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: October 15, 2017 Version 1

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 Kingland's central 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 a social security number should be unique, as such, detecting such inconsistencies is important to Kingland's clients. 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 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 central 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 a central 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 central 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: 01/20/2017

  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: 02/20/2017

  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: 03/20/2017

  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: 04/02/2017

  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: 04/02/2017

  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

- Doesn’t use SQL inner-join statements.
- Utilize only relevant information.
- Compare current records to previous records as well as other current records
- Validate all fields are present in data
- Handles various forms of input
- Update central database after analysis

2.1.2 Non-Functional Requirements

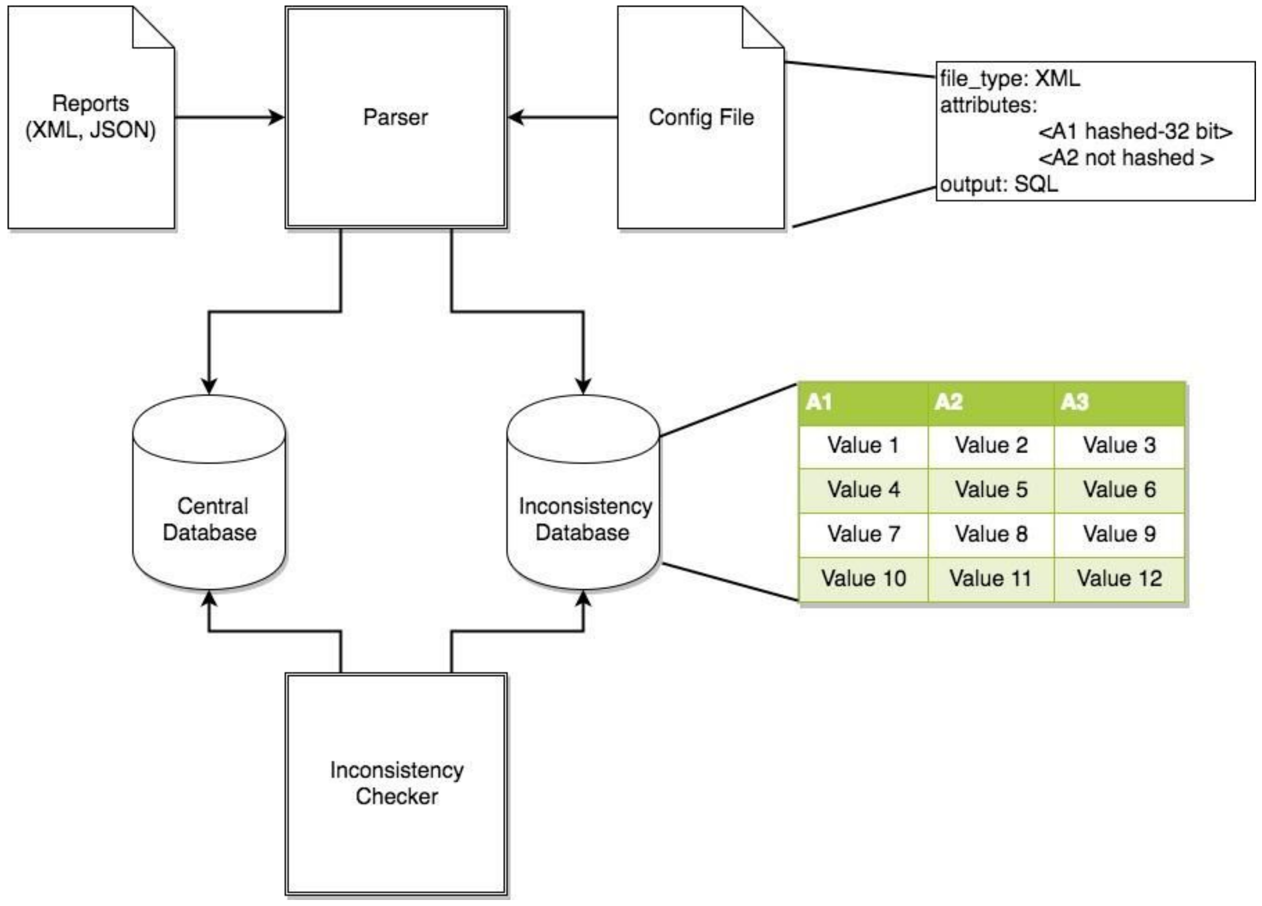
- Perform inconsistency check in less than 1 hour for daily reports
- Analyze 100 million or more records at a time
- Solution should run on Kingland’s system
- Less than 5% false positives

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 as we only need to do equality checking. Therefore values can simply be compared after they are hashed to see if they are equal. This allows us to eliminate the current use of SQL inner join statements in favor of lookups on indexed columns. By index the entries in the database by the columns that are important for equality comparison we can 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 those attributes that are necessary for inconsistency detection. This reduces the table size and 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 the we should process the daily reports received. We will then create a parser that will turn this file into a configuration object.Then we will create a parser which is a program that accept new data sent to Kingland in either XML, JSON, or some other format. The parser will run through this data and convert send it into the database as specified by the configuration object. The parser may also update the central database which holds all records in their full state. Next, we will run an inconsistency checker which goes through either the inconsistency database using SQL lookups on the indexed columns and recognizes conflicts. Once a conflict has been found the inconsistency checker will go to the central database and update the correct records with an error code. An overview of the different modules in 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.

