
TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

CENTRAL CAMPUS, PULCHOWK

**MID-TERM PROGRESS REPORT**

**ON**

# TEST DATA GENERATOR

By:

**Binod Shrestha / 068/BCT/513**
**Dipesh Bhatta / 068/BCT/516**
**Shashwat Mehta / 068/BCT/543**
**Yashasvi Pant / 068/BCT/547**

A PROJECT WAS SUBMITTED TO THE DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING IN PARTIAL FULLFILLMENT OF THE REQUIREMENT FOR THE BACHELOR'S DEGREE IN ELECTRONICS & COMMUNICATION / COMPUTER ENGINEERING

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

LALITPUR, NEPAL

AUGUST 18, 2015

## ACKNOWLEDGEMENTS

We would like to express our deepest gratitude towards the Department of Electronics and Computer Engineering, Central Campus, for providing us with the wonderful opportunity, encouragement and environment for this project. We are also grateful to our supervisor Prof. Dr. Subarna Shakya for guiding us in each step for the progress of this project.

We are truly thankful to Deerwalk Services Pvt. Ltd. for their faith upon us and motivation for carrying out this project. Our sincere thanks also goes to our mentor Er. Dinesh Amatya, a Senior Engineer at Deerwalk, for his tremendous support and motivation.

Last but not the least, we would like to thank all our teachers and colleagues for their direct and indirect help related to our project.

Any kind of suggestion or criticism will be highly appreciated and acknowledged.

**Team Members**

Binod Shrestha (068/BCT/513)

Dipesh Bhatta (068/BCT/516)

Shashwat Mehta (068/BCT/543)

Yashasvi Pant (068/BCT/547)

## ABSTRACT

Software testing is one of the most important phases in software development life cycle. In today's world, testing costs 1/3$^{rd}$ of the total project expenditure. Almost all test data generation methods for any domain today are manual which is very hectic, time consuming and inefficient. This project is a smart test data generator, developed for Deerwalk's data analytics engine, Makalu. The system will be accessible for two agents: Admin and User. Admins can set up and configure particular domain by defining fields and constrains on the domain. The user can get the generated test data under a defined domain as per his need. The generation is done by processing a simple input query and producing test cases as required, with consideration of all boundary conditions, constraints and all logical anomalies. Later, he may use the system's components to explore the generated data.



***Keywords:*** *test data generation, makalu, domain configuration, generation query, data exploration*

# TABLE OF CONTENTS

## LIST OF ABBREVIATIONS

| | |
|---|---|
| **ANTLR** | ANother Tool for Language Recognition |
| **CSV** | Comma Separated Value |
| **DOB** | Date of Birth |
| **IDE** | Integrated Development Environment |
| **Inc.** | Incorporated |
| **JDK** | Java Development Kit |
| **JSON** | JavaScript Object Notation |
| **JVM** | Java Virtual Machine |
| **Ltd.** | Limited |
| **POSIX** | Portable Operating System Interface |
| **Pvt.** | Private |
| **SQL** | Structured Query language |
| **TDD** | Test Driven Development |
| **UI** | User Interface |
| **US** | United States |
| **WORA** | Write Once Run Anywhere |

## LIST OF FIGURES

# 1. INTRODUCTION

## 1.1 Background

Software testing is one of the indispensable parts of software development in today's world. Proper testing of the software during and after the development phase can save lot of efforts, time and money. Software testing requires proper test data that can meet all the boundaries and constraints. Not only valid data but we also require invalid data for the testing of software. Considering the fact, our project 'Smart Test Data Generator' can serve as an important part in software development life cycle.

'Test Data Generator' takes a domain configuration file and makes the user input a simple query as per the requirement and generates thousands of valid as well as invalid sets of data within a few seconds.

We have been approached to work at Deerwalk Services Pvt. Ltd. as interns to build the software after the company find our senior batch project 'SmarTest' inefficient, incomplete and difficult to fit into their requirement. We will overcome the limitations of our senior batch project and try to generalized it so that our software come handy not only to the company but to all the kind of software development process as a whole.

## 1.2 Problem Statement

In today's world of software development, almost all test data generation methods for any domain are manual. Currently, Deerwalk's data analytics product, Makalu, uses the same manual method for test data preparation which is very hectic and time consuming.

Even though, there are some traces of automatic test data generation software in the world today, they are not smart enough to generate data that looks completely real and the fields aren't associated with each other either.

Another motivation for the development of this project was the severe limitations in current system – 'SmarTest', that was developed by our seniors. The limitations rendered the product unusable for Deerwalk, which is why a more reliable, accurate and smart data preparation software was needed - 'Smart Test Data Generator'. 'Smart Test Data Generator' takes a domain configuration file and makes the user input a simple query as per the requirement and generates thousands of valid as well as invalid sets of data within a few seconds.

