# **CHAPTER ONE**

## 1.0 INTRODUCTION

### 1.1 BACKGROUND STUDY

Despite ever-increasing capacities, significant benefits can still be realized by reducing the number of bytes needed to represent an object when it is stored or retrieved from a database. The benefits can be especially great for mobile devices with limited storage or bandwidth; reference data (data that are saved permanently and accessed infrequently); e-mail, in which large byte sequences are commonly repeated; and data transferred over low-bandwidth or congested links. Reducing bytes generally equates to eliminating unneeded data, and there are numerous techniques for reducing redundancy when objects are stored or sent. The most longstanding example is data compression, which eliminates redundancy internal to an object and generally reduces textual data by factors of two to six. Duplicate suppression eliminates redundancy caused by identical objects which can be detected efficiently by comparing hashes of the objects' content (OOCL, 2016).

Delta-encoding eliminates redundancy of one object relative to another, often an earlier version of the object by the same name. Delta-encoding can in some cases eliminate an object almost entirely, but the availability of base versions against which to compute a delta can be problematic. Recently, much work has been performed on applying these techniques to pieces of individual objects. This includes suppressing duplicate pieces of files and web pages. Delta-encoding has also been extended to pairs of files that do not share an explicit versioning relationship. There are also approaches that combine multiple techniques; for instance, the vcdiff program not only encodes differences between a ``version'' file and a ``reference'' file, it compresses redundancy within the version file. Delta-encoding that simultaneously compresses is sometimes called ``delta compression''. In fact, no single technique can be expected to work best across a wide variety of data sets. There are numerous trade-offs between the effectiveness of data reduction and the resources required to achieve it, i.e. its efficiency.

The relative importance of these metrics, effectiveness versus efficiency, depends on the environment in which techniques are applied. Execution time for example, which is an important aspect of efficiency, tends to be more important in interactive contexts than in asynchronous ones. In this paper, we describe a new data reduction technique that achieves comparable effectiveness to current delta- encoding techniques but with greater efficiency. It simultaneously offers better effectiveness than current duplicate suppression techniques at moderately higher cost. The argue that performing comparisons at the granularity of files can miss opportunities for redundancy elimination, as can techniques that rely on large contiguous pieces of files to be identical. (Henson's study of issues relating to comparing blocks by hashes of their content made a similar argument) Instead, we consider what happens if some of the above techniques are further combined. Specifically, we describe a system that supports the union of three techniques: compression, elimination of identical content-defined chunks, and delta-compression of similar chunks. We refer to this technique as Redundancy Elimination at the Block Level (REBL). The key insight of this work is the ability to achieve more effective data reduction by exploiting relationships among similar blocks, rather than only among identical blocks, while keeping computational and memory overheads comparable to techniques that perform redundancy detection with coarser granularity. We compare our new approach with a number of baseline techniques, which are summarized here and described in detail in the next section: Whole-file compression. With whole-file compression (WFC), each file is compressed individually. This approach gains no benefit from redundancy across files, but it scales well with large numbers of files and is applicable to storage and transfer of individual files. Joining a collection of files into a single object which is then compressed has the potential to detect redundancy both within and across files. This approach tends to find redundancy across files only when the files are relatively close to one another in the object. The abbreviated this technique TGZ, for tar+gzip, the combination we used. Block-level duplicate detection. There are a number of approaches to identifying identical pieces of data across files more generally. These include using fixed-size blocks, fixed-size blocks with rolling checksums and content-defined (and therefore variable-sized) chunks. Delta-encoding using resemblance detection. Resemblance detection techniques can be used to find similar files, with delta compression used to encode them effectively. There are also cases where effectiveness is dramatically improved by combining multiple techniques, such as adding compression to block-level or chunk-level duplication detection. (Fred, Jason, & John, 2002).

Duplicate detection in notice and public comment rulemaking is one such task and our work is designed to be general enough to apply to other detection tasks for documents with “intermediate level of similarity”. In notice and public comments rulemaking domain there are two broad categories of documents: comments that are written more or- less from scratch (they contain unique insights and opinions, derivative opinions, spam or viruses), and comments that are written based on a form letter (Yang & Callan, 2006).

The former will produce singleton clusters and the later will form duplicate clusters. Two documents tends to be duplicates if they are from the same origin. In particular, several subcategories of duplicates are defined based on common editing styles:

1. *Block Added:* Add one or more paragraphs (<200 words) to a document;
2. *Block Deleted*: Remove one or more paragraphs (<200words) from a document;
3. *Key Block*: Contains at least one paragraph from a document;
4. *Minor Change*: A few words altered within a paragraph (<5% or 15 word change in a paragraph);
5. *Minor Change & Block Edit*: A combination of minor change and block edit;
6. *Block Reordering*: Reorder the same set of paragraphs;
7. *Repeated*: Repeat the entire document several times in another document;
8. *Bag-of-word similar*: >80% word overlap (not in above categories); and
9. *Exacta*: 100% word overlap.

# **REFERENCES**

Fred, D., Jason, V., & John, T. M. (2002). REDUNDANCY ELIMINATION WITHIN LARGE COLLECTIONS OF FILES.

OOCL. (2016, January). *Duplicate Detection and Resolution.* Retrieved from https://www.oclc.org.

Yang, H., & Callan, J. (2006, August 6–11). Near-Duplicate Detection by Instance-level Constrained Clustering. pp. 2-3.

### 1.2 PROBLEM DEFINITION

Duplicate detection is different from other Information Retrieval (IR) considering how it defines the similarity check between two or more documents. In many IR document similarity refers to semantic “relevance” among documents, which are could be syntactically very different but still relevant. In contrast, the appearance similarity check in duplicate detection in early database research is very conservative, which is mainly to find syntactically “almost-identical” documents, for other tasks that need to detect documents with “intermediate level of similarity”, there has not been much research done.

The major problem that do arise is that, as more data is being populated into a database table, there is tendency for the table to store duplicate or redundant record which results in the consumption of data spaces in the database and also in the storage device the database resides.

### 1.3 AIMS AND OBJECTIVES

This project is majorly designed based on Simil algorithm to reduce the presence of duplicate records in a database, and it also aim to achieve the following objectives:

1. To document the activity progress of the application itself for reference purpose.
2. To identify the presence of duplicate records in a MySQL database local server installed on a computer system.
3. To serve as a means of reducing the quantity of disk space that MySQL database server is accommodating on a computer system.
4. To provide an automated means of executing database record optimization.

