

AN INVESTIGATION INTO SOLR AND THE FULL-TEXT INDEXING OF DIGITIZED BOOKS

TOM TERNQUIST AND JONATHAN YU

ABSTRACT. The objective of this project was to conduct exploratory work in preparation for the full-text indexing of the collection of Cornell University Library's digitized books, using the facilities provided by the Web Lab at the Cornell Center for Advanced Computing. The project's primary focus was on exploring the Solr search system, but investigations of several other Lucene-based searched systems were included as well a study of possible integration with the Web Lab's Hadoop cluster.

The study of Solr involved profiling preprocessing and indexing performance of a subset of approximately 2,000 of Cornell's digitized books. Results indicate that while Solr has the potential to serve as a search system for Cornell's digitized books, work needs to be done in configuring the system so that computationally intensive tasks can be offloaded to a distributed system on the Web Lab's Hadoop cluster.

CONTENTS

1. Introduction	3
2. Examination of Lucene Distributions	3
2.1. Lucene	3
2.2. Nutch/NutchWAX	3
2.3. Solr	3
3. Methodology	4
4. Results	4
4.1. Profiling Indexing Time	4
4.2. Profiling of Processing Time	5
5. Discussion of Profiling Results	6
5.1. Indexing Time	6
5.2. Processing Time	6
6. General Discussion	6
6.1. Solr Schema	6
6.2. Other Analysis and Solr Features	7
7. Conclusions	7
8. Acknowledgments	8
References	8
Appendices	9

Date: May 8, 2009.

A. Zip Extraction Script	9
B. Text to XML Script	10
C. Solr Schema	12

1. INTRODUCTION

One of the primary challenges of working with large data-sets is dealing with the computationally intensive tasks that arise from the size of the data. In the case of the the Web Lab's Digitized Books team, the data consists of large collections of digitally scanned books and the primary task needing to be performed is a full-text indexing on these collections.

The Web Lab's Hadoop cluster provides an ideal environment for such processing through use of MapReduce programming [4]. Hadoop [2] and its distributed filesystem (HDFS) uses MapReduce tasks effectively to divide the load across multiple nodes in the cluster.

The Digitized Books team has based much of its knowledge off of the extensive work done by the Internet Archive [1] and their use of NutchWAX for the search and indexing of web archive collections. NutchWAX is a Lucene-based searched system which is capable of indexing using Hadoop and Map-Reduce tasks and is a natural candidate for searching at the Web Lab.

Despite its integration with Hadoop, the team has focused on exploring the Apache Solr distribution of Lucene[3]. This package is being studied primarily because it supports using the rich metadata existing for books, by using Solr's flexible schemas. The team's initial study of Solr has found that it does not currently support operation in a Hadoop environment.

The team's work focuses on the utility and scalability of Solr for the indexing and searching of Cornell digitized books collection.

2. EXAMINATION OF LUCENE DISTRIBUTIONS

2.1. Lucene. The team explored a variety of packages for full-text indexing of books. Many of the distributions were based of the Apache Lucene distribution. Lucene is an open-source application which is well known for its use in full-text indexing and search capabilities.

2.2. Nutch/NutchWAX. Much of the team's early investigations into a potential solution was based off of previous work in the searching and indexing of web archive collections. Specifically, work from the Internet Archive and their use of NutchWAX.

NutchWAX is an extension of the Nutch, an open-source web-search system based on Lucene. NutchWAX is oriented toward the searching of web archives, typically stored in ARC files, as opposed to Nutch which focusing on building its indices from crawls of the web.

One of the more attractive features of Nutch/NutchWAX is its ability to run on the Hadoop cluster for the purposes of indexing. The Hadoop cluster supports a distributed file system and the MapReduce programming paradigm. This functionality is desirable both on the web and in the case of digitized books, where there is a significant amount of batch data to be processed in indexing.

2.3. Solr. Unlike the open web, digitized books maintain large amounts of metadata which can be used for fielded searching. While the benefits of full-text search may reduce the

reliance on such metadata, any search solution would need to allow for the indexing of these data as well.

Solr's indexing is defined by its schema, which provides a wide array of customizations to provide for more intelligent processing and indexing of data. Use of Solr's schema looks to offer support for metadata as well as other indexing techniques that may be found to provide better results for book search.