### 1.3 Objectives

The objectives of this project would be,

- To develop a query parser and analyzer to generate quality test data quickly, efficiently and accurately.
- To improve the efficiency of the entire testing process by speeding up the preliminary data manipulation processes.
- To detect and satisfy all the mentioned boundary conditions for datasets.
- To implement the data blinding in the test data provided from the client.

### 1.4 Scope of Work

In today's world, testing costs 1/3rd of the total project expenditure. Almost all test data generation methods for any domain are manual. In a software development life cycle, manual software testing can be so expensive that it is not effective to continue it in the future. So, an automation has to be introduced in software testing process to reduce software development cost and still manage to find as much bugs. Moreover, automated test case generation will further help us cover all the test cases required for the testing process which can be missed in case test cases are generated manually. Hence our product 'Smart Test Data Generator' can be valuable in production of every software and can replace the traditional way of software testing and quality assurance.

Currently, the software is being demanded by Deerwalk Inc. for testing their product Makalu (US based health data analytic software).

### 1.5 Feasibility Analysis

#### 1.5.1 Technical Feasibility

Technically our system is very feasible. It requires minimum hardware and software setup to establish our system into the organization. We are building our query language and GUI in such a way that even ones with very little technical knowledge can use it easily with the help of a simple documentation.

#### 1.5.2 Economic Feasibility

Currently, the manual test data preparation process in deerwalk is very hectic and time consuming. It directly affects the economic aspects of the organization. With the introduction of our system, the time and manpower of the whole organization will reduce dramatically causing direct economic benefits. Moreover, the system increases the efficiency to carry out work in best possible way from economic point of view.

It is estimated that, with our system the data generation and validation time will decrease by approximately 90%. Testing currently requires about one third of the overall project expenditure. With our Test Data Generator, this cost of any of the projects in Deerwalk, can be brought down to one tenth of the original expenditure, thus increasing the efficiency and decreasing the overall cost of any of its projects.

### 1.5.3 Operational Feasibility

Since we are designing a web based system, it can operate in any platform. It is totally Operating System independent. Also operating our system need only one time glance at a documentations. Since, our final system will pass through different testing, verifications and validations, it will have high operational accuracy and efficiency. On the other hand, the data blinding techniques will ensure the operational security. Indeed, the user friendly interface and query language will ensure operational easiness to our system.

### 1.5.4 Legal Feasibility

Our system is totally compliance with the current Data Protection Acts (most importantly HIPPA act). Our system will not take, operate or modify any private data. Also, the data blinding feature of our system will ensure the maximum security and respect to the privacy of the individual.

### 1.5.5 Schedule Feasibility

We have done detail studies and planning about the schedule feasibility of our project from the beginning of the project phase. We are updating it time to time to meet our requirements as well as time limits. We are completing all the work in the estimated times with very less tolerance factor. Our work has been on par regarding the time constraints provided by Deerwalk as well as our major project deadlines.

# 2. LITERATURE REVIEW

## 2.1 ANTLR

ANTLR (ANother Tool for Language Recognition) is a powerful parser generator for reading, processing, executing, or translating structured text or binary files. It's widely used to build languages, tools, and frameworks.

From a formal language description called a grammar, ANTLR generates a parser for that language that can automatically build parse trees, which are data structures representing how a grammar matches the input. ANTLR also automatically generates tree walkers that you can use to visit the nodes of those trees to execute application-specific code.

ANTLR has contributed to the theory and practice of parsing including:

- linear approximate lookahead
- semantic and syntactic predicates
- ANTLR Works
- tree parsing
- LL(*)
- Adaptive LL(*) in ANTLR v4

## 2.2 Grails

Grails is an open source web application framework that uses the Groovy programming language (which is in turn based on the Java platform). It is intended to be a high-productivity framework by following the **"coding by convention"** paradigm, providing a stand-alone development environment and hiding much of the configuration detail from the developer

It integrates smoothly with the JVM, allowing the developer to be immediately productive whilst providing powerful features, including integrated ORM, Domain-Specific Languages, runtime and compile-time meta-programming and Asynchronous programming.

Grails is a full stack framework and attempts to solve as many pieces of the web development puzzle through the core technology and its associated plugins. Included out the box are things like:

- An easy to use Object Relational Mapping (ORM) layer built on Hibernate
- An expressive view technology called Groovy Server Pages (GSP)
- A controller layer built on Spring MVC

- An interactive command line environment and build system based on Gradle
- An embedded Tomcat container which is configured for on the fly reloading
- Dependency injection with the inbuilt Spring container
- Support for internationalization (i18n) built on Spring's core Message Source concept
- A transactional service layer built on Spring's transaction abstraction

### 2.3 Existing Data Generation Tools

Currently there are some applications that implement Data generation. However, most of them generate data randomly without consideration of boundary conditions, constraints and validations. Such data doesn't look anything like real data and they are not accurate for testing purposes.

