# Data Analytic Tools for Inconsistency Detection in Large Data Sets

Design Document

Team 27
Client - Kingland
Advisers - Cai Ying
Team Members/Roles -
Logan Heitz (Team Lead), Camden Voigt (Technical Lead),
CJ Konopka (Communication Lead), TJ Rogers (QA Lead)
Team Email - sdmay18-27@iastate.edu
Team Website - http://sdmay18-27.sd.ece.iastate.edu/
Revised: December 4, 2017 Version 2

# Table of Contents

# 1 Introduction

## 1.1 Acknowledgement

This project would not be possible without the assistance of the faculty advisor Dr. Cai. Working with Dr. Cai on this project are two graduate students Guolei Yang and Zehua Li who have provided invaluable in design and implementation of this project. Finally this project relies on the support of Kingland Systems for testing data and any other needed materials for implementation of the project.

## 1.2 Problem and Project Statement

Kingland processes a large amount of data that it receives from its clients everyday. This data can be relating to customers, companies, or agreements between entities. This data is compared to a central inconsistency database in order to detect inconsistencies and then added to the database. An example of an inconsistency would be two customer records containing the same social security number, but different names. This is an issue, since a social security number should be unique. The database contains over 100 million records, and around 10% of these records are updated or inserted daily. Due to its size, this comparison takes several hours to run every day. This time stems from the fact the entire database cannot be loaded into main memory at one time and the use of SQL inner join statements to check for inconsistencies, which is inefficient. Kingland would like to process 100 million records for inconsistencies in an hour or less. Additionally this detection must begin with the latest version of the inconsistency database after the reports come in.

## 1.3 Operational Environment

Our product will operate on the backend of Kingland's system and will be automated to detect inconsistencies on incoming reports. Thus, the product will need to be able to operate with minimal user input and will need to generate results that can be integrated into Kingland's existing infrastructure.

## 1.4 Intended Users and Uses

This project will supply Kingland's analysts with information on inconsistencies within their records. The product will be on the backend of Kingland's system and its processes will be automated. As such no users will directly interact with our system on a day to day passes as Kingland will display the results of our output using their own user interface. However, if Kingland wishes to improve the system in the future or needs to fix something their developers will need access to the code and documentation on how it works. Thus, it is important to provide material for future developers on this project.

## 1.5 Assumptions and Limitations

### 1.5.1 Assumptions

- There will be an inconsistency database containing more than 100 million historical reports
- The end product shall not require a user interface
- The product will only need to detect equality comparisons
- The inconsistency database will be periodically updated with new data

### 1.5.2 Limitations

- The program will not be able to be tested on the full sized dataset
- The program cannot be tested with all possible configurations
- Program will be deployed on a machine with less than 64 GB of RAM

## 1.6 Expected End Product and Deliverables

- System architecture of proprietary solution
    - Delivery Date: 01/20/2018
    - This deliverable will encompass the design of the proprietary solution that will be developed to solve this problem. This deliverable will be expanded on in the design document and will involve the overall system block diagram, UML class diagrams, and class documentation.

- System implementation of proprietary solution
    - Delivery Date: 02/20/2018
    - In addition to the architecture of the proprietary solution an implementation of the solution will be developed in java. This

implementation will be provided to the client for use in their daily inconsistency checking.

- Analysis of proprietary solution against industry standard solutions
    - Delivery Date: 03/20/2018
    - There are many standard industry solutions that could be utilized to solve this problem. Following the implementation of the proprietary solution the team will test the solution to determine its average runtime along with the detection rate of inconsistencies. The team will perform similar analysis of standard industry solutions and provide the findings to the client so they might evaluate which solution is best for their needs.

- Test cases of solutions
    - Delivery Date: 04/02/2018
    - All test cases that are used for the solution will be provided for the client so they might verify the solution is valid using the test cases. They will also be able to utilize the test case if further development is needed for the project.

- User manual
    - Delivery Date: 04/02/2018
    - A user manual that will discuss how to set up the application and automate its processes. This will include documentation on how to set up the configuration file for their needs. It will also include how they can incorporate the outputs of the application with their user interface.