### 1.4 PROJECT SCOPE AND LIMITATION

The project, Database Record Duplicate Detection System is designed using Java Object Oriented Programming Language and MySQL ODBC J-Connector Library for connecting to the database server of MySQL in order to cover up the following scope:

1. An interface for executing the entire application activity for database duplicate optimization.
2. A Tab-pane to display the list of record stored in a selected database table.
3. An optimization action to triggers Simil Algorithm for checking duplicate from a database tables and the creation of the activity log file.
4. An action to save the application log file.
5. A Tab-pane to fetch out a selected log form the log folder as a documented text format.
6. A sample database to test the effectiveness if the application.

The project is designed to limit its operation towards the use of MySQL database server only and does not include the check for file duplicate witch is also possible as at later.

### 1.5 DEFINITION OF TERMS

* 1. **Automation:** the use of computers and machines instead of people to do a job.
  2. **Database:** A repository for storing operational data.
  3. **Duplicate File:** the part of a file that can be eliminated without loss of essential information.
  4. **Information Retrieval:** the techniques of storing and recovering and often disseminating recorded data especially through the use of a computerized system.
  5. **ODBC:** Object Database Connectivity.
  6. **Optimization:** The procedure or procedures used to make a system or design as effective or functional as possible, especially the mathematical techniques involved.
  7. **Records:** The collection of logical related fields.
  8. **Redundancy:** the part of a message that can be eliminated without loss of essential information.
  9. **Simil Algorithm:** one of the similarity search and verification algorithm designed for data, string, and pattern matching.

# **CHAPTER TWO**

## 2.0 LITERATURE REVIEW

### 2.1 BRIEF HISTORY OF DUPLICATE DETECTION

Beginning in 1991, OCLC used its Duplicate Detection and Resolution (DDR) software to match WorldCat bibliographic records in the books format against themselves to find and merge duplicates.

The first computer programs to use pattern matching were text editors. At Bell Labs, Ken Thompson extended the seeking and replacing features of the QED editor to accept regular expressions. Early programming languages with pattern matching constructs include SNOBOL from 1962, SASL from 1976, NPL from 1977, and KRC from 1981. The first programming language with tree-based pattern matching features was Fred McBride's extension of LISP, in 1970

Knuth-Morris-Pratt Algorithm was conceived by Donald Knuth and Vaughan Pratt and independently by James H. Morris in 1977. The both discovered first linear time string-matching algorithm by analysis of the naïve algorithm. It keeps the information that naive approach wasted gathered during the scan of the text. By avoiding this waste of information, it achieves a running time of O (m + n). The implementation of Knuth-Morris-Pratt algorithm is efficient because it minimizes the total number of comparisons of the pattern against the input string (Knuth–Morris–Pratt Algorithm, 2002).

By mid-2005 when WorldCat migrated to its new platform, sixteen runs through WorldCat had been completed, resulting in the elimination of a total of 1.6 million duplicate records.

In 2005, a project was started to re-invent the DDR software to work in the new environment and to expand its capabilities to deal with all types of bibliographic records. This large multi-year project is now bearing fruit. Great improvements to our matching software, which are a key component of the new DDR, have regularly been incorporated into the batch loading process. This helps bring both DDR and batch loading processes into alignment as never before in dealing with the problem of duplicate records in WorldCat.

In May 2009, the new software was put into production following rigorous planning, development, and testing. In addition to its ability to deal with continuing resources, scores, sound recordings, visual materials, maps, and electronic resources, as well as books, this new DDR is much more sophisticated than its predecessor in its power to distinguish legitimate matches from incorrect ones. It also has the flexibility to allow selection of certain categories of bibliographic records to target for deduplication. Processing of small subsets of WorldCat against the live database has begun. A full pass through the WorldCat database began in February 2010 and ended in September 2010.

Having the new DDR software in production is resulting in the merging of a larger number of bibliographic records. Libraries will notice fewer duplicate records in WorldCat. This should be particularly visible for printed music, sound recordings and AV materials since the previous DDR software did not address these duplicates. Regular removal of duplicates provides a better WorldCat for all its users (OOCL, 2016).

Often, in the real world, entities have two or more representations in databases. Duplicate records do not share a common key and/or they contain errors that make duplicate matching a difficult task. Errors are introduced as the result of transcription errors, incomplete information, lack of standard formats, or any combination of these factors. In this paper, we present a thorough analysis of the literature on duplicate record detection. We cover similarity metrics that are commonly used to detect similar field entries, and we present an extensive set of duplicate detection algorithms that can detect approximately duplicate records in a database. We also cover multiple techniques for improving the efficiency and scalability of approximate duplicate detection algorithms. We conclude with coverage of existing tools and with a brief discussion of the big open problems in the area (Elmagarmid, Ipeirotis, & Verykios, 2006 ).

# **REFERENCES**

Elmagarmid, A., Ipeirotis, P., & Verykios, V. (2006 , November 30). *IEEE Xplore Digital Library.* Retrieved from http://ieeexplore.ieee.org: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=4016511&url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel5%2F69%2F4016508%2F04016511

OOCL. (2016, January). *Duplicate Detection and Resolution.* Retrieved from https://www.oclc.org.

### 2.2 THE CONCEPT DUPLICATE DETECTION

The problem of finding duplicated documents has been a subject of research in the database and web-search communities for some years. The applications range from plagiarism detection in web publishing to redundancy detection in large datasets. The common duplicate detection techniques are classified into two categories: Fingerprint-based and Full text-based.

**2.2.1DUPLICATE DETECTION USING FINGERPRINTS**

A fingerprint of a document is a set of integers, each of which is the hash value for a substring extracted from the document. In this paper, to be clearer in concept, the term “fingerprint” refers only to document-level fingerprint while the term “integer” or “hash value” refers to hash function output, which in many other paper is sometimes also called “fingerprint”. Each integer is stored in an index for fast access during query process. Similarity between two documents is measured by counting the number of common integers. Algorithms are different in their choices of hash functions, substring size, substring number, and substring selection strategy.