### 2.4 SmarTest

For the shortcomings of existing data generation tools, our predecessor project 'SmarTest' tried to address those problems. The system could basically be described by the architecture below,

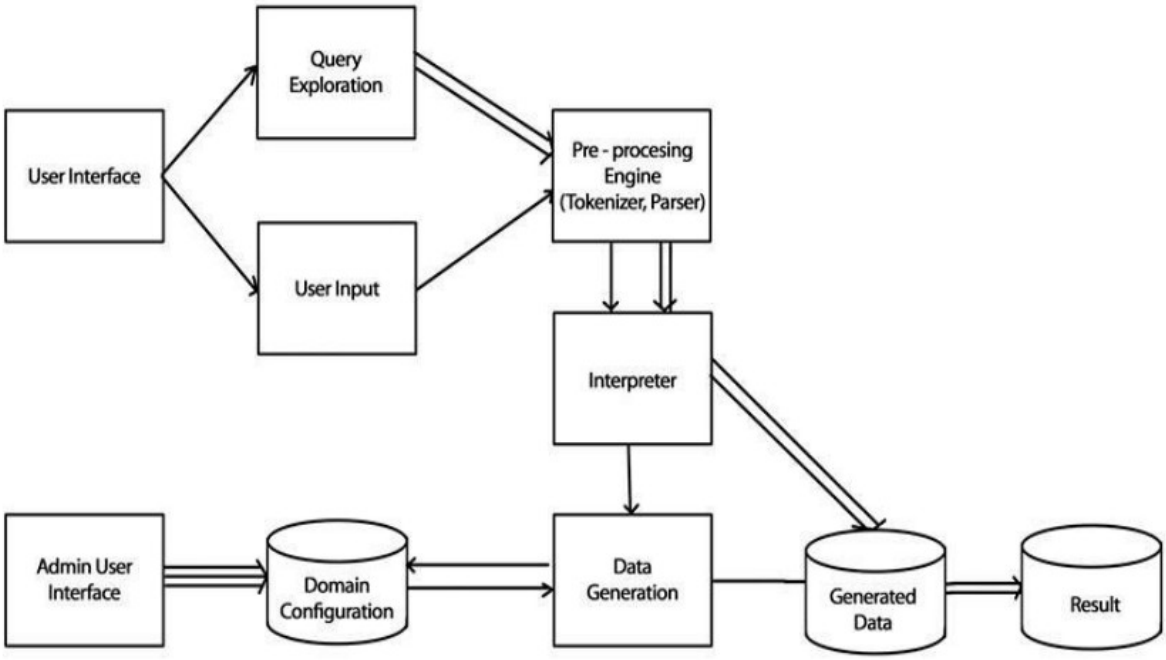


**Figure 1: SmarTest Architecture**

There are basically two kinds of roles in the software: Admin Input and User Input.

The admin is responsible for creating the domain configuration. The domain configuration contains all the knowledge about a particular domain.

On the other hand, the task of user is to generate data for different available domains. The user gives a simple query like business logic to generate the data satisfying the logic.

Components like Lexer, Parser, Interpreter, Generator and Explorer are used to evaluate the business logic, parse them, give required test data and explore them.

However, there are severe limitations in this system – 'SmarTest'. The limitations rendered the product unusable for Deerwalk, which is why a more reliable, accurate and smart data preparation software was needed.

Following would be shortcomings of SmarTest,

i. There is no facility to configure domain relating two or more fields, tables and domains.
ii. The system has not supported enough business logic to represent all types of data manipulation that a SQL supports.
iii. Data randomization technique is not effective. Same data occurs frequently.
iv. There is no proper data validation and exploration to validate valid as well as invalid data.
v. There is no facility for data blinding. Data blinding is necessary to those data that come from real client so that testing on those data could be done without any privacy issues.
vi. User interface is not user friendly. A new user must go through user documentation first to be able to understand the application.
vii. The system, having to run as a desktop app is not really portable and usable across machines in synchrony.
viii. The application does not support output of various file formats. Only CSV (comma separated value) file format is supported.
ix. Only one date format is available.
x. Special data types like currency, temperature, geographical values are still manipulated as numeric data type.