# 2. Specifications and Analysis

## 2.1 Specifications

### 2.1.1 Functional Requirements

- Solution must not use SQL inner-join statements
    - Kingland’s current solution to this problem is to use SQL inner-join statements which can take a long time. Thus, our solution should eliminate these statements to save time.

- Solution must utilize only relevant information
    - Our solution needs to run using only the small amount of fields needed to actually detect an inconsistency. This will reduce memory utilize and speed up the detection.
- Solution must compare current records to previous records as well as other current records
    - Our solution needs to be able to compare inconsistencies between records found in new reports and also between new reports and previously saved records.
- Solution must validate all fields are present in data
    - Our solution needs to ensure that all required fields are in each received record.
- Solution must handle various forms of input
    - Our solution should be able to handle data input in multiple formats including XML and JSON.
- Solution must update inconsistency database after analysis
    - Our solution should update an inconsistency database so that future checks will work with the latest version of the database.

#### 2.1.2 Non-Functional Requirements

- Solution must perform inconsistency check in less than 1 hour for daily reports
    - Our solution should be able to detect and report all inconsistencies in a new report in an hour or less.
- Solution must be able to analyze 100 million or more records at a time
    - New reports can have 100 million records or more. Therefore, our solution should be able to handle this size of input.
- Solution must run on Kingland's system
    - Our solution is a proprietary solution for Kingland and therefore must be able to run on their hardware.
- Less than 5% false positives
    - Our solution should have less than 5% false positive inconsistencies to reduce the time spent fixing these.

## 2.2 Proposed Design

Our proposed solution to this problem is to create a proprietary system that will utilize hashing to speed up comparisons and to reduce the memory required by the database. Hashing will improve the speed of comparisons since we only need to do equality checking. Therefore, values can simply be compared after they are hashed to see if they are equal to the value in the database. This allows us to eliminate the current use of SQL inner join statements, which are time consuming, in favor of lookups on indexed columns. Indexing the entries in the database by the columns that are important for equality comparison allow us to quickly lookup information. Our solution also reduces the space of the table as we will only need to store the hash values which can be smaller than the original values and only stores attributes necessary for inconsistency detection. This reduces the table size and will allow more or all of it to be loaded into main memory at once.

First, we will create a configuration file format. This file will specify how we process the received daily reports. We will then create a configuration parser that will turn this file into a configuration object. The configuration object will be utilized by the raw data parser to accept the daily reports that Kingland receives. As each item in a report is parsed it will be sent to the inconsistency matcher. By sending the parsed data one element at a time we will not have to bring all of them into main memory at once, which will improve the performance of our program. The inconsistency matcher will check the element against the inconsistency database. If there is an inconsistency, then the element will be sent to an inconsistency report. Otherwise it will be sent to the exporter which will add it to the inconsistency database. An overview of the different modules used in the project can be found in *Figure 2.1*.