3. METHODOLOGY

This semester, the team mainly focused on determining the feasibility of processing and indexing the books on a single node on the Web Lab cluster. The digital books were transferred early in the semester to the hadoop distributed file system.

The current installation of hadoop did not support selective extraction directly from the distributed cluster, thus the team needed to copy zip files of the books locally to process. The team temporary local storage was a Windows 2003 server. This shared drive was mounted to the team's Unix-based node.

The team wrote a perl script to selectively extract all OCR scans from the book's zip files. The extraction retrieved all raw text and a file listing all the containing OCR pages. It ignored scanned images and layout/position files as they were not used for initial Solr indexing. This manifest file could have been used to verify files and also further exploration and integrating with other Solr distributions (such as Nutch).

Solr requires documents be formatted in the proper schema for indexing and posting to the installation. The OCR extracted text contained text files of individual pages, and thus needed to be cleaned and written to match our created schema designation. The team wrote a processing script in Java which would take the folder of raw OCR, remove miscellaneous control characters, and write each page to a separate XML file. These were placed in a separate folder.

Following, the team used the existing posting tool provided in the sample Solr package to index and post the XML files to the Solr installation. The team recorded the runtimes in order to give indication on the scaling issues associated with indexing and posting on a single machine.

In addition to profiling indexing, the team also explored the variety of features provided by Solr. The team also began to examine the OCR text provided in the book scans zip files to see if more detailed processing and formatting could be achieved.

4. RESULTS

4.1. Profiling Indexing Time. The team was able to conduct profiling tests of the time required for sets of books to be indexed by Solr once in XML form. Configuration issues led to difficulty processing sets of data larger than approximately 2,000 books and as such our results are limited to data-sets that were possible with our processing capabilities. Below is a summary of our results.

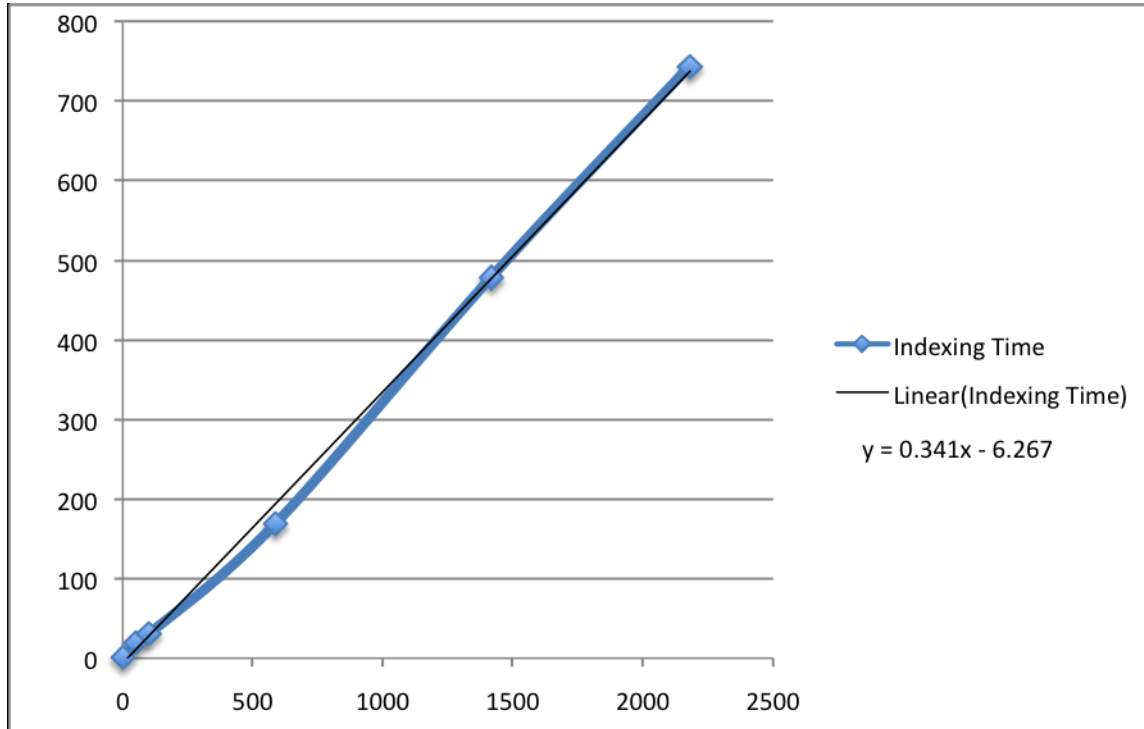


FIGURE 1. Graph of indexing time as a function of the number of books indexed

Indexing Time	
Books	Time(sec)
1	1.0
50	19.3
100	30.6
587	168.9
1417	478.0
2180	742.8