# 3. METHODOLOGY

## 3.1 System Architecture

The overall system architecture is shown in the figure below,

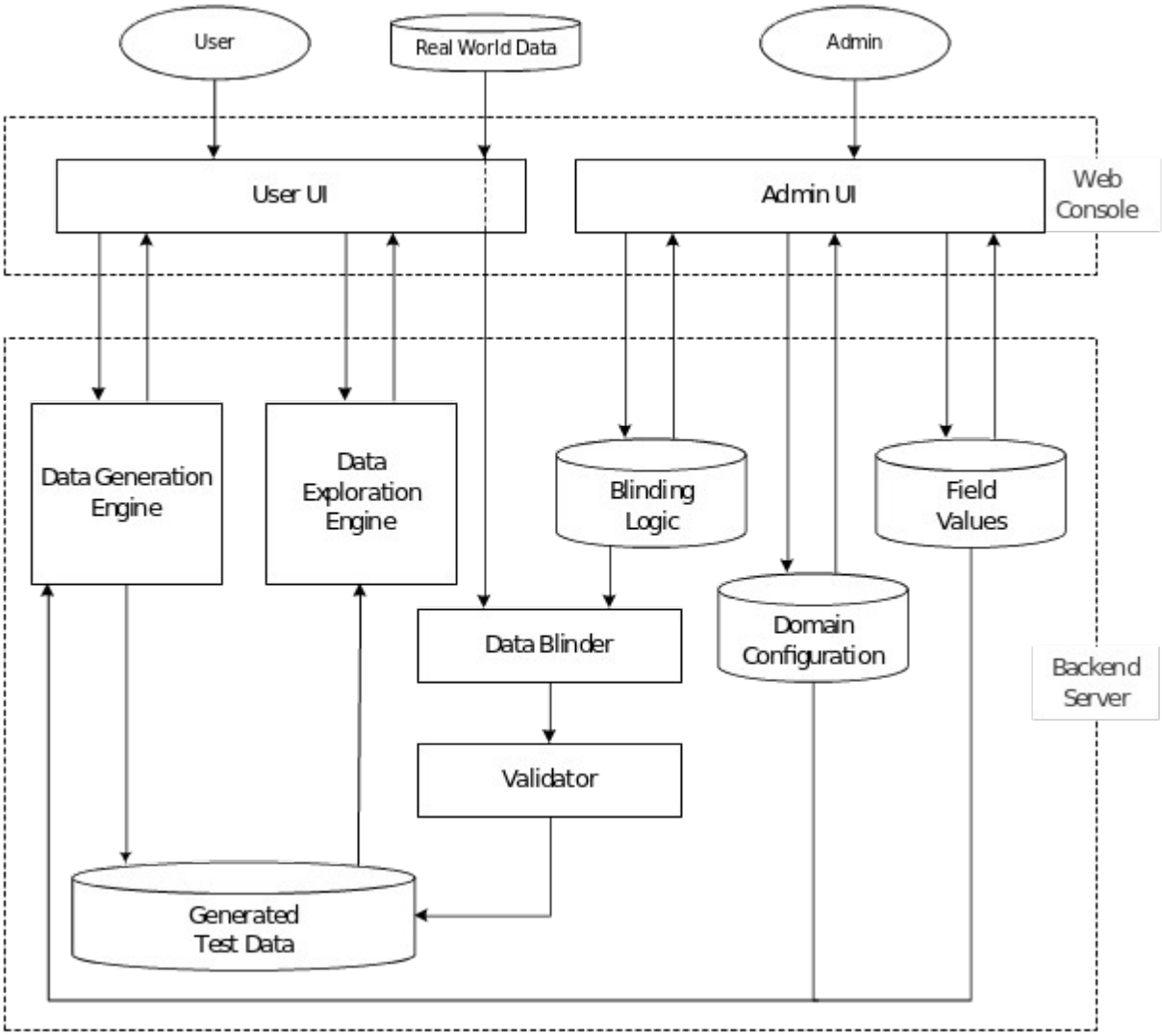


**Figure 2: System Architecture**

The user and admin access the system via web based interface. The backend, which is written using Grails, implements all the functionalities including all the processing, configuration and database maintenance. The database, in essence, refers to the domain configuration files, possible field values, blinding logic files and of course, the generated test data. The data generation and exploration engines are the chief processing components of the system.