1. **Hashing functions**is used to generate hash values for substrings. Popular hash functions include NIST’s SHA1 and Rabin. However, many other hash functions are qualified for this task as long as they are reproducible and with a low rate of hash collision.
2. **Substring size**is defined by the length of each substring extracted from a document. Larger size increases the chance of false negatives in duplicate detection while smaller size increases that of false positives, e.g., SCAM used a very small substring, word, as the unit for fingerprinting. Prior research suggested that substrings of 3-5 words are good.
3. **Substring number**is the number of substrings extracted from a document to build a fingerprint. Some techniques used a fixed number of substrings for efficiency, e.g., I-Match presented in, while many others used a variable number of substrings for a more accurate representation of the document, e.g., DSC presented in. A smaller number of substrings have the risk of ignoring short documents and increasing false negatives.
4. **Substring selection strategy** is the way to pick which substrings to hash. It can be categorized as position-based, hash-valuebased, anchor-based and frequency-based strategies. Position-based strategy selects substrings based on their offsets in a document, a sentence or a paragraph. It includes full fingerprinting, non-overlapping fingerprinting, and overlapping fingerprinting. It is popular due to the simplicity. Hash-value-based strategy is also popular.

The famous shingling approach (or DSC), picks substrings whose hash values are multiples of an integer. Anchor-based strategy extracts substrings that start with special words or character sequences. It was shown to be one of the best substring selection methods; however this approach has to be manually tuned to fit a specific collection and hence is not that practical. Frequency-based strategy selects substrings based on their frequency of occurrences in the document, the entire collection and/or external collections. Term frequency (tf) within a document and inverse document frequency (idf) in a collection are used to select the substrings, used 30-60 highest idf words, and also selected terms with high idf. However term selection based on idf alone can be overly sensitive to small changes in document content and hence the false negative is high. The more recent extended Match used external collection statistics to select the lexicon. It achieved a better recall by introducing multiple fingerprints. However, it is computationally expensive.

**2.2.2DUPLICATE DETECTION USING FULL-TEXT**

The simplest full-text approach is to adapt methods originally developed for search engines, for example, vector-space model, which treats a document as bag-of-words, with term weights determined by tf.idf values, and similarity determined by cosine similarity. Traditional cosine-similarity measure focuses on finding a semantic relevant document while near-duplicate detection focuses more on syntactic similarity. Several previous works thus have been done in finding suitable similarity measures to address syntactic similarity among documents. The identity measure proposed by emphasizes that the gap between rare words’ term frequency in two documents should be smaller than that between common words’ and their best ranking is giving by a term weighting function biased towards rare terms. Metzler et al. used statistical translation models to estimate the probability that one sentence in a document is a translation of another sentence in another document. The probability of aligning to a absent term is estimated by the background language model. The translation probability serves as the basis of the sentence-level and the document-level similarity. (Hui Yang, Jamie Callan, 2005)

### 2.3 CLASSES OF DUPLICATION

Observing pairs of documents with high redundancy score r (u; v) in search engine logs, we found three types of redundant pairs of results:

1. Exact duplicates, where both pages appear identical, perhaps with the exception of advertisements.
2. Content duplicates, where both pages essentially provide the same information with respect to the query, but from different sources.
3. Navigational duplicates, where navigating from one page to the other is very easy. Note that one of the pages may be more relevant than the other.

While exact duplicates do not require examples, we illustrate the others with specific examples.

Examples of content duplicates include two different web sites with lyrics for the same song, two different documents with similar recipes for oatmeal cookies, or two different sites for converting centimeters into inches. While these alternatives may differ in relevance (for example, due to the clarity of presentation), we expect that most users would find either redundant if they have already observed the other. This study will determine whether users in fact behave in a way consistent with this hypothesis.

Note the difference between the definition of content duplicates and the concept of duplicates studied in the literature: content duplication here refers to duplication in terms of information content, whereas duplication pertains to duplication in terms of lexical content. As an example, two different documents with similar recipes for oatmeal cookies can be content duplicates but not duplicates if the documents are describing similar recipes with different words.

Navigational duplicates are different: here one page is very often more relevant than the other, but it is conceivably easier (or less risky) for a user to get to the more relevant result from the less relevant result by browsing than by returning to the web search results. This is often the case when a user clicks on one search result without having considered the next result; if the user expects that the cost of backing out from an almost correct result to find the right result in a search engine ranking is higher than the expected homepage and the sports page of a newspaper, or the online banking login page of a bank and the contact us" page of the same bank. Occasionally, we also observed pairs of results where neither was quite what the user was looking for, but navigating to the correct page from either was equally trivial. (Radlinski, Bennett, & Yilmaz, 2011)

### 2.4 PRE-DUPLICATE RECORD DETECTION PHASE

Detection and removal of duplicate records that relate to the same entity within one data set is an important task in case of the date preprocessing. Data linkage and duplication can be used to improve data quality and integrity, to allow re-uses of existing data sources for future research work. Data processing: In real-world, data tend to be incomplete, noisy and inconsistent. Such situation requires data preprocessing. Various forms of data preprocessing includes data cleaning, data integration, data transformation and data reduction. In other words, the data preparation stage includes data cleaning, data transformation and data standardization. Typically, the process of duplicate detection is preceded by a data preparation stage, during which data entries are stored in a uniform manner in the database. Data cleaning process attempts to fill the missing values, smooth out noise while identifying outliers and correct inconsistencies in the data. Data transformation process converts the data into appropriate forms for mining. Data reduction techniques can be used to obtain a reduced representation of the data while minimizing the loss of information content. Data transformation: Simple conversions applied to the data in order to confirm their corresponding data types also refer to data transformation. This type of conversion focuses mainly on one field at time without any consideration of the values in the related fields. Example of data transformation:

1. Conversion of a data element from one data type to another
2. Renaming of a field from one name to another
3. Range checking, which involves examination of data in a field to ensure that if falls within the expected range, usually a numeric or date range?
4. Dependency checking, involves comparison of value in a particular field to values in another field

Steps in data preprocessing Pre-duplicate record detection phase: The process of standardizing the information represented in certain fields to a specific content format is called as data standardization. This is done to ensure that the information stored in many different ways in various data sources must be converted to a uniform representation before the duplicate detection process starts. Without the standardization process, many duplicate entries could erroneously be designated as non-duplicates. Data standardization is a rather inexpensive step that can lead to fast identification of duplicates. After the data preparation (stage) phase, the data are typically stored in such a manner which easily facilitates for comparison. Dimensionality reduction technique is used for reduction of the dimensionality in datasets could be divided into three classes: Feature Extraction, Feature Selection and Feature Clustering (Subramaniyaswamy.v & Pandian.s, 2012).