4.2. **Profiling of Processing Time.** In addition to indexing, some measurements of the time it takes to process the books prior to indexing. These times ended up being significant and are outlined below.

Extraction Time (from zip)		Time to XML	
Books	Time(sec)	Books	Time(sec)
587	50.9	587	55.9
2180	7.7	2180	752.7

5. DISCUSSION OF PROFILING RESULTS

5.1. Indexing Time. Although the team's tests lacked the scale desired, the data suggest that indexing time on a single node increases linearly as a function of the number of books indexed. The indexing times, while significant, are not unreasonable. While clearly there is room for improvement, it appears that indexing time is likely to fall within a passable range for a corpus on the order of 100,000 books.

The results do not reflect any memory usage, and as the size of the index grows, it is unclear if memory will become an issue.

5.2. Processing Time. Preliminary analysis of the tools used to perform the preprocessing of the books for indexing indicate that the time necessary for completion is quite significant and may be a limiting factor in the indexing process as the number of books indexed increases.

5.2.1. Hadoop and MapReduce. The time results for processing and formatting the books into correct document schema syntax is significant; it prompts the question the runtime for processing even a large number of files or even the entire library collection. This existing workflow does not cater to such a large number, and there could even be memory issues with processing everything on a single node. In order to counter the increasing processing time and to fully utilize the distributed computing environment afforded by the WebLab, the next step would be to examine ways to extract and convert OCR text to XML using MapReduce techniques.

5.2.2. Integration with Hadoop. As previously mentioned, Solr does not support Hadoop in the creation of an index or for search. A configuration where extraction, processing, and indexing was handled using Hadoop and MapReduce tasks has the potential to significantly reduce the time it takes create an index, especially as the size of the dataset grows larger.

While the lack of Hadoop support is certainly a severe limitation, the potential to utilize Hadoop for the preprocessing tasks, such as extracting the OCR text from the zip files and the conversion of this text to XML, may not be difficult to realize.

The actual integration of Solr with Hadoop will likely require a significant amount of time and effort. However, since Nutch/NutchWAX does support Hadoop for indexing, it may be feasible to use the code from these systems and integrate it into a custom Solr configuration.

6. GENERAL DISCUSSION

6.1. Solr Schema. Solr's XML schema is used to designate all the details and fields that each document should contain. It specifies how fields should be treated when indexing and also in querying. This allows a Solr distribution to be customized for its underlying corpus.

The schema file allows one to specify data types, fields, and dynamic fields. These can be specified with varying levels of granularity. For our initial indexing experiments, we were focused mainly on determining the scalability factors associated with a Solr installation on

a single machine. We used a very basic schema that split each book as a document, and had each page as a separate field.

6.2. Other Analysis and Solr Features. Solr includes other useful features including search highlighting and faceted search. This semester, the team did not incorporate many of these built-in features. These other features can be explored in the future especially after discussing needs of researchers and librarians for a full-text index search.

6.2.1. Dirty OCR and Fuzzy Searching. The team's initial examination of the OCR scans of the books to be indexed has revealed that the text files contain noticeable typographic errors as a result of the scanning process. This dirty OCR presents a few challenges to the indexing of the books.

First, the typographical errors may lead to both missed terms and redundancy, likely skewing the results of the search. A possible solution, to at least the problem of missed terms, may come from Lucene's support of fuzzy searching through its querying interface. A custom Solr application, with default support for this querying option could provide be relatively easy to implement, but its affect on performance is unknown.

While not necessarily a directly a result of dirty OCR, much of the formatting found in the original books, such as paragraphs, is lost in the digitization process. Future work may will likely benefit from the study of what semantic information can be retained from the books beyond page separation.