### 3.1.1 Data Generation Engine

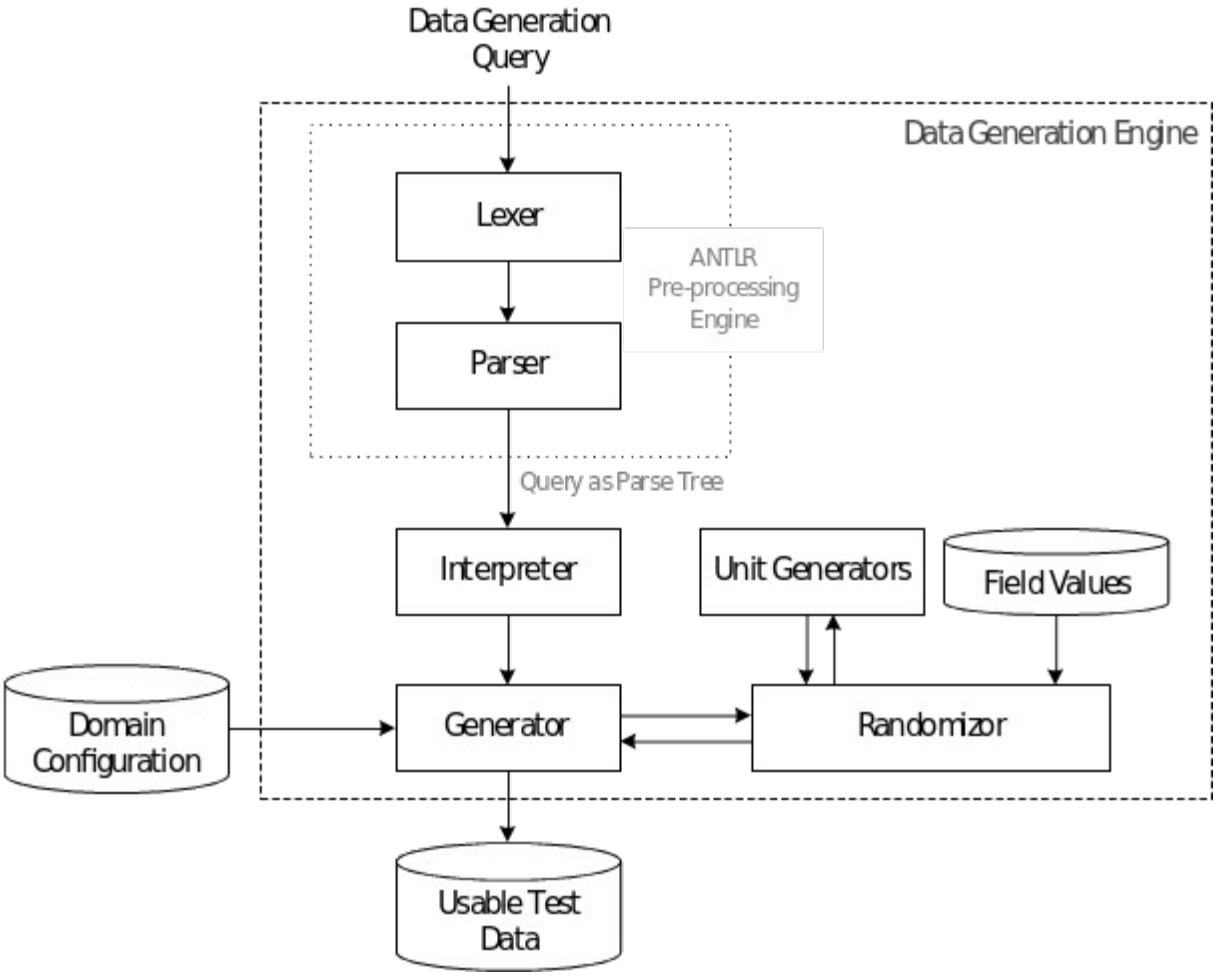


**Figure 3: Data Generation Engine**

As one can see, the data generation engine takes in *generation query* as the primary input. The input is received from the user. The query is first passed through the preprocessing engine written using ANTLR. The output of this is a well formed parse tree that can then be explored. This means finding out and implementing the data generation techniques corresponding to the meanings presented in the parse tree. The randomizer makes sure that same data (that satisfies the condition) is not used every time. In fact, even for subsequent generations, different data are generated each time. The unit generators refer to specific functions that generate numbers, dates and strings that comply to given condition and constraints. The final output of the engine is the test data which is ready to be used for exploring or testing.