### 2.5 OTHER DUPLICATE DETECTION ALGORITHM

1. **Jaccard Similarity:** The Jaccard index, also known as the Jaccard similarity coefficient (originally coined coefficient de communauté by Paul Jaccard), is a statistic used for comparing the similarity and diversity of sample sets. The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets:

(If A and B are both empty or thesame, we define .) for   
.

The MinHash min-wise independent permutations locality sensitive hashing scheme may be used to efficiently compute an accurate estimate of the Jaccard similarity coefficient of pairs of sets, where each set is represented by a constant-sized signature derived from the minimum values of a hash function (Jaccard index, 2011).

1. **Tanimoto Similarity and Distance:** Various forms of functions described as Tanimoto similarity and Tanimoto distance occur in the literature and on the Internet. Most of these are synonyms for Jaccard similarity and Jaccard distance, but some are mathematically different. a "similarity ratio" is given over bitmaps, where each bit of a fixed-size array represents the presence or absence of a characteristic in the plant being modelled. The definition of the ratio is the number of common bits, divided by the number of bits set (i.e. nonzero) in either sample.

Presented in mathematical terms, if samples are bitmaps, is the ***i***th bit of , and *Λ*, *V* are bitwise and, or operators respectively, then the similarity ratio ***Ts*** is:

If each sample is modelled instead as a set of attributes, this value is equal to the Jaccard coefficient of the two sets. Jaccard is not cited in the paper, and it seems likely that the authors were not aware of it. Tanimoto goes on to define a "distance coefficient" based on this ratio, defined for bitmaps with non-zero similarity:

This coefficient is, deliberately, not a distance metric. It is chosen to allow the possibility of two specimens, which are quite different from each other, to both be similar to a third. It is easy to construct an example which disproves the property of triangle inequality (Jaccard index, 2011).

1. **Sorensen-Dice:** The choice of measure depends on the characteristics of the domain to which they are applied. Among many different similarity indexes, the similarity defined in CloneDR is worth notice. Baxter et al. define the similarity between two trees T1 and T2 as follows:

where H is the number of shared nodes in trees T1 and T2, L is the number of unique nodes in T1, and R is the number of unique nodes in T2. Within the context of a tree structure, this definition can be seen as an extension of the Sorensen-Dice index. The Sorensen-Dice index is originally defined by two sets and is formulated as follows:

Here, indicates the number of elements in the intersection of sets and (Yoshihisa, 2013).

1. **Euclidean Algorithm:** Euclid's algorithm, is an efficient method for computing the greatest common divisor (GCD) of two numbers, the largest number that divides both of them without leaving a remainder. It is named after the ancient Greek mathematician Euclid, who first described it in Euclid's Elements (c. 300 BC). It is an example of an algorithm, a step-by-step procedure for performing a calculation according to well-defined rules, and is one of the oldest numerical algorithms in common use. It can be used to reduce fractions to their simplest form, and is a part of many other number-theoretic and cryptographic calculations (Euclidean algorithm., 2015).

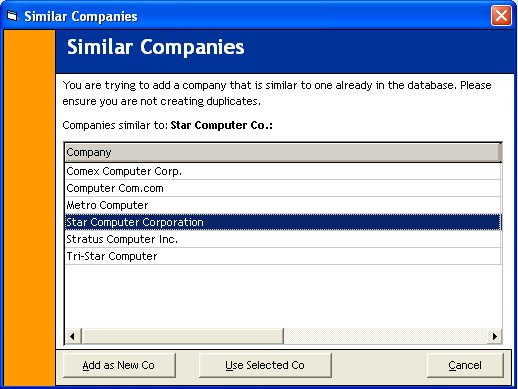
Where n is the number of dimensions (attributes) and and are, respectively, the ***k***th attributes (components) of records p and q.

### 2.6 SIMILALGORITM AND DATABASE DUPLICATE PREVENTION

T-SQL allows us to perform a wide range of text searches. Still, a lot remains to be desired, especially with regard to misspellings. If you want to find a set of records even if they have misspellings, or want to prevent misspellings, you need to perform fuzzy string comparisons, and Simil is one algorithm suited for that task.

One use for Simil is in data cleanup. In one example, a company had a table with organic chemistry compounds, and their names were sometimes spelled differently. The application presents the user with the current record and similar records. The user can decide which records are duplicates, and choose the best one. One button click later, all child records are pointed to the chosen record, and the bad records are deleted. Then the user moves to the next record.

Another typical use for Simil is in preventing bad data from entering the database in the first place. When a salesperson is creating or importing a new company, the application uses Simil to scan for similar company names. If it finds any records, it’ll show a dialog box asking the user if the new company is one of those, or indeed a new company, as shown in figure 2.1



**Figure 2.1:** Use of Simil on Duplicate Prevention

Other uses include educational software with open-ended questions. One tantalizing option the original authors mention is to combine Simil with a compiler, which could then auto-correct common mistakes.

Let us look at Simil in more detail, and learn how we can take advantage of it. In 1988, *Dr. Dobb’s Journal* [published](http://www.ddj.com/184407970?pgno=5) the Ratcliff/Obershelp algorithm for pattern recognition (Ratcliff and Metzener, “Pattern Matching: The Gestalt Approach”). This algorithm compares two strings and returns a similarity between 0 (completely different) and 1 (identical). Ratcliff and Obershelp wrote the original version in assembly language for the 8086 processor. In 1999, Steve Grubb [published](http://web.archive.org/web/20050213075957/www.gate.net/~ddata/utilities/simil.c) his interpretation in the C language. This is the version I used as a starting point for the project implementations I’m presenting here. The purpose of Simil is to calculate a similarity between two strings.

Over the recordings of pattern matching algorithm that ever exists, Simil is highlighted as one or the accurately measured algorithm for checking the identity of string patterns unlike Jaccard and other set driven algorithm which uses set union and intersection rules, Simil check is based on the prefixes and surrogate computation method.