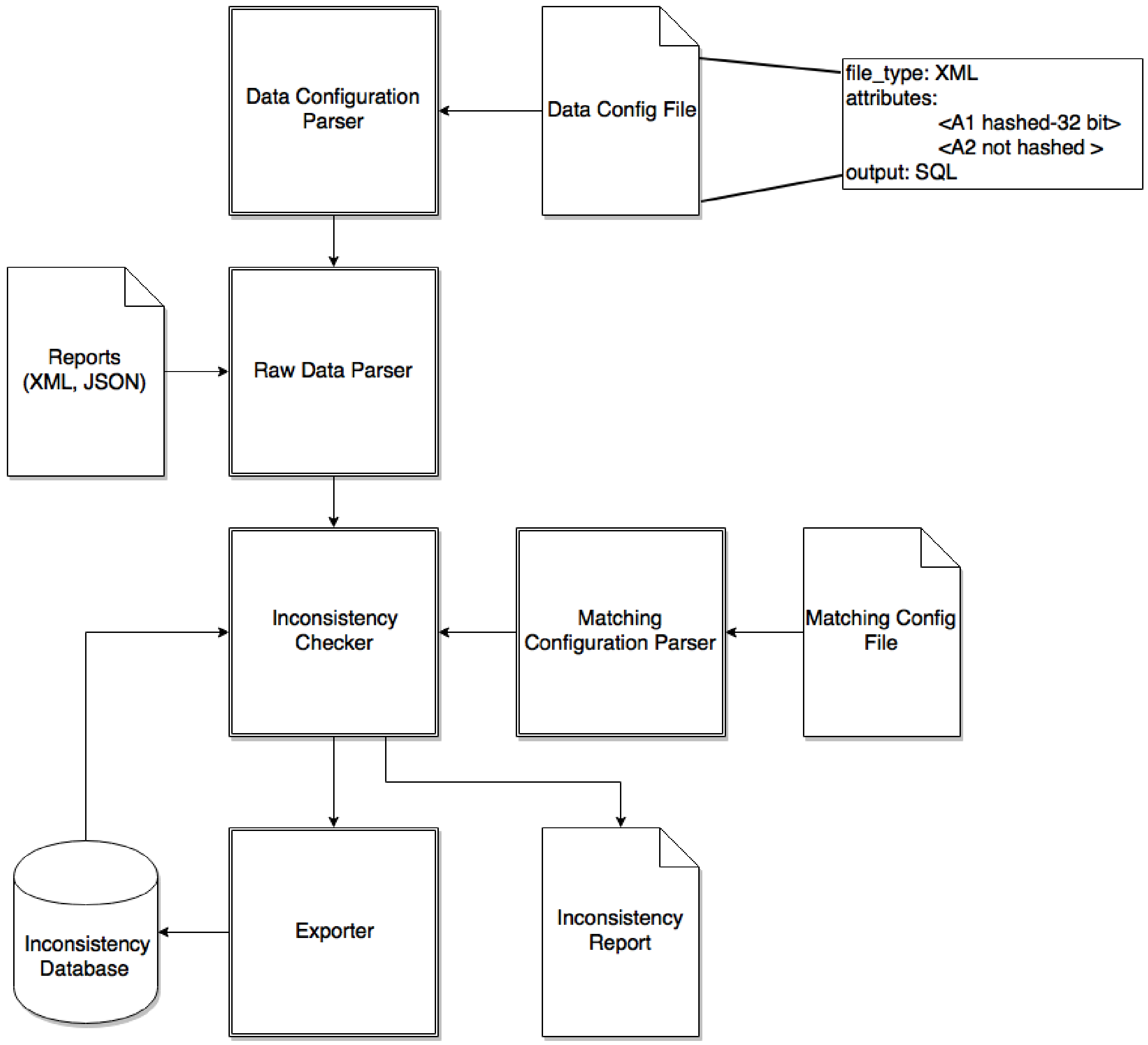

*Figure 2.1*: Block diagram of high-level system architecture.

### 2.2.1 Module Design

This section will present an analysis of the various modules that will be needed for this project.

**Data Configuration File**

The configuration file will be an XML file with three major sections. The first of these is the input path which will take the path to either a file or a folder. *Figure 2.2* shows how the inputpath should be specified if it is a file or a folder.

```
<inputpath>/path/to/file.xml</inputpath>
<inputpath>/path/to/folder</inputpath>
```

*Figure 2.2*: Example XML input tags

If a file is specified this will be used as the report and if a folder is specified it will utilize all files within the folder. This allows for easy processing of multiple reports.

