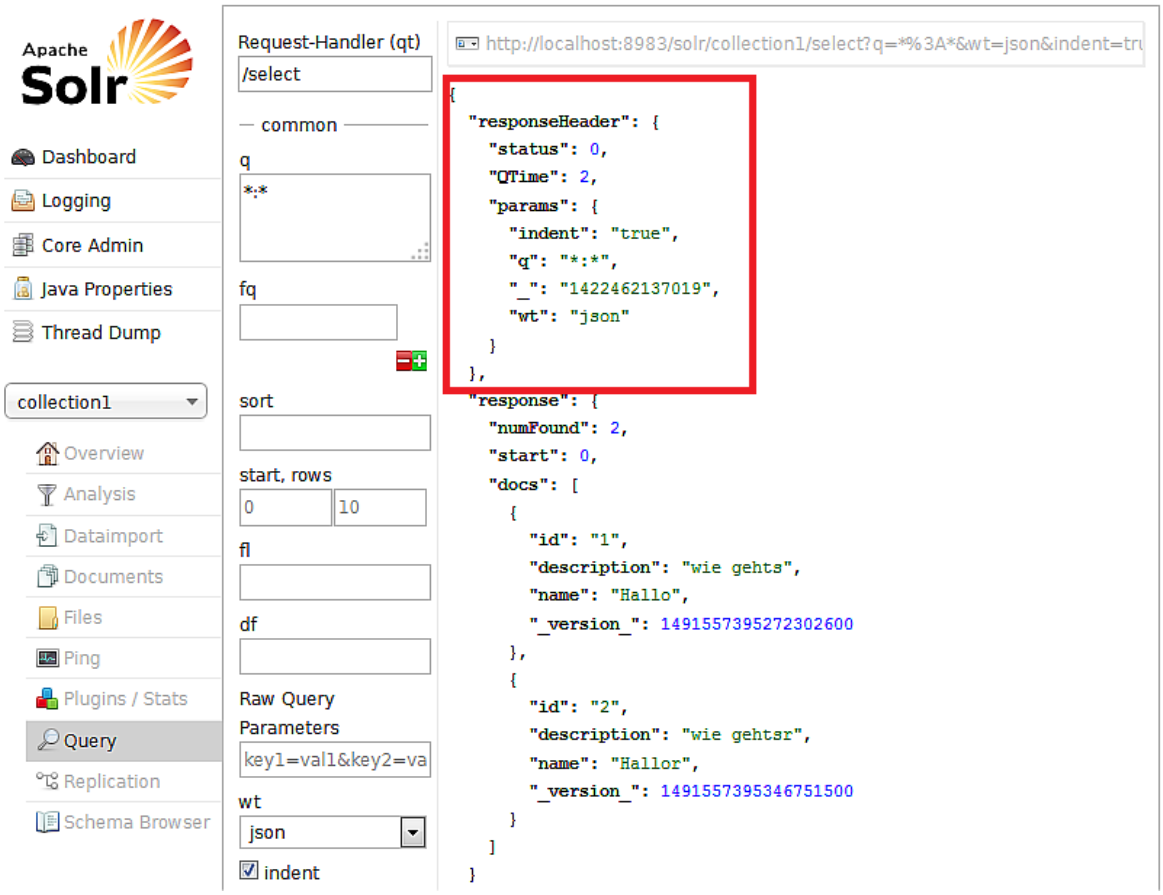

Abbildung 6.8: AdminUI - Response Header

6.7.3 Anzahl der Ergebnisse

Die Anzahl der Ergebnisse kann entweder über das Anwendungsprogramm geregelt werden, indem von Solr alle Ergebnisse der Suchanfrage angefragt werden und die für die Seite benötigten Ergebnisse herausgefiltert und angezeigt werden. Eine andere Möglichkeit ist, von Solr nur die Elemente für die aktuelle Seite liefern zu lassen. Dies wird über die Parameter *start* und *rows* gemacht. Bei *start* wird der *Startindex* angegeben, ab dem die Ergebnisse ausgegeben werden. Das erste Ergebnis hat dabei den Index 0. Mit *rows* wird die Anzahl der Ergebnisse angegeben, die ausgegeben werden sollen. Diese Methode hat den Vorteil, dass kleinere Datenpakete versendet werden und die Suchanfrage somit schneller bearbeitet werden kann. Allerdings muss für

jede weitere Seite bzw. der Erhöhung der Anzahl der Elemente pro Seite eine neue Suchanfrage gestellt werden.