**2.6.1 JUSTIFICATION OF SIMIL ALGORITHM**

Simil has good performance and is easy to understand. It also has several weaknesses, including the following:

1. The result value is abstract. Therefore it’ll take some trial and error to find a good threshold value above which you’d consider two strings similar enough to take action. For data such as company names, I recommend a starting Simil value of about 0.75. For the organic chemistry names, we found that 0.9 gave us better results.
2. It’s insensitive for very small strings. For example, Adams and Ramos have three out of five characters in common, so the Simil value is 0.6. Most people wouldn’t call those names similar.
3. It treats every letter the same, without regard for vowels or consonants, or for letters that often occur together, or for the location in the string, or any other criteria. Some other algorithms do; for example, in the English language the letters Q and U nearly always occur together and in that order, so much so that they could almost be considered a single letter. In a more comprehensive algorithm, such occurrences could be given special consideration. SOUNDEX is another algorithm that does take into account that some consonants are almost the same (for example, *d* and *t*).

Simil cannot be pre-calculated, always requires a table scan, and cannot take advantage of indexes. This may be a problem for large datasets (Tom, 2011).

# **CHAPTER THREE**

## 3.0 RESEARCH AND DESIGN METHODOLOGY

### RESEARCH METHODOLOGY

The approach to establish a research can be Primary; consisting of the native study of the topic under research and chiefly involves first direct methods in the form of questionnaires, surveys, interviews, observations and discussion groups, or Secondary; which requires external approach to research which usually involves perusal of mostly published works like researching through archives of public libraries, court rooms and published academic journals. Considering the nature of this project, the non-empirical approach was taken which is all about visiting existing archives of dictionary book and data matching solution approach and consult a meta-analysis on them to study how word search works.

**3.1.1 THE USE OF ELECTRONIC SOURCE**

In this project, the approach taken is the secondary method but this time majority of the survey was made through online documents downloads from internet archives related to duplicate detection solutions, and the use of string similarity search algorithm (Simil). Some of the downloaded documents includes:

1. Framework for Evaluating Clustering Algorithms in Duplicate Detection, by Oktie Hassanzadeh thesis development team.
2. Efficient Partial-Duplicate Detection Based on Sequence Matching, by Qi Zhang, Yue Zhang, Haomin Yu, and Xuanjing Huang.
3. A replicated study on duplicate detection: Using Apache Lucene to search among Android defects, by Jens Johansson, Markus Borg, Per Runeson and Mika Mantyla.
4. Simil: An algorithm to look for similar strings, by Tom van Stiphout.
5. Simil Is An Algorithm Used To Find Similar Strings – And It Is Awesome, by Dirk Strauss and many more.

### 3.2 DESIGN METHODOLOGY

During the design process, there exist tools preferences for the system development with reasons to justify why these tools are selected. We concluded to use one of the Object Oriented Programming Language called **Java** by installing its development kit **JDK** and **Wamp** database server for **MySQL** database development of the system to store and easily retrieval of data. These two Software tools are interconnected with an object to database connecting tools provided by every java NetBeans IDE known as the **MySQL JDBC Driver** for connecting the Application Interfaces to MySQL Database.

**3.2.1 JAVA PROGRAM:** Java is an object based programming language used to design both system and application software. The major reason behind the choice of java over all other programming language is that it is capable of executing on any system platform, and it filters out memory that are not being used after building and compilation is being done.

**3.2.2 MySQL DATABASE SERVER:** Database is needed for easily storage, retrieval, and update of data-items generally refers to as a repository for data. There are several choices of databases but the one being chosen as a database choice is MySQL due to the luxurious acquisition of data, its flexibility in querying of database, and its nonselective connection to all computer object oriented languages (it is compatible with any object oriented programming language).

**3.2.3 MYSQL.JDBC SERVER CONNECTION:** The MySQL database server is directly interlinked with **com.mysql.JDBC.Driver** which allows the application to connect to the application to access the entire databases that may exist in the MySQL database server of a particular computer system it was added to the application library so as to detect and connect with the MySQL database server of a machine while installed. The architecture below shows the relationship between the software design in java program and the entire MySQL database

Application Design Interface(s) Using Java Programming

**com.mysql.JDBC.Driver** database connection library

WAMP (MYSQL) server data repository Design

**Figure 3.1:** Design Tools Relationship Architecture

### 3.3 HOW SIMIL ALGORITHM WORKS

The Simil algorithm looks for the longest common substring, and then looks at the right and left remainders for the longest common substrings, and so on recursively until no more are found. It then returns the similarity as a value between 0 and 1, by dividing the sum of the lengths of the substrings by the lengths of the strings themselves.

Table 1 shows an example for two spellings of the word Pennsylvania. The algorithm finds the largest common substrings lvan, and then repeats with the remaining strings until there are no further common substrings.

|  |  |  |  |
| --- | --- | --- | --- |
| **Word 1** | **Word 2** | **Common substring** | **Length** |
| Pennsylvania | Pencilvaneya | Lvan | 8 |
| Pennsy    ia | Penci    eya | Pen | 6 |
| nsy    ia | ci    ey | A | 2 |
| nsy    i | ci    ey | (none) | 0 |
| Subtotal |  |  | 16 |
| Length of original strings |  |  | 24 |
| Simil = 16/24 |  |  | 0.67 |

**Table 1:** How Simil Algorithm work on two strings (Pennsylvania and Pencilvaneya).

Simil is case sensitive. If you want to ignore case, convert both strings to uppercase or lowercase before calling Simil.

At its core, Simil is a longest common substring or LCS algorithm, and its performance can be expected to be on par with that class of algorithms. Anecdotally, we know that using Simil to test a candidate company name against 20,000 company names takes less than a second (Tom, 2011).

### 3.4 DATABASE RECORD DUPLICATE DETECTION APPLICATION

The software Database Record Duplicate Detection App is an application program designed solely to interact with MySQL database server connecting to all database that may exists in the server database but limited to those that is not secured with a password. The application is designed on a single interface which consists of two tabs to cover the major activity of the designed app. The first tab is designed to operate on the duplicate detection and in the process generate a log in a text file format to store the operation performed by the application user during the duplication. On the other side of the tab is a front end table whish shows the record stored in the database table to be optimized. While on the other tab, the application provide a platform to retrieve stored log for reference purpose.

Simil Algorithm takes effect while the button check duplicate is triggered. The algorithm uses the record comparison between the short range of 0.95 and absolute value of 1 on each field that a record has to the other succeeding record’s field in the selected database for optimization. The optimization takes effect with the action to delete duplicate files if and only if user request the optimization action. The architecture below shows the summary of how the designed application operates.