The second section is the exports. This section will specify all the ways to output the processed data. The output can be either a file type or a database. Both will require a location to be defined while a database will also require a driver, username, and password. An example of the exports section from the configuration file can be seen in *Figure 2.3*.

```
<exports>
      <export type="database">
            <location>http://localhost:3306/database</location>
            <driver>mysql</driver>
            <username>myusername</username>
            <password>mypassword</password>
      </export>
      <export type="file">
            <location>~/path/to/file.xml</location>
      </export>
</exports>
```

*Figure 2.3*: Example XML export tag

Finally, there will be a key element section that will contain information on all the attributes that we will need from the records contained in the report. This will include what key the attribute has along with a list of other keys the attribute may be called that are equivalent. It will include how the attribute should be outputted which is raw data and/or hash value. *Figure 2.4* gives an example of how the key-elements will be formated in the configuration file.

```
<key-elements>
      <element key="ssn">
```

```
              <output-format>binary</output-format>
              <output-format>md5</output-format>
         </element>
         <element key="name" optional-keys="firstname,lastname">
              <output-format>raw</output-format>
              <output-format>binary</output-format>
              <output-format bits="128">hash</output-format>
         </element>
    </key-elements>
```

*Figure 2.4*: Example XML elements tag

## Data Configuration Parser

The configuration parser will have several components, the first of which is the configuration parser class. This class will take in a data configuration file and output a configuration object for the class by parsing over the XML. The configuration object will be a separate class that stores the a list of export objects, a list of element objects, and the input path. An element object will contain a key value, a list of alias keys, and a list of output format objects. The output format object will include an output type, the number of bits if hashing is used, and the hash function that should be used if applicable. An export object will store the export format, file or database, and the location to export to. If it is a database, the object will also store the driver, the username, and the password. The UML diagram in *Figure 2.5* shows the different classes and methods that will be included in this module.

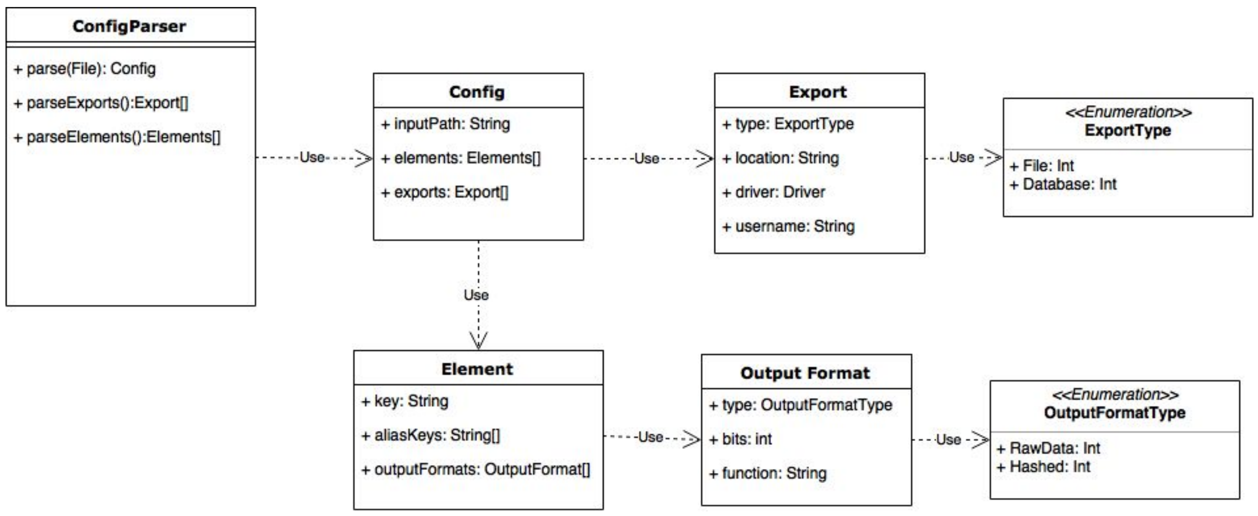

*Figure 2.5*: Configuration Parser UML class diagram

## Raw Data Parser

The raw data parser will utilize the configuration object that is created by the configuration parser in order to parse the records within the daily reports received by Kingland. The parser will parse through the XML of the input file and utilize the output formats defined within the element objects in order to determine how the elements should be outputted. As it parses through the records it will send them each to the inconsistency checker one at a time.