6.7.4 Ausgabe von Synonymen

Um Synonyme bei der Suche zu unterstützen, muss ein Synonymfilter konfiguriert werden. Dabei muss in der Datei **schema.xml** ein spezieller Feldtyp eingefügt werden (siehe Listing 6.13). Dabei wird beim Attribut *synonyms* die Datei angegeben, in der die Synonyme definiert sind.

```
<fieldtype name="syn" class="solr.TextField">
  <analyzer>
    <tokenizer class="solr.WhitespaceTokenizerFactory"/>
    <filter class="solr.SynonymFilterFactory" synonyms="syn.txt" ignoreCase="true"
        expand="false"/>
  </analyzer>
</fieldtype>
```

Listing 6.13: Quelle: [ATT]

Listing 6.14 zeigt eine Beispieldatei für die Synonyme.

```
# blank lines and lines starting with pound are comments.

#Explicit mappings match any token sequence on the LHS of "=>"
#and replace with all alternatives on the RHS. These types of mappings
#ignore the expand parameter in the schema.
#Examples:
i-pod, i pod => ipod,
sea biscuit, sea biscit => seabiscuit

#Equivalent synonyms may be separated with commas and give
#no explicit mapping. In this case the mapping behavior will
#be taken from the expand parameter in the schema. This allows
#the same synonym file to be used in different synonym handling strategies.
#Examples:
ipod, i-pod, i pod
foozball , foosball
universe , cosmos

# If expand==true, "ipod, i-pod, i pod" is equivalent to the explicit mapping:
ipod, i-pod, i pod => ipod, i-pod, i pod
# If expand==false, "ipod, i-pod, i pod" is equivalent to the explicit mapping:
ipod, i-pod, i pod => ipod

#multiple synonym mapping entries are merged.
foo => foo bar
foo => baz
#is equivalent to
foo => foo bar, baz
```

Listing 6.14: Quelle: [ATT]

6.7.5 Sortierung nach Relevanz

Die sortierte Ausgabe nach der Relevanz ist bei Solr die Standardeinstellung. Dabei berechnet Solr einen *score* für die Suchanfragen. Dieser berechnet sich aus den Übereinstimmungen der einzelnen Suchwörter. Beispielsweise ist der *score* eines Ergebnisses, bei dem zwei mit OR verknüpfte Suchbegriffe zutreffen höher, als bei einem Ergebnis, bei dem nur eins zutrifft. Der Score einzelner Suchwörter kann auch über ^ um einen Multiplikator geändert werden, beispielsweise **id:a^2, content_type:application/pdf**. Dann wird der Score für das Dokument mit der ID a erhöht.
Es kann auch eine beliebige Sortierung angegeben werden. Dabei muss

sort=<feld>+<asc|desc>

angegeben werden. Soll nach mehreren Feldern sortiert werden, können diese im *AdminUI* mittels "," getrennt werden.

6.7.6 Vermeidung doppelter Einträge

Um doppelte Einträge zu verhindern, bietet Solr die *De-Duplication* an. Diese berechnet für jede Datei, die indiziert wird, einen Hashwert, über den die eindeutige Identifizierung der Datei möglich ist. Dabei muss eine *UpdateRequestProcessorChain* in **solrconfig.xml** eingefügt werden (siehe Listing 6.15).

```
<updateRequestProcessorChain name="dedupe">
  <processor class="solr.processor.SignatureUpdateProcessorFactory">
    <bool name="enabled">true</bool>
    <str name="signatureField">id</str>
    <bool name="overwriteDupes">false</bool>
    <str name="fields">name,features,cat</str>
    <str name="signatureClass">solr.processor.Lookup3Signature</str>
  </processor>
</updateRequestProcessorChain>
```

Listing 6.15: Quelle: [Foua]

Dabei sind die *fields*, die Felder, die zur Berechnung des Hashwertes verwendet werden. Der Standard ist alle Felder. Das *signatureField* ist das Feld, in das der Hashwert gespeichert wird. Der Standard ist *signatureField*. Es ist wichtig, dass dieses Feld in der Datei *schema.xml* definiert ist. *Enable* gibt an, ob die *De-Duplication* verwendet werden soll. Der Standard ist *true*. Die *signatureClass* gibt die Methode an, mit der gehasht werden soll. Es gibt die Methoden *MD5Signature, ein 128 bit Hash, Lookup3Signature, ein 64 bit Hash, und TextProfileSignature, ein Fuzzy Hash.*