6.2.2. Incorporating Metadata into Schema. Several libraries have already experimented with creating specialized search interfaces with MARC records for books. These explorations have demonstrated various features of Solr, including Lucene's robust query language. This semester, the team was most interested in determining whether existing techniques and processes used for indexing and archiving the web were applicable to the domain of digital books.

A significant amount of rich metadata and other information exists in the zip files, and further research could focus on using and integrating these into the index to allow additional search criteria and fields. As mentioned, further work should be conducted on a suitable Solr schema for fulltext sources.

The zip files also include image scans of pages as well as position files of words and possibly images. The team did not explore these in depth this semester, however, these could be useful in determining semantically relevant method to split books.

7. CONCLUSIONS

The team's analysis of the Solr and the various other search systems potential available for the full-text indexing of digital books has revealed some interesting results. Solr has been found to be a potentially viable search solution, but it remains to be seen if the system can be configured to scale to accommodate the large amounts of data associated with the digital books.

Looking ahead, the team suggests that future work be focused on which elements of the indexing process can best be adapted for distributed processing using Hadoop and

MapReduce programming. Additionally, further study of Solr's schema and how it can best be used to extract more semantic information from the books.

8. ACKNOWLEDGMENTS

We would like to thank Professor William Arms for his guidance over the course of this project. We would also like to thank Lucia Walle of the Center for Advanced Computing for her assistance in maintaining the nodes on the Hadoop cluster and Bill Kehoe of the Cornell University Library for his support in the transfer of digitized books to the Web Lab.

The Cornell Web Lab is funded in part by National Science Foundation grants CNS-0403340, SES-0537606, IIS-0634677, IIS-0705774 and IIS 0917666.

REFERENCES

- [1] The Internet Archive <http://www.archive.org/>
- [2] Hadoop. <http://hadoop.apache.org/>
- [3] Solr. <http://lucene.apache.org/solr/>
- [4] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters." 137-150.

Appendices

A. ZIP EXTRACTION SCRIPT

```
#!/usr/bin/perl

use File::Basename;

my($path) = $ARGV[0];
my($dest) = $ARGV[1];
$dest .= "/" if ($dest !~ /\$/);
$path  .= "/" if ($path !~ /\$/);

for my $file (glob($path.*')) {
    my $fn = basename($file);
    if($fn =~ /(\w+)(.zip)/) {
        my $filename = $1;
        $cmd = "unzip -x -o -d ".$dest.$filename." ".$file."
            ... METADATA/".$filename."_AllFilesManifest.txt OCR/*.txt";
        #print $cmd,"\n";
        system $cmd;
    }
}

exit;
```

B. TEXT TO XML SCRIPT

```

import java.io.*;

public class ReadPages {
/**
 * Read in files , remove control characters dump to xml
 *
 * @param Args
 * @throws IOException
 */
public static void main(String args []) {
    String input_folder = args [0];
    String output_folder = args [1];
    File path = new File(input_folder);
    File dataSet [] = path.listFiles ();
    for (int i = 0; i < dataSet.length; i++) {
        File XMLfile = new File(output_folder + "output_" +
            ... dataSet [i].getName () + ".xml");
        //System.out.println (XMLfile);
        BufferedWriter wr;
try {
    wr = new BufferedWriter(new OutputStreamWriter(
        new FileOutputStream (XMLfile), "UTF-8"));
    BufferedReader br = null;
    String line = null;
    File dir [] = dataSet [i].listFiles ();
    if (dir.length < 2)
        continue;
    File files [] = dir [0].listFiles ();
    wr.write("<add><doc>" + "\t<field name =\"id\">"
        + dataSet [i].getName () + "</field>\n");
    for (int x = 0; x < files.length; x++) {
        if (!files [x].toString ().substring (8, 9).equals (".")
            && !files [x].toString ().equals ("")) {
            wr.write ("\t<field name =\"page_" + x + "\" >");
            br = new BufferedReader (new FileReader (files [x]));
            while ((line = br.readLine ()) != null) {
                line = line.replaceAll ("\p{Cntrl}", "");
                line = line.replaceAll ("[\\"\\|\\|\\|]", " ");
            }
        }
    }
}
}
}

```

```
        line = line.replaceAll("[\\t]", "");
        line = line.replaceAll("[\\W]", " ");
        wr.write(line);
    }
    br.close();
    wr.write("</field>\n");
}
}
wr.write("</doc></add>\n");
wr.close();
} catch (UnsupportedEncodingException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (FileNotFoundException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
}
}
}
```