Database Record Duplicate Detection Application

Send Table for optimization

MySQL Database Server

Request a database table

Delivers table on Request

Create a Log File and Prepare to Save

Store Log File in Application Folder

**Figure 3.2:** Database Record Duplicate Detection Architecture

### 3.5 DATABASE RECORD DUPLICATE DETECTION FLOWCHART

Begin

Display the Application Interface

Are there duplicate records?

Execute the Database Duplicate detection app

Choose a database from MySQL database server on the system

Execute System’s MySQL database server

Choose a table from the selected database to check duplicate

Display list of records in the table and generate a log file

Click optimization button to optimize duplicate records

Save generated log file

End

Exit Application

**Figure** 3.3: Database Duplicate Detection Software Flowchart.

# **CHAPTER FOUR**

## 4.0 IMPLEMENTATION AND RESULT

### 4.1 SYSTEM REQUIREMENTS

The application is working effectively and supports some system requirements which are basically divided into the hardware and the software requirement as listed below:

**4.1.1 HARDWARE REQUIREMENTS**

The following are the some hardware specifications to support using the designed application:

* 512MHz or Higher Intel Premium or AMD Processor.
* 256Mb Memory (RAM) or Higher.
* VGA 800 x 600, 256 color Minimum.
* Hard Disk Storage of 60 GB Minimum.

**4.1.2 SOFTWARE REQUIREMENTS**

* A 32 or 64bit Windows Operating Systems (OS) or any other OS that support the use of Java Runtime Library.
* Reliable and licensed Antivirus software like Avast, AVG, or any system security shield.
* MySQL Database Testing (Xampp, Wamp, Lamp, etc.) with port set to 8080 connecting to system address (127.0.0.1 or localhost) and not shared by any others app.
* Java Software Development Kit.

### 4.2 SYSTEM DOCUMENTATION AND USAGE

Database Record Duplicate Detection usage and settings in a naked computer system requires some instructional procedures which appears to be simple and straight forward. To make it swift and detailed, these procedures will be listed and explain below but following the settings procedure at first list and the usage as next list.

The software settings requires the following steps:

Copy the file dbrdds.zip to the required Computer System.

Unzip the file and install the wamp server that exists in the unzipped pack, or may decide to download the latest version for use and ensure that it is properly working.

Once again, open the wamp server control panel and access its phpmyadmin page.

Create several databases with any names and populate them with several tables for testing**.**

Open the unzipped file once again and install the application DBRDDS pack.

The software usage requires just few steps which is listed below:

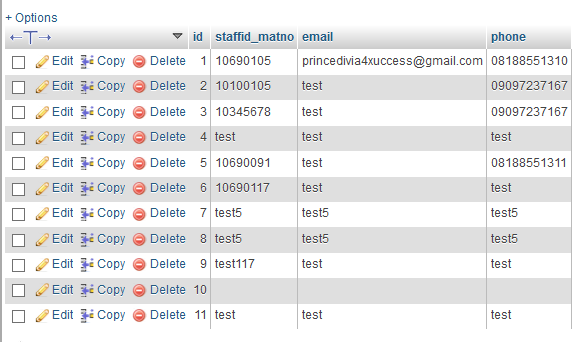
Run the installed Application.

Test the application on created databases and tables.

Save Progresses for reference purpose.

### 4.3 TESTED DATABASE SCREENSHOTS FROM MYSQL SERVER

Although the application does not need much of database explanation nor does it require a specific database to execute in the database server, it is designed to connect directly to the server and communicate to all existing databases that may resides in the server. But nevertheless, a table for a database called **department.signup\_tbl** was tested on the software and the two screenshot below in figure 4.1 and figure 4.2 shows the record of the table before and after the application testing was made:



**Figure 4.1:** department.signup\_tbl records before Application testing (before optimization).

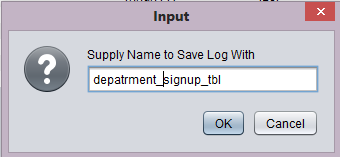
After the selected database table is optimized, the duplicate records in the table will be reduced to one distinct records each as a resolution to the presence of redundant records in the database table. Figure 4.2 shows the lists of records in table department.signup\_tbl at the end of the duplicate detection and optimization process.



**Figure 4.2:** department.signup\_tbl records after Application testing (after optimization).

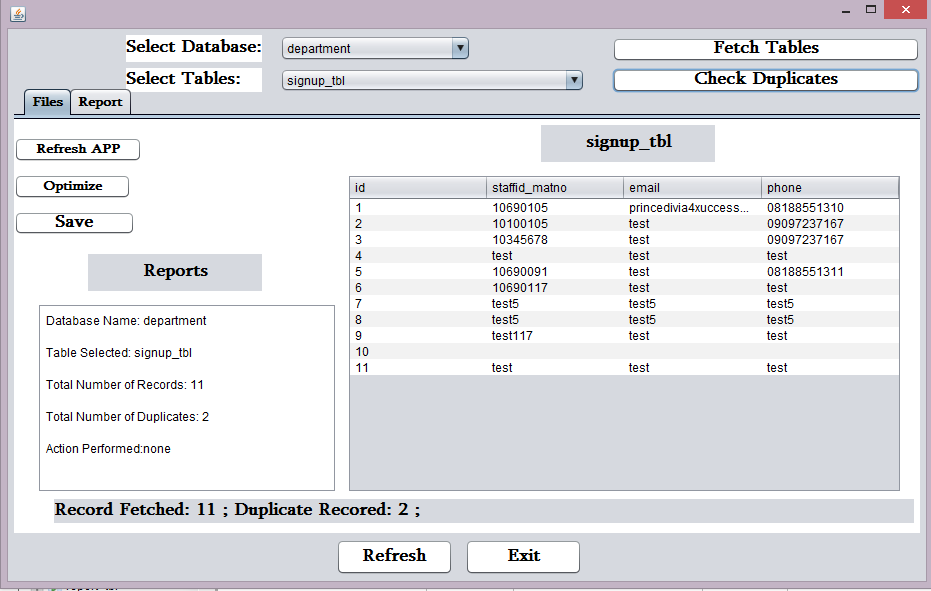
### 4.4 DATABASE RECORD DUPLICATE DETECTION SCREENSHOTS

The input dialog below in figure 4.3 is responsible for preparing the application system logs at first installation and launching.



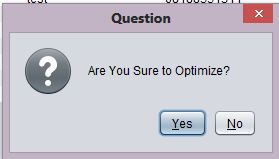
**Figure 4.3:** App log input dialog.