Weiter muss in **solrconfig.xml** noch der *UpdateHandler* derart geändert werden, dass er die *De-Duplication* verwendet (siehe Listing 6.16).

```
<requestHandler name="/update" >
  <lst name="defaults">
    <str name="update.chain">dedupe</str>
  </lst>
</requestHandler>
```

Listing 6.16: Quelle: [Foua]

Wird ein extra Feld zur Speicherung des Hashwertes verwendet, muss dieses, wie Listing 6.17 zeigt, in **schema.xml** eingefügt werden.

```
<field name="signature" type="string" stored="true" indexed="true"
    multiValued="false" />
```

Listing 6.17: Quelle: [Foua]

6.7.7 Anzeige von aussagekräftigen Namen

Die Anzeige von Namen wird im Anwendungsprogramm spezifiziert. Bei den Dateien, die vom Tika Plugin indiziert werden, sollte das Feld *title* verwendet werden, da dies das Feld ist, in den der Titel des Dokuments, falls dieser vorhanden ist, gespeichert wird. Ansonsten speichert Tika entweder den Dateipfad als *title* oder lässt ihn leer. Dann kann bei der Indizierung der Parameter *title* explizit angeben werden.

6.8 Fazit

Im Folgenden wird die Konfigurierbarkeit und Bedienbarkeit von Solr betrachtet. Die Installation von Solr gestaltet sich dank gut verständlicher Anleitungen [ASI] recht einfach. Jedoch ist die Installation nicht ohne Zusatzsoftware möglich. Zum einen muss eine Java Runtime Environment, beziehungsweise für die vollständige Unterstützung aller internationaler Zeichensätze ein komplettes JDK installiert werden. Dieses muss ab Solr 4 eine Versionsnummer von 1.6 oder größer haben. Zum anderen muss ein Servlet-Container oder ein *Stand Alone Runner* [SUF] installiert werden. In der Installationsanleitung finden sich Anleitungen für die Installation von Solr in vielen bekannten Containern. Im Download-Paket von Solr befindet sich bereits eine Installation in einem kleinen Jetty-Container. Dieser ist bereits optimal für Solr konfiguriert und kann direkt über eine .jar Datei ohne Konfigurationsaufwand gestartet werden. Jedoch unterstützt diese Jetty-Installation nur die Benutzung einer einzelnen Solr-Instanz. In diesem Fall kann also die Installation eines Servlet-Containers entfallen. Ansonsten besteht die Möglichkeit eines Updates des Jetty-Containers, oder der Installation in einem anderen Container bzw. stand alone runner. Weiter enthält das Download-Paket einen vorkonfigurierten Beispielkern, in dem ohne Konfiguration Dateien indiziert und gesucht werden können. So kann das System ohne Konfigurationsaufwand getestet werden. Außerdem kann die Beispielkonfiguration als Basis für die eigene Konfiguration dienen.

Die Konfiguration von Solr erfolgt über XML Dateien. Dabei sind die zwei wichtigsten schema.xml und solconfig.xml.
Die Konfiguration von schema.xml ist leicht verständlich. Es werden Felder, dynamische Felder, copyFields, ein Schlüssel und Feldtypen konfiguriert. Zu beachten ist, dass keine Feldtypen vordefiniert sind. Folglich müssen Typen wie string, int, etc., falls diese verwendet werden, ebenfalls in schema.xml definiert werden. Jeder Typ wird von einer vordefinierten Klasse abgeleitet. Dabei bietet Solr eine Vielzahl von Typen an und jeder Typ kann mittels Analysatoren verfeinert werden. So lassen sich für viele Fälle passende Typen definieren. Da man in einer XML-Datei konfiguriert, erhält man keine Debuggingunterstützung. Der Programmierer muss selbst darauf achten, dass die Datei korrekt ist, beispielsweise jeder Feldtyp, der in den Feldern verwendet wird, auch definiert ist. Ansonsten gibt es beim Starten Fehler, und Solr kann nicht verwendet werden. Dabei wird als Fehlermeldung der *Java Stack Trace* [Ora, Foua] ausgegeben, aus dem der Fehler nicht immer direkt abzulesen ist. Nach dem Beheben des Fehlers muss der Servlet-Container neu gestartet werden.
Die Konfiguration von *solrconfig.xml* ist deutlich umfangreicher und komplexer. Man kann in den meisten Fällen jedoch die im Download-Paket enthaltene Beispielkonfiguration verwenden bzw. diese für seine Bedürfnisse anpassen. Eine ausführliche Beschreibung zum Inhalt der *solrconfig.xml* und wie diese optimal konfiguriert wird, ist im Handbuch [Foua] von Solr im Abschnitt *The Well-Configured Solr Instance* gegeben. Auch hier muss der Programmierer wieder selbst darauf achten, dass die Datei korrekt ist.
Die Benutzung von Solr ist relativ leicht zu erlernen. Zum einen gibt es eine einheitliche Schnittstelle über HTTP. So sehen die Anfragen in jeder Programmiersprache gleich aus und es kann der Browser für Beispielanfragen verwendet werden. Zum anderen bietet Solr ein grafisches *AdminUI*, das die meisten Funktionen für die Indizierung und Suche unterstützt. So können die Funktionen ohne Programmierungsaufwand und ohne bzw. mit sehr geringem Einarbeitungsaufwand getestet werden.
Nach der Installation und der Grundkonfiguration können dann die Anforderungen getestet werden. Tabelle 6.3 zeigt alle Anforderung mit einem Fazit.