### 3.1.2 Data Exploration Engine

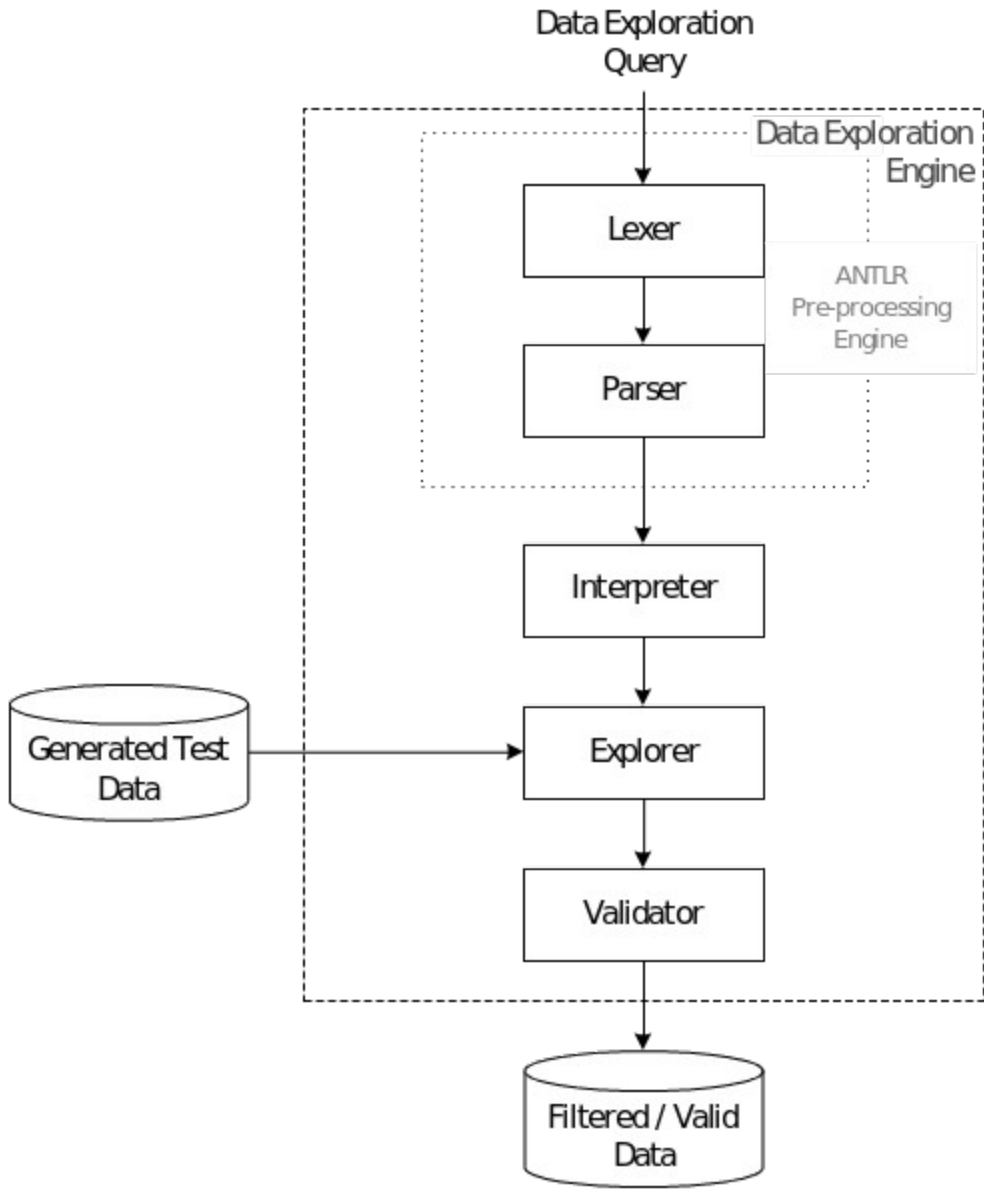


**Figure 4: Data Exploration Engine**

When exploring the generated data, the input query goes through the same processing components as in generation. But this time the parse tree is fed to an Explorer which loops through every entry in the generated data and checks whether the data meets the condition as specified by the query or not. If it does, it is then written to the valid data file, else, it is ignored. Where exploration becomes useful, is when the user has to check whether the generated data are exhaustive with reference to the input generation query, and whether it meets his demands. If not, this helps him modify the query to obtain the results he wanted.