The figure 4.4 below shows the design screenshot of the application project the application’s behavior and results before the optimization takes place, the processes conducted in this figure are: launching the main application from start, select a database to optimize from the lists of displayed databases in the database dropdown options provided by the application interface, clicking on the fetch table button to gain access to all the available tables stored in the application database, selecting a table to detect the presence of duplicate records in and the check duplicate button for listing out the records on the system with status representing the number of the entire records and detected duplicates.



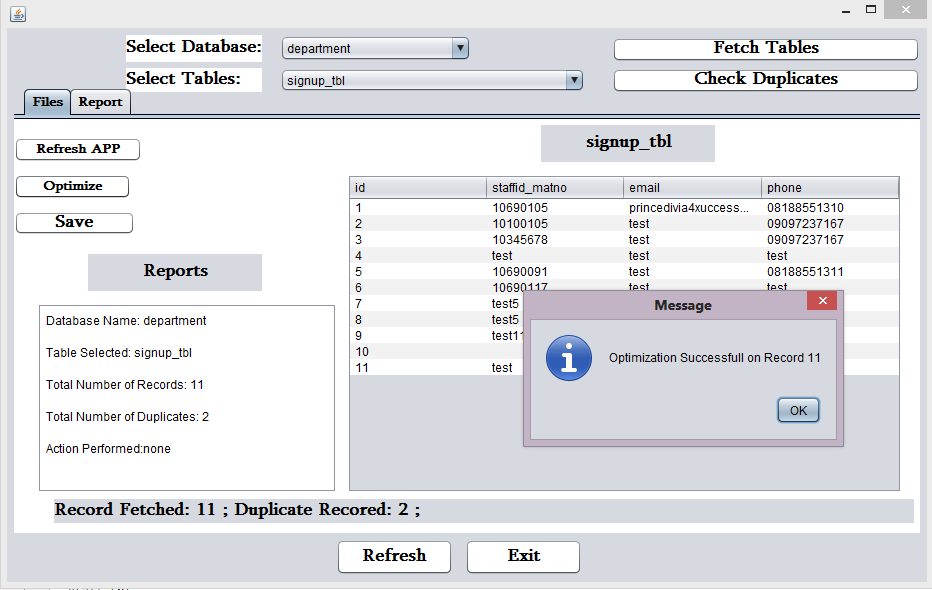
**Figure 4.4:** Database duplicate detection software screenshot.

The figure 4.5 below is confirmation dialog requesting user to optimize the detected duplicate records in which if the response is Yes, the optimization takes place but nothing is done on the selection of No option.



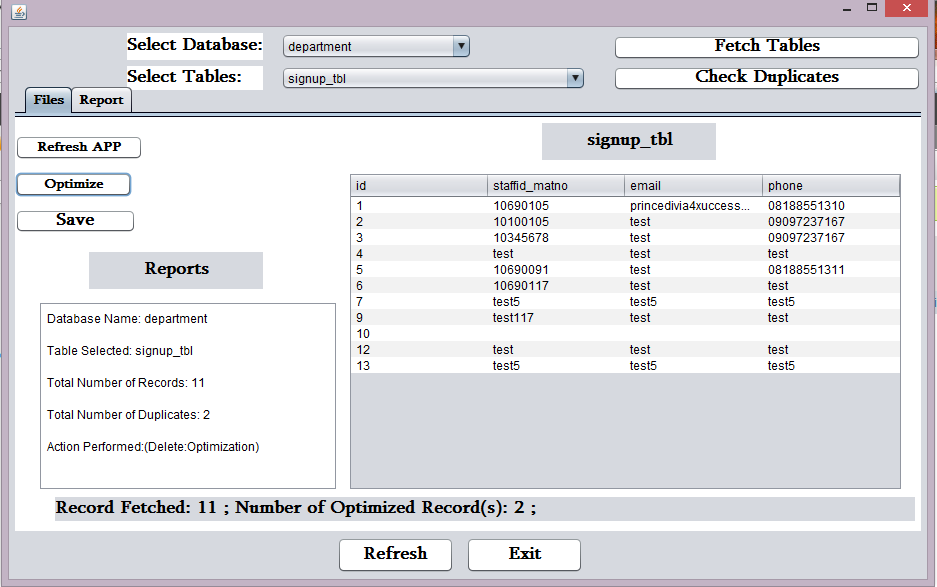
**Figure 4.5:** Duplicate detection optimization confirm dialog.

The screenshot in figure 4.6 below displays an additional message dialog notifying the application user of the success of a concluded optimization that took place on a selected database table which helps to eliminate duplicate records from the table keeping the original as the project optimization is defined.



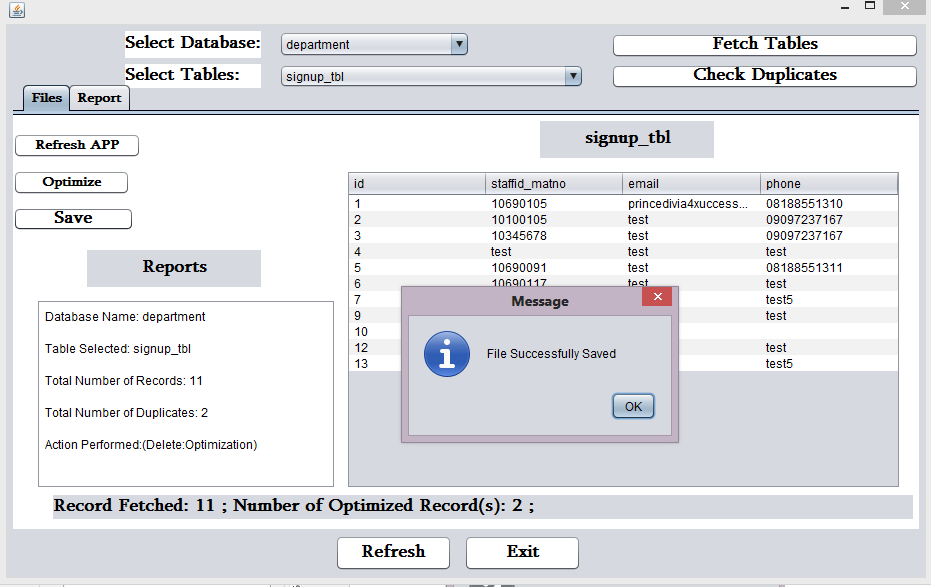
**Figure 4.6:** Application screenshot after optimization.