Anforderungen	**Fazit**
Indizierung verschiedener Dateiformate	Alle geforderten Formate konnten indiziert werden
Tägliche/Dynamische Indizierung der Datenbasis	Wird von Solr nicht unterstützt
Separate Indizierung eines passwortgeschützten Bereichs	Kann in Solr über einen separaten Kern gelöst werden
Einschränkung von Suchen	Solr bietet verschiedene Konzepte für die Einschränkung von Suchen
Plausibilität von Suchen	Bei Solr werden alle Anfragen über den *q-Parameter* angegeben, ist dieser nicht ausgefüllt, wird ein Fehler ausgegeben, also sind die Suchanfragen an Solr plausibel
Suchvarianten	Solr bietet Konzepte für die Suche nach einem, allen, keinem der Suchworte und der exakten Suchwortreihenfolge
Formatierung von Suchergebnissen	Solr bietet Konzepte für das Query Highlighting, die Ausgabe der Suchgeschwindigkeit, die Beschränkung der Anzahl der Ergebnisse, die Ausgabe von Synonymen, der Sortierung nach der Relevanz und der Vermeidung doppelter Einträge.

Tabelle 6.3: Anforderungsauswertung

7

Zusammenfassung und Ausblick

Ziel dieser Arbeit war es, die Realisierung verschiedener Suchszenarien anhand von Apache Solr zu testen. Dazu wurden in Kapitel 2 zu Solr verwandte Arbeiten betrachtet. Weiter wurde in Kapitel 3 die Architektur und in Kapitel 4 die API von Solr untersucht. Außerdem wurden in Kapitel 5 die Anforderungen definiert, die in Kapitel 6 dann untersucht wurden.
Dabei unterstützt Solr über das integrierte Tika Plugin und einen *Data Import Handler* die Indizierung der verschiedenen Datenformate. Über einen separaten Kern, der mit einem Passwort geschützt werden kann, kann die separate Indizierung eines passwortgeschützten Bereichs realisiert werden. Die Einschränkung der Suchen wird mittels **[wert1 TO wert2]** bzw. **{wert1 TO wert2}** realisiert, wobei *wert1* und *wert2* Daten, Zahlen und *Strings* sein können. In Solr laufen alle Abfragen über den *q-Parameter*, ist dieser leer, wird ein Fehler ausgegeben, ansonsten das Suchergebnis. Also sind Suchen in Solr plausibel. Solr unterstützt die Suche nach einem, allen oder keinem der Suchworte, außerdem wird die Suche nach der exakten Suchwortreihenfolge unterstützt. Weiter ist in Solr *Query Highlighting*, die Ausgabe der Suchgeschwindigkeit, die Einschränkung der Anzahl der Suchergebnisse, die Ausgabe von Synonymen, die Sortierung nach der Relevanz und die Vermeidung doppelter Einträge möglich.
Allerdings ist es in Solr nicht möglich, eine tägliche bzw. dynamische Indizierung der Datenbasis zu realisieren. Es ist ein Programm nötig, dass die Datenbasis täglich bzw. dynamisch an Solr sendet.
Diese Szenarien kommen aus der Anwendung. Dabei wurden die Wichtigsten identifiziert und für die Untersuchung ausgewählt.