### 3.2 System Diagrams

#### 3.2.1 Data Flow Diagram

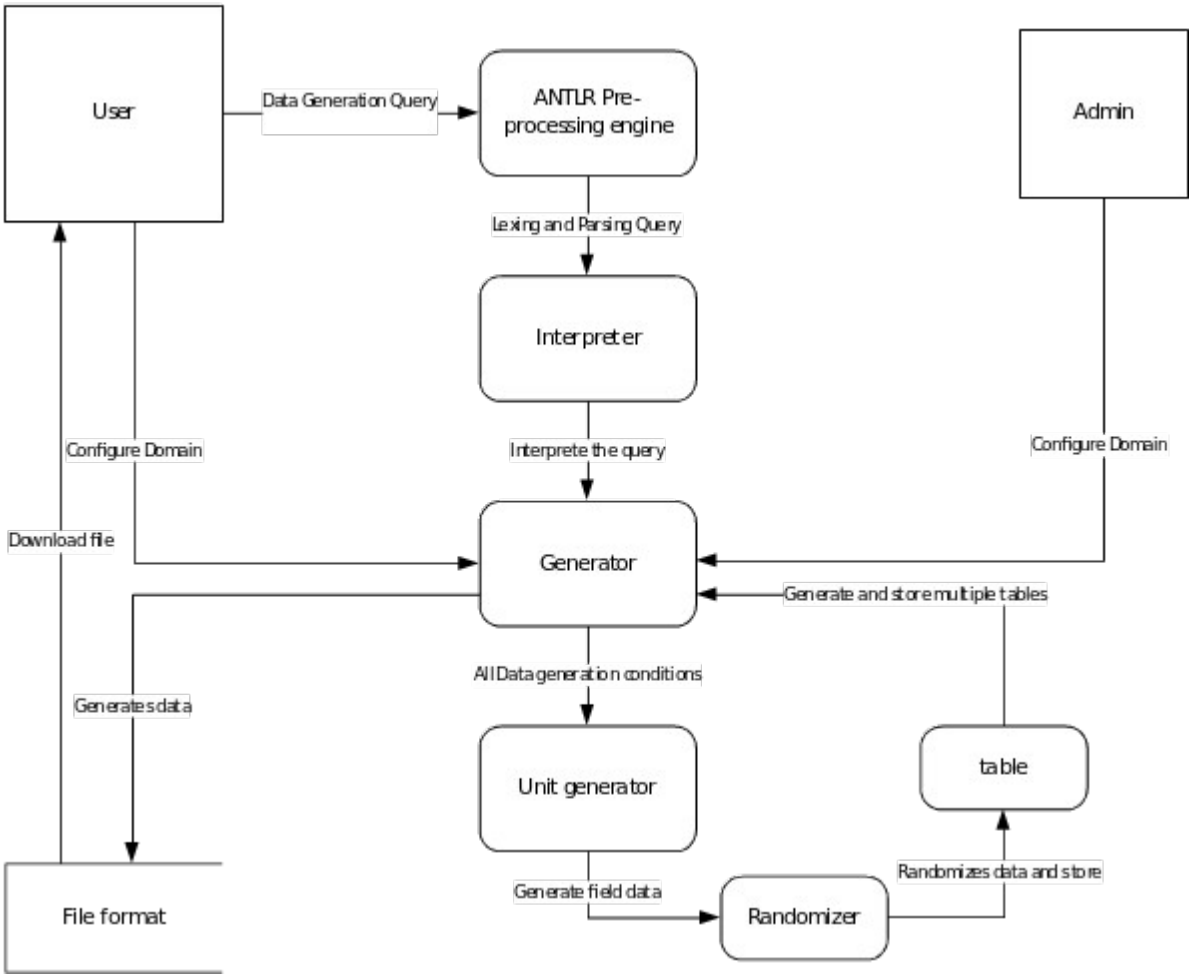


**Figure 5: Level 1 Data Flow Diagram**

### 3.2.2 Use Case Diagram

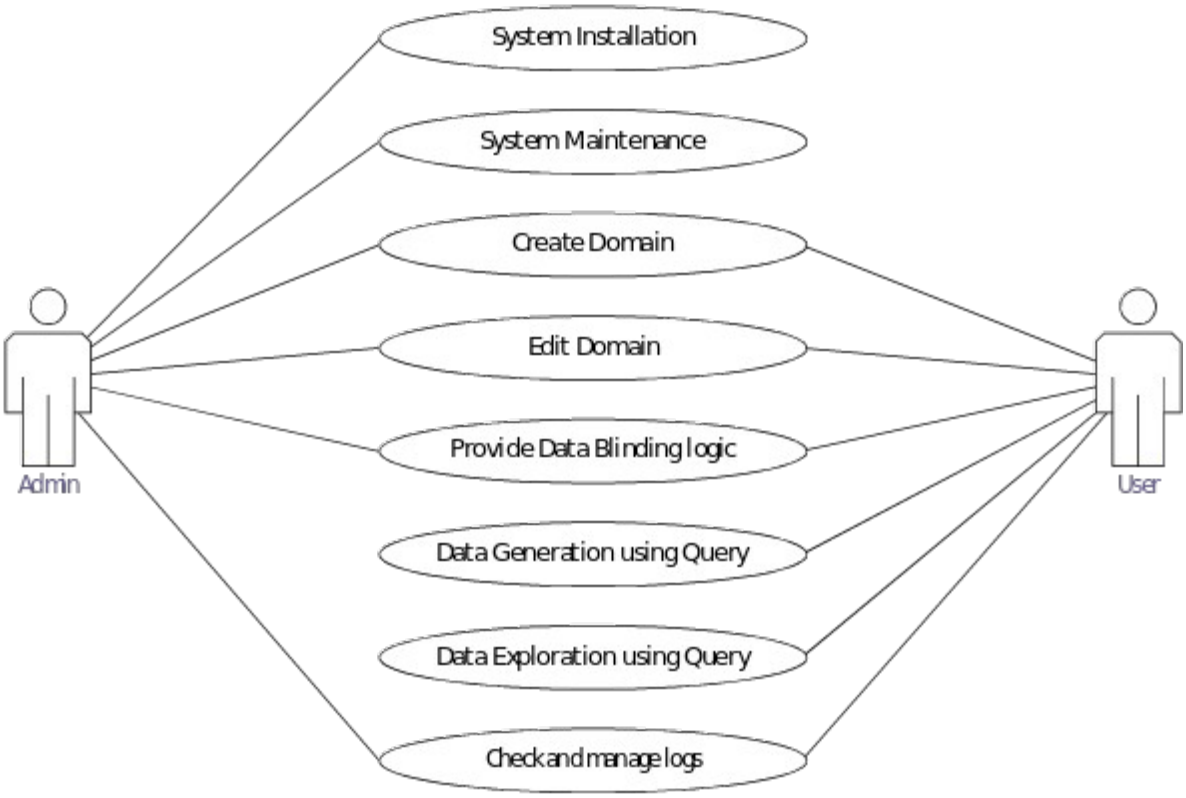


**Figure 6: Use Case Diagram**

### 3.3 System Operation

The front end of the system will be displayed on a web console. The backend web server will be written using Grails. The front end web console will be accessible to the end user and the admin.