## Inconsistency Configuration File

The inconsistency configuration file will be used to specify what inconsistency the program should detect. This will give a list of inconsistency in XML and will have several attributes for each inconsistency. The inconsistency status is the status code associated with that inconsistency. The index is the value that should be used to pull records from the database as the corresponding columns will be indexed in the table. The compare attribute is the value that needs to be compared.

```
<inconsistencies>
      <inconsistency status=”101” index=”TID” compare=”NAME”>
      <inconsistency status=”201” index=”SSN” compare=”NAME”>
</inconsistencies>
```

*Figure 2.6*: Example inconsistencies XML

## Matching Configuration Parser

This module will function similar to the data configuration parser and will parse over a matching configuration file to get the information it contains. It will create an inconsistency object for each inconsistency listed in the file and the list of these inconsistencies will be given to the inconsistency checker.

## Inconsistency Checker

The inconsistency checker will compare incoming records against existing records within the inconsistency database to find inconsistencies. This will be down by utilizing the information in the inconsistency configuration file. When the checker gets a new record it will go through each defined inconsistency and search in the database for records with the same index value as the current record. It will then compare the compare attribute with that of the incoming record and verify equality. If they are not equal, then the record will be added to the inconsistency report with the status code defined for that inconsistency and the conflicting record in the database. Otherwise the record will be sent to the exporter to be inserted into the database.

**Exporter**

The exporter will accept records from the inconsistency checker that has been found to be consistent with existing data. It will then either update the record in the database or insert it if it is a new record.

**Inconsistency Database**

The inconsistency database is used to store the hashed or raw data values, as needed, for inconsistency detection. The columns of this database will correspond to the keys and formats that are specified in the configuration file.

**Inconsistency Report**

The inconsistency report will be a text file containing the records that were found to be inconsistent, the status code of the inconsistency, and the records that they conflicted with in the database.

## 2.3 Design Analysis

### 2.3.1 Specification Fulfilment

The design laid out in above fulfills the specifications laid out in section 2.1 through a variety of methods. The use of a configuration file allows us to handle various forms of input and to utilize only the relevant information by any given scan by allowing Kingland to specify those things before the program runs. Creating a hashed database allows us to get rid of SQL inner-join statements because we can just do simple equality comparisons. It also lets us do inconsistency checking in less than one hour because we can do equality checks very quickly. Also, by using a good hashing function we are able to only have a few conflicts and therefore only a few false positive marks. Finally, our design allows ours programs to easily access the inconsistency database used in inconsistency detection.

### 2.3.2 Strengths

This solution has several strengths that make it an appropriate choice for this problem. The first strength is the ease of implementation. This solution is built on a few small parts that can be implemented with relative ease. This will give more time to analyze the proposed solution against the existing solution and other industry solutions to determine its viability. Another strength of this solution is it is very modular. The solution is separated out into several distinct parts and changing one part will not require changes to the entire design. The solution also allows for us to quickly adapt to changes in the problem statement. The design of the configuration file allows for us to quickly add in new inconsistencies if needed and modify the settings of current ones to adapt to any changes we encounter. Finally, the configuration file also allows us to testing much easier as it can be utilized to output to multiple different types of database or files so they can be tested against each other.

### 2.3.3 Weaknesses

This solution comes with a few tradeoffs including having to duplicate data into another database and the possibility of not being able to detect every type of inter-record inconsistency. Both of these shortcomings are small issues. While usually duplicating data isn't a great solution, in this case the duplicated data will actually take up less space than the original and only needs to updated as often as the original data. Also, not being able to solve every inconsistency isn't a huge issue as even if we can only solve a large amount of these issues it would still reduce the time needed to run a inconsistency scan significantly.

A more significant issue with this solution is the potential for false positive detections of inconsistencies. Since the values we compare will be hashed there is a possibility of collisions. This is a tradeoff of speeding up the system that is deemed acceptable. As a analyst will need to go through the flagged inconsistencies in any case it will be a simple matter for them to mark it as a false positive and move on.

The biggest shortcoming of our proposed solution would be that a third party solution may be able to do this job almost as well. In this case it would almost be easier for Kingland to use this third party solution as it would have better support from a full development team, and could be used in some of Kingland's other solutions.