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.

Figure 2.2: Example XML input tags

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

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*.

Figure 2.3: Example XML export tag

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

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.

Figure 2.4: Example XML elements tag

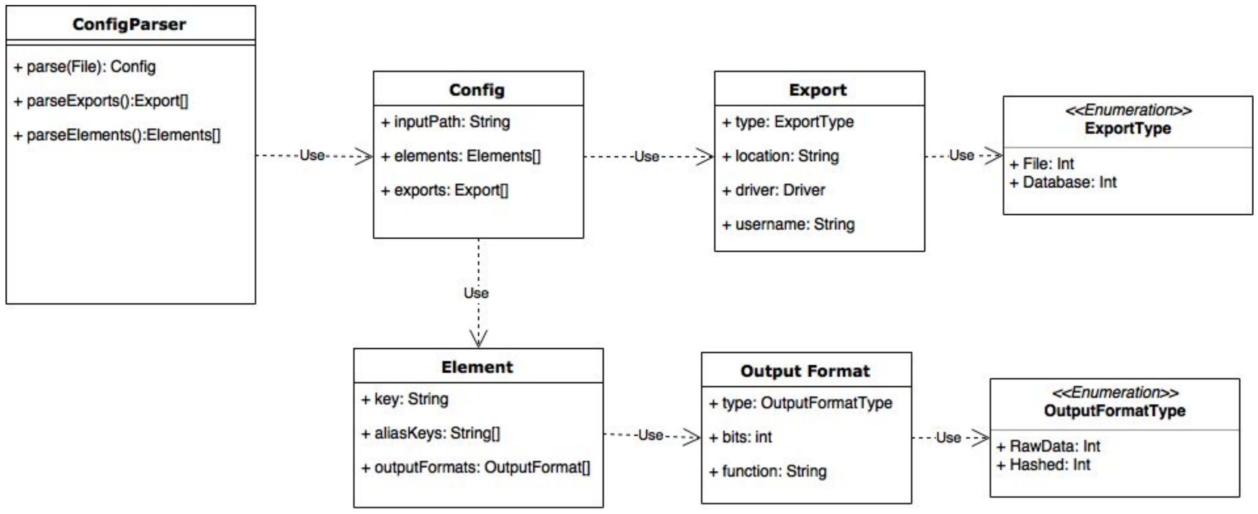
```
<key-elements>
      <element key="ssn">
```

```
                <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>
```

Configuration Parser

The configuration parser will have several components, the first of which is the configuration parser class. This class will take in a configuration file and output a configuration parser 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. The 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. The 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, username, and 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 config object that is created by the configuration parser in order to parse the records within the daily reports received by Kingland. The parse 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. It will than connect to a database or file to output the records.

Inconsistency Checker

The inconsistency checker will compare incoming records against existing records within the inconsistency database to find inconsistencies. This will be down by doing equality comparisons on the hashed or raw data values stored in the table that could have inconsistencies. An example of an inconsistency would be two different records containing the same social security number, but different names. This would be found by checking the table for the SSN and then comparing the name of those two records. The checker will then set a flag within the central database that an inconsistency with the record was detected.

Inconsistency Database

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

Central Database

The central database will be a database that contains all of the information from the reports. This database will not utilize hashing to reduce the table size. This will allow us to reference the real values of elements as needed. When an inconsistency is detected in the inconsistency database it will flag the corresponding reports in this 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 both the central database and the hashed database which makes comparing comparing and updating both very easy.

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).
- Arquillian - Allows developers to easily create automated integration, functional and acceptance tests for Java. Arquillian also allows the developer to run tests in the run-time so you don't have to manage the run-time from the test(or the build). Arquillian can be used to manage the lifecycle of the container (or containers), bundling test cases, dependent classes and resources. It is also capable of deploying archives into containers and executing tests in the containers and capturing the results to create reports from.

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

- Doesn’t use SQL inner-join statements.
- Utilize only relevant information.
- Compare current records to previous records as well as other current records
- Validate all fields are present in data
- Handles various forms of input
- Update central database after analysis

3.3.2 Non-Functional Testing

- Perform inconsistency check in less than 1 hour for daily reports
- Analyze 100 million or more records at a time
- Solution should run on Kingland’s system
- Less than 5% false positives
- Check performance of each class

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 300GB 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.

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.

3.5 Results

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

4.3 Appendices

4.3.1 List of Figures