C. SOLR SCHEMA

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--
Licensed to the Apache Software Foundation (ASF) under one or more
contributor license agreements. See the NOTICE file distributed with
this work for additional information regarding copyright ownership.
The ASF licenses this file to You under the Apache License, Version 2.0
(the "License"); you may not use this file except in compliance with
the License. You may obtain a copy of the License at
```

```
http://www.apache.org/licenses/LICENSE-2.0
```

```
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

```
—>
```

```
<!--
```

```
This is the Solr schema file. This file should be named "schema.xml" and
should be in the conf directory under the solr home
(i.e. ./solr/conf/schema.xml by default)
or located where the classloader for the Solr webapp can find it.
```

```
This example schema is the recommended starting point for users.
It should be kept correct and concise, usable out-of-the-box.
```

```
For more information, on how to customize this file, please see
http://wiki.apache.org/solr/SchemaXml
```

```
—>
```

```
<schema name="marcxml" version="1.1">
<!-- schema for marcxml metadata and indexed pages of corresponding book —>
```

```
<types>
```

```
<!-- The format for this date field is of the form 1995-12-31T23:59:59Z, and
```

is a more restricted form of the canonical representation of `dateTime`
<http://www.w3.org/TR/xmlschema-2/#dateTime>

The trailing "Z" designates UTC time and is mandatory.

Optional fractional seconds are allowed: `1995-12-31T23:59:59.999Z`

All other components are mandatory.

Expressions can also be used to denote calculations that should be performed relative to "NOW" to determine the value, ie...

`NOW/HOUR`

... Round to the start of the current hour

`NOW-1DAY`

... Exactly 1 day prior to now

`NOW/DAY+6MONTHS+3DAYS`

... 6 months and 3 days in the future from the start of the current day

Consult the `DateField` javadocs for more information.

—>

```
<fieldType name="date" class="solr.DateField" sortMissingLast="true"
omitNorms="true"/>
```

```
<fieldType name="string" class="solr.StrField" sortMissingLast="true"
omitNorms="true"/>
```

```
<fieldType name="page" class="solr.TextField" positionIncrementGap="100">
<analyzer type="index">
<tokenizer class="solr.WhitespaceTokenizerFactory"/>
<!-- in this example, we will only use synonyms at query time
<filter class="solr.SynonymFilterFactory" synonyms="index-synonyms.txt"
ignoreCase="true" expand="false"/>
```

—>

```
<!-- Case insensitive stop word removal.
enablePositionIncrements=true ensures that a 'gap' is left to
allow for accurate phrase queries.
```

—>

```
<filter class="solr.StopFilterFactory"
ignoreCase="true"
words="stopwords.txt"
```

```

        enablePositionIncrements="true"
    />
<filter class="solr.WordDelimiterFilterFactory" generateWordParts="1"
    generateNumberParts="1" catenateWords="1" catenateNumbers="1"
    " catenateAll="0" splitOnCaseChange="1"/>
<filter class="solr.LowerCaseFilterFactory"/>
<filter class="solr.EnglishPorterFilterFactory" protected="protwords.txt"/>
<filter class="solr.RemoveDuplicatesTokenFilterFactory"/>
</analyzer>
<analyzer type="query">
<tokenizer class="solr.WhitespaceTokenizerFactory"/>
<filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" i
    gnoreCase="true" expand="true"/>
<filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"
/>
<filter class="solr.WordDelimiterFilterFactory" generateWordParts="1"
    generateNumberParts="1" catenateWords="0" catenateNumbers="0"
    catenateAll="0" splitOnCaseChange="1"/>
<filter class="solr.LowerCaseFilterFactory"/>
<filter class="solr.EnglishPorterFilterFactory" protected="protwords.txt"/>
<filter class="solr.RemoveDuplicatesTokenFilterFactory"/>
</analyzer>
</fieldType>