# 3 Testing and Implementation

## 3.1 Hardware and software

We will be utilizing several software testing libraries to verify that the requirements for
our product are met. The libraries we will be using are as follows:

- JUnit- A unit testing framework designed for the Java programming language.
- Mockito - Allows programmers to create and test double objects (mock objects) in automated unit tests for the purpose of Test-driven Development (TDD) or Behaviour Driven Development (BDD).

## 3.2 Modeling and Simulation

This problem will be modeled with test data obtained from Kingland that will provides a look at how reports will be structured and how much data can be expected to arrive each day. Once the prototype of the project has been created it will help facilitate simulation of many different potential final implementations. Since our project will utilize a configuration file that will specify how to output and format the data we will be able to quickly change the settings for different test runs. This will allow us to get data on many different methods of implementation, such as what kind of data base we will want to use.

## 3.3 Process

### 3.3.1 Functional Testing

| Requirement | Validation/Acceptance test |
|---|---|
| Solution must not use SQL inner-join statements | A Style Checker will be used to ensure that SQL inner-join statements do not appear in the production code. |
| Solution must utilize only relevant information | The size of the Inconsistency database created by this solution shall be compared to Kingland's central database to determine if this requirement is satisfied by our solution. |
| Solution must compare current records to previous records as well as other current records | Using Kingland's current solution as an Oracle to determine if our solution detects the same inconsistencies. |
| Solution must validate all fields are present in data | A unit test will be used to confirm that any missing fields in the raw data are flagged as such. |

|  |  |
|---|---|
| Solution must handle various forms of input | Unit tests will be used with multiple forms of sample data to determine how well the parser can handle different configurations of input data. |
| Solution must update inconsistency database after analysis | Kingland's main database will be checked to verify that it has been updated with the information that has been verified as consistent. |

*Table 3.1*: Validation tests for the Functional requirements

### 3.3.2 Non-Functional Testing

| Requirement | Validation/Acceptance test |
|---|---|
| Solution must perform inconsistency check in less than 1 hour for daily reports | We will compare performance log files gathered in Log4J to determine the success of this. |
| Solution must be able to analyze 100 million or more records at a time | This will be validated using actual data Kingland receives on a daily basis and success will be determined based on whether or not our solution can perform faster and with the same accuracy as Kingland's system. |
| Solution must run on Kingland's system | We will work with Kingland to deploy our solution on their machine to test its performance. We will also attempt to test on machines with similar specifications to Kingland's. |
| Solution reports less than 5% false positives | This will be tested by processing inconsistency files and determining an average number of false positives. |

*Table 3.2*: Validation tests for Non-Functional requirements

## 3.4 Implementation Issues and Challenges

We expect to run into a few issues and challenges while implementing our design. First, we need to have hardware that can hold the large amounts of data that we will be getting which could be 100GB or larger. This could pose a problem when using personal computers that won't have enough memory to hold both the raw data and all of the copied data. To solve this challenge we have set up a dedicated server, with the assistance of our advisor, in order to do full scale testing. Modular testing can still be conducted on personal machines on segments of the test data.

We may also run into problems with our chosen programming language, Java. Java is for easy development and ensures that our program can run on any system. Java can also be slow and CPU heavy because it runs in the Java virtual machine. We believe java is worth the risk at this point in time but we may change programming languages if our early prototypes are too slow.

Our implementation has to be heavily adaptable. It is very possible that the input to a solution may be in various file formats and with differing organization. Additionally, the inconsistencies we could potentially check for are numerous and dynamic. To mitigate these issues we have created configuration files and parsers. That way the end user will not have to edit the actual code at all, only the configuration files, to get different functionality.

## 3.5 Implementation

The configuration files have been completely implemented and allow users to specify the file location of an XML file or a directory containing many XML files along with the data the user would like to check for inconsistencies and a location that the evaluated data should be exported to. The data that is parsed from the XML files is currently very dependent upon the format of the XML file and may cause challenges if Kingland receives XML files with formats that vary drastically because of the way the key-value pairs are stored. Log4J was also added to monitor the time it takes to parse the XML data, validate it and then export it. After The files have been parsed, the data is can be stored in either a database or a text file as the parsed data, or as a hashed form of it in order to cut down on the size of the data.