The screenshot below in figure 4.7 represents the list of records as the newly optimized record stored in the application database after conducting the optimization on previously fetched duplicates which serves the aim of redundant records elimination.



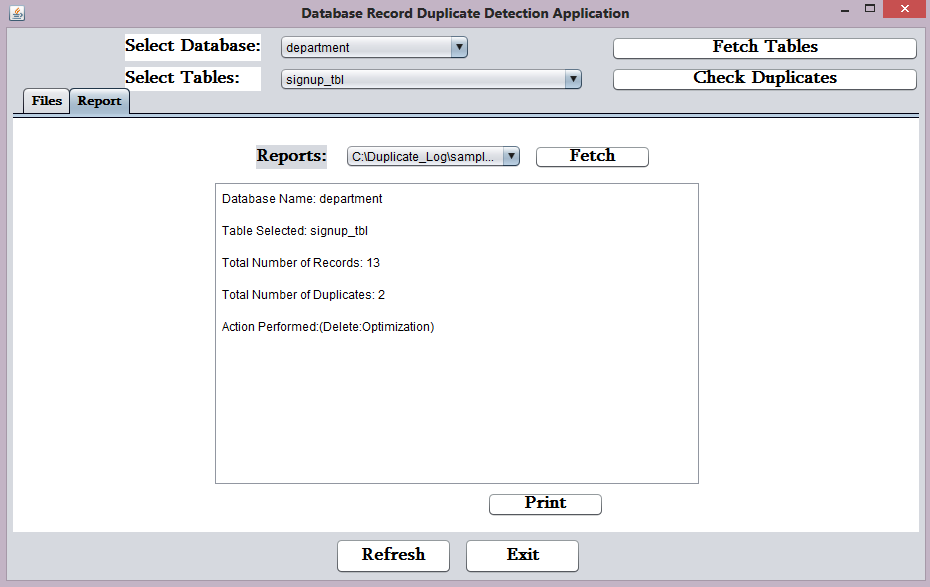
**Figure 4.7:** Screenshot of optimized version

The figure 4.8 screenshot below represent the application display after saving the optimization process log into the application log directory which projects a message dialog at save success.



**Figure 4.8:** Screenshot of saved optimization log.

The screenshot in figure 4.9 below shows the section of the software interface where saved regressive logs can be retrieved and displayed which is selected through the application’s report tab panel and dropdown box and the tab also provides a print button for user to be able to communicate with a system printer hardware.



**Figure 4.9:** Report interface for duplicate detection software.

# **CHAPTER FIVE**

## 5.0 CONCLUSION AND RECOMMENDATIONS

### 5.2 CONCLUSION

Duplicate detection is different from other Information Retrieval (IR) tasks in how it defines what it means for two documents to be “similar”. In many IR tasks document similarity refers to semantic “relevance” among documents, which are could be syntactically very different but still relevant. In contrast, the definition of similarity in duplicate detection in early database research is very conservative, which is mainly to find syntactically “almost-identical” documents, for other tasks that need to detect documents with “intermediate level of similarity”, there has not been much research done.

This project is implements a matching algorithm in a duplicate detection. Using a matching algorithms such as Simil, the search for duplicate records in a database table requires checking the similarity match of each fields that a record composes in the database using a close range of 0.8, 0.9, or 1.0 similarity of alike records under the similarity space of 0.0 and 1.0. Data matching algorithm in general has gone a long way supporting in several areas of need, ranging from redundancy optimization, throughout the level of pattern verification, to the concentrated length of diagnostic level and others, and it will be a continual useful in the progressive buildup of information technology.

### 5.1 RECOMMENDATIONS

All the way from the buildup process of this project, it will be of a great impact if the use of Data Matching Algorithm should much more encouraged if already applicable in several fields like: Reduction of Files and Data Redundancies, Database Record Deduplication, Duplicate Optimization Checking, Medical Diagnostic Field, User’s Access Authorizations, Criminal Investigations, and other fields, to aid in catalyzing their operation process because they have proven to be a very useful and numerical result oriented to for expressing comparatives between files and patterns such as Simil Check, for it to be useful in database record duplicate detection, it has a very long way to go in other areas of need as mentioned.

# **REFERENCES**

Elmagarmid, A., Ipeirotis, P., & Verykios, V. (2006 , November 30). *IEEE Xplore Digital Library.* Retrieved from http://ieeexplore.ieee.org: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=4016511&url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel5%2F69%2F4016508%2F04016511

Fred, D., Jason, V., & John, T. M. (2002). REDUNDANCY ELIMINATION WITHIN LARGE COLLECTIONS OF FILES.

Hui Yang, Jamie Callan. (2005, August 6–11). Near-Duplicate Detection by Instance-level Constrained Clustering. pp. 4-5.

OOCL. (2016, January). *Duplicate Detection and Resolution.* Retrieved from https://www.oclc.org.

Radlinski, F., Bennett, P. N., & Yilmaz, E. (2011). *Detecting Duplicate Web Documents using Clickthrough Data.* Cambridge.

Subramaniyaswamy.v, & Pandian.s, C. (2012, june 05). A COMPLETE SURVEY OF DUPLICATE RECORD DETECTION USING DATA MINING TECHNIQUES.

Tom, v. S. (2011, April 10). *Simil: An algorithm to look for similar strings.* Retrieved from accessmvp: http://www.accessmvp.com/tomvanstiphout/simil.htm

Yang, H., & Callan, J. (2006, August 6–11). Near-Duplicate Detection by Instance-level Constrained Clustering. pp. 2-3.

Balinski, J., & Danilowicz, C. (2005). Re-ranking method based on inter-document distance. Information Processing and Management, 41, 759–775.

CrossRef.

Bear, J., Israel, D., Petit J., & Martin D. (1997). Using information extraction to improve document retrieval. Proceedings of TREC.

Chen, K., Chen, H., Kando, N., Kuriyama, K., Lee, S., Sung, H., et al. (2003). Overview of CLIR task at the third NTCIR workshop. Proceedings of NTCIR III.

Crouch, C., Crouch, D., Chen, Q., & Holtz, S. (2002). Improving the retrieval effectiveness of very short queries. Information Processing and Management, 38, 1–36.