Literaturverzeichnis

[Aok] Osamu Aoki. Debian referenz (v2). https://www.debian.org/doc/manuals/debian-reference/. Aufrufdatum: 11.01.2015.

[ASI] Solrinstall. https://wiki.apache.org/solr/SolrInstall. Aufrufdatum: 07.01.2015.

[ATT] Analyzers, tokenizers, and token filters. https://wiki.apache.org/solr/AnalyzersTokenizersTokenFilters. Aufrufdatum: 07.01.2015.

[BBP09] Stephan Buchwald, Thomas Bauer, and Rüdiger Pryss. *IT-Infrastrukturen für flexible, service-orientierte Anwendungen - ein Rahmenwerk zur Bewertung.* 13. GI-Fachtagung Datenbanksysteme für Business, Technologie und Web, 2009.

[Dam64] Fred J. Damerau. *A technique for computer detection and correction of spelling errors.* Communications of the ACM, 1964.

[DIH] Solrwiki - dataimporthandler. https://wiki.apache.org/solr/DataImportHandler. Aufrufdatum: 07.01.2015.

[Ela] Elasticsearch. Elasticsearch reference - 1.4. http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/index.html. Aufrufdatum: 07.01.2015.

[Foua] Apache Software Foundation. *Apache Solr Reference Guide 10.4.*

[Foub] The Apache Software Foundation. Apache lucenetm 4.10.3 documentation. http://lucene.apache.org/core/4_10_3/index.html. Aufrufdatum: 07.01.2015.

[Fouc] The Apache Software Foundation. Apache tika - supported document formats. http://tika.apache.org/1.6/formats.html#Supported_Document_Formats. Aufrufdatum: 07.01.2015.

[Foud] The Apache Software Foundation. Apache tika 1.6. http://tika.apache.org/1.6/index.html. Aufrufdatum: 07.01.2015.

[GP14] Trey Grainger and Timothy Potter. *Solr in Action.* Manning Publications, 2014.

[INCa] OPENSEARCHSERVER INC. Open search server documentation. http://www.opensearchserver.com/documentation/README.md. Aufrufdatum: 07.01.2015.

[Incb] Sphinx Technologies Inc. Sphinx 2.2.6-release reference manual. http://sphinxsearch.com/docs/latest/index.html. Aufrufdatum: 07.01.2015.

[Int] Inc. Intalio. Jetty : The definitive reference - revision 9.2.7-snapshot. http://www.eclipse.org/jetty/documentation/current/. Aufrufdatum: 11.01.2015.

[Kar13] Hrishikesh Vijay Karambelkar. *Scaling Big Data with Hadoop and Solr.* Packt Publishing, 2013.

[Kar14] Hrishikesh Vijay Karambelkar. *Scaling Apache Solr*. Packt Publishing, 2014.

[Kuc13] Rafal Kuc'. *Apache Solr 4 Cookbook*. Packt Publishing, 2013.

[Kuc15] Rafal Kuc'. *Solr Cookbook - Third Edition*. Packt Publishing, 2015.

[Kum13] Jayant Kumar. *Apache Solr PHP Integration*. Packt Publishing, 2013.

[KW14] Markus Klose and Daniel Wrigley. *Einführung in Apache Solr*. O'Reilly Verlag, 2014.

[Lev66] Vladimir I. Levenshtein. *Binary codes capable of correcting deletions, insertions, and reversals*. Soviet Physics Doklady, 1966.

[Moh13] Surendra Mohan. *Administrating Solr*. Packt Publishing, 2013.

[Moh14] Surendra Mohan. *Apache Solr High Performance*. Packt Publishing, 2014.

[Ora] Oracle. jstack - stack trace. `http://docs.oracle.com/javase/7/docs/technotes/tools/share/jstack.html`. Aufrufdatum: 13.02.2015.

[Raf13] Alexandre Rafalovitch. *Instant Apache Solr for Indexing Data How-to*. Packt Publishing, 2013.

[SEC] Solr security. `https://wiki.apache.org/solr/SolrSecurity`. Aufrufdatum: 07.01.2015.

[Ser13] Alfredo Serafini. *Apache Solr Beginner's Guide*. Packt Publishing, 2013.

[SP11] David Smiley and Eric Pugh. *Apache Solr 3 Enterprise Search Server*. Packt Publishing, 2011.

[SUF] Solr-undertow for solr 4+. `https://github.com/bremeld/solr-undertow`. Aufrufdatum: 12.02.2015.