## 3.6 Results

Currently, the project is still in the prototyping stage. Each part has been built out and the project can currently run in a development environment with a specific setup and data, but the project as a whole may still be subject to many changes. This instability is making it hard to use our full testing plan on the project so far. Even so, we have been able to use Log4J (our logging library) to start testing how long the project will take to run. The preliminary results of these timing tests show that on a personal machine comparing 2 GB of data to a database with 550,000 entries our project about takes about 3 minutes to complete and finds 3847 inconsistencies.This look good for our initial prototype, but we think there are still improvements to be made. Especially in the time spent updating the database.

# 4 Closing Material

## 4.1 Conclusion

Our solution for this problem will utilize hashing to speed up table lookups and reduce table sizes. This will allow us to speed up the time needed to find inconsistencies within the daily reports received by Kingland. We will develop test cases for our solution in order to verify its performance and accuracy. Testing of other solutions used within the industry will also be done in order to benchmark the performance of our solution. This information, along with our solution, will be provided to our client so they might determine what solution will best suit their needs.

## 4.2 References

Haufler, Andreas. “Conveniently Processing Large XML Files with Java.” *Dzone.com*, 10 Jan. 2012, dzone.com/articles/conveniently-processing-large.

*This article explains how using SAX as an alternative to a DOM parser reduces the amount of memory needed to handle large files because SAX invokes callbacks to detect XML tokens instead of loading the entire file into memory. It also shows example uses of the SAX parser.*

Murnane, Tafline. “ISO/IEC/IEEE 29119 Software Testing.” *ISO/IEC/IEEE 29119 Software Testing Standard*, softwaretestingstandard.org, 24 Oct. 2013, www.softwaretestingstandard.org/part4.php.

*ISO/IEC/IEEE standards for specification-based, structure-based and experience-based testing techniques. Along with outlining a variety of testing technique this document outlines some standard methods used to derive test cases that have been approved and adopted as international standards for software testing.*

Smrcka, Ales I., Ph.D. "TEST PLAN OUTLINE (IEEE 829 Format)." *IEEE 829 - Standard for Test Documentation Overview*. Brno University of Technology, n.d. Web.

*An outline for a test plan document that pinpoints important areas to address when creating a test plan. The template also provides a description of each area so that the reader knows exactly what purpose that area of the test plan addresses.*

Staveley, Alex. “JAXB, SAX, DOM Performance.” *Dzone.com*, 31 Dec. 2011, dzone.com/articles/jaxb-sax-dom-performance.

*This article analyzes a performance evaluation of a JAXB, SAX and DOM XML parser. It provides the source code the tester used to perform the tests as well as the various times required for each xml parser with various sizes of data. For the largest data set, the SAX parser performed the parsing operations in the least amount of time, and has a much smaller memory usage than that of a DOM parser. The drawback to the SAX parser was the amount of developer attention needed to calibrate the parser to perform the desired operations.*

Sug, Hyontai. "An Efficient Method of Data Inconsistency Check for Very Large Relations." S International Journal of Computer Science and Network Security 7.10 (2007): 166-69. Web. 22 Sept. 2017.
*This defines a strategy to help detect inconsistencies in databases based on functional dependencies between attributes in a relation and apply an association rule algorithm based on the attribute sets. It is also assumed that the database under observation is not designed with much consideration about normalization.*

Zhang, Du. (2013). Inconsistencies in big data. Proceedings of the 12th IEEE International Conference on Cognitive Informatics and Cognitive Computing, ICCI*CC 2013. 61-67. 10.1109/ICCI-CC.2013.6622226.

*This article examines four types of inconsistencies in big data - temporal, spatial, text and functional inconsistencies - and how categorizing the inconsistencies in data can help to improve the quality of big data analysis.*

## 4.3 Appendices

### 4.3.1 List of Figures

### 4.3.2 List of Tables