```

```

<fieldType name="text" class="solr.TextField" positionIncrementGap="100">
<analyzer type="index">
<tokenizer class="solr.WhitespaceTokenizerFactory"/>
<!-- in this example, we will only use synonyms at query time
<filter class="solr.SynonymFilterFactory" synonyms="index-synonyms.txt"
    ignoreCase="true" expand="false"/>
-->
<!-- Case insensitive stop word removal.
        enablePositionIncrements=true ensures that a 'gap' is left to
        allow for accurate phrase queries.
-->
<filter class="solr.StopFilterFactory"
    ignoreCase="true"
    words="stopwords.txt"

```

```

        enablePositionIncrements="true"
    />
<filter class="solr.WordDelimiterFilterFactory" generateWordParts="1"
generateNumberParts="1" catenateWords="1" catenateNumbers="1" catenateAll="0"
splitOnCaseChange="1"/>
<filter class="solr.LowerCaseFilterFactory"/>
<filter class="solr.EnglishPorterFilterFactory" protected="protwords.txt"/>
<filter class="solr.RemoveDuplicatesTokenFilterFactory"/>
</analyzer>
<analyzer type="query">
<tokenizer class="solr.WhitespaceTokenizerFactory"/>
<filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"
ignoreCase="true"
expand="true"/>
<filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"/>
<filter class="solr.WordDelimiterFilterFactory" generateWordParts="1"
generateNumberParts="1" catenateWords="0" catenateNumbers="0" catenateAll="0"
splitOnCaseChange="1"/>
<filter class="solr.LowerCaseFilterFactory"/>
<filter class="solr.EnglishPorterFilterFactory" protected="protwords.txt"/>
<filter class="solr.RemoveDuplicatesTokenFilterFactory"/>
</analyzer>
</fieldType>

</types>

```

```
<fields>
```

```
<!-- Valid attributes for fields:
```

```
name: mandatory – the name for the field
```

```
type: mandatory – the name of a previously defined type from the <types> section
```

```
indexed: true if this field should be indexed (searchable or sortable)
```

```
stored: true if this field should be retrievable
```

```
compressed: [false] if this field should be stored using gzip compression
(this will only apply if the field type is compressable; among
the standard field types, only TextField and StrField are)
```

```
multiValued: true if this field may contain multiple values per document
```

```
omitNorms: (expert) set to true to omit the norms associated with
this field (this disables length normalization and index-time
boosting for the field, and saves some memory). Only full-text
```

fields or fields that need an index-time boost need norms.
termVectors: [false] set to true to store the term vector for a given field.
When using MoreLikeThis, fields used for similarity should be stored for
best performance.

—>

<!-- Here, default is used to create a "timestamp" field indicating
When each document was indexed.

—>

```
<field name="timestamp" type="date" indexed="true" stored="true"
  default="NOW" multiValued="false"/>
<field name="text" type="text" indexed="true" stored="false"
  multiValued="true"/>
<field name="id" type="string" indexed="true" stored="true"
  required="true" />
<field name="page" type="page" indexed="true" stored="true"
  multiValued="true" />
```

<!-- Dynamic field definitions. If a field name is not found,
dynamicFields will be used if the name matches any of the patterns.
RESTRICTION: the glob-like pattern in the name attribute must have
a "*" only at the start or the end.

EXAMPLE: name="*_i" will match any field ending in _i (like myid.i, z_i)
Longer patterns will be matched first. if equal size patterns
both match, the first appearing in the schema will be used. —>

```
<dynamicField name="page_*" type="page" indexed="true" stored="true"/>
```

```
</fields>
```

```
<copyField source="page_*" dest="page"/>
```

<!-- Field to use to determine and enforce document uniqueness.

Unless this field is marked with required="false", it will be a required field

—>

```
<uniqueKey>id</uniqueKey>
```

<!-- field for the QueryParser to use when an explicit fieldname is absent —

```
<defaultSearchField>page</defaultSearchField>
```

```
<!-- SolrQueryParser configuration: defaultOperator="AND|OR" -->
<solrQueryParser defaultOperator="OR"/>

<!-- Similarity is the scoring routine for each document vs. a query.
A custom similarity may be specified here, but the default is fine
for most applications. -->
<!-- <similarity class="org.apache.lucene.search.DefaultSimilarity"/> -->
<!-- ... OR ...
Specify a SimilarityFactory class name implementation
allowing parameters to be used.
-->
<!--
<similarity class="com.example.solr.CustomSimilarityFactory">
<str name="paramkey">param value</str>
</similarity>
-->

</schema>
```