There are basically three principle operations that make up the system:

#### 3.3.1 Domain Configuration

The admin is responsible for the creation and maintenance of the Domains that the system can use. As discussed earlier, the domains indicate the field names, their possible values, ranges, and also the relations that the fields have in between them.

#### 3.2.2 Data Generation

There are two ways to generate test data for use.

**a. Generation Query**

The user of the system can input a simple business logic (analogous to a query) to generate test data as per specified the query. The query goes through lexer, parser, interpreter and then through generator to finally give valid as well as invalid data that exhausts the conditions mentioned in the query. The generator accesses unit generator functions for data such as dates, numbers, currencies, zip codes etc. However for fields such as names, addresses, urls and so on, *field values* database is used. The generated data is then validated before output to the user.

**b. Blinding real world data**

Real world client's data cannot be used in the system for security and confidentiality reasons. However they can be blinded to form real-like data and used for testing purposed. Simple logics like increments and rotations are applied to the data to obtain blinded data. Real data can later be obtained by un-blinding this test data. The blinding logic is managed by the admin as a simple JSON file that contains all the cues about how every field is to be blinded.

#### 3.2.3 Data Exploration

The generated test data can also be explored by the system. This basically means applying a query similar to the generation query to filter the data into only those that satisfy the query.

Here too components like lexer, parser and interpreter come into play. The aim of data exploration is to analyze the generated data and verify whether the intended result was obtained. This will show what changes to the generation query are needed to achieve the desired result.

### 3.4 Project Timeline


**Figure 7: Project Timeline**

### 3.5 Tools

**Server Side:**

- JDK 1.7.0 or above
- Grails 2.2.4 or above
- Maven: It is a build automation tool used primarily for Java projects. This is required to fetch dependencies and build the project.
- IntelliJ IDEA: IntelliJ IDEA is a Java IDE by JetBrains is recommended but not necessary. The IDE will come in handy in case of quick error fix, maintenance and upgrade.

**Platform**

- Linux: It is a Unix-like and POSIX-compliant computer operating system assembled under the model of free and open source software development and distribution.

**Client Side:**

- JavaScript, CSS compatible browser

# 4. REQUIREMENT ENGINEERING

## 4.1 Requirement Elicitation

### 4.1.1 Functional Requirements

This section includes the requirements that specify all the fundamental actions of the software system.

***User Class 1 - The User***

**Functional requirement 1.1**

ID: FR1

TITLE: Domain Creation

DESC: Given that a user has opened the web console, then the user must be able to create new domains. The domain creation stage should involve providing domain name, table names, the fields in the table, the data types of the field, data sources in case of String, and data ranges in others.

RAT: In order for a user to create a new domain.

DEP: None

**Functional requirement 1.2**

ID: FR2

TITLE: Domain Editing

DESC: Given that a user has opened the web console, then the user must be able to edit existing domains. The domain editing stage should involve changing domain name, table names, the fields in the table, the data types of the field, data sources in case of String, and data ranges in others. Also adding and removing fields should be supported.

RAT: In order for a user to edit existing domains.

DEP: FR1

**Functional requirement 1.3**

ID: FR3

TITLE: Execute Queries for Data Generation

DESC: Given that a user has selected a domain, then the user must be able to execute queries on the selected domain for data generation.

RAT: In order for a user to generate desired data.

DEP: FR1

**Functional requirement 1.4**

ID: FR4

TITLE: Save generated data locally

DESC: Given that the user has generated data using a query, then the user must be able to save/download the generated data onto local storage.

RAT: In order for the user to save the generated data.

DEP: FR3

**Functional requirement 1.5**

ID: FR5

TITLE: Execute Queries for Data Exploration

DESC: Given that a user has selected a domain and generated data file, then the user must be able to execute queries on the generated data to check its validity with regard to fulfillment of initial requirements.

RAT: In order for a user to explore the generated data.

DEP: FR1 and FR3

**Functional requirement 1.6**

ID: FR6

TITLE: Check Logs

DESC: Given that the user has performed some action, then the user must be able to check the logs related to the performed action.

RAT: In order for a user to reference the internal processes that were performed by the system.

DEP: FR3 or FR5

**Functional requirement 1.7**

ID: FR7

TITLE: Define Blinding logic

DESC: Given that the user has stored some data files, then the user must be able to define blinding logic for individual fields in that file.

RAT: In order for a user to blind data as desired.

DEP: None

**Functional requirement 1.8**

ID: FR8

TITLE: Blind Data

DESC: Given that the user has stored some data files and defined blinding logic for them, then the user must be able to execute commands to blind the data.

RAT: In order for a user to blind data as desired.

DEP: FR7

***User Class 2 - Admin***

**Functional requirement 2.1**

ID: FR9

TITLE: Domain Creation

DESC: Given that the admin has opened the web console, then the admin must be able to create new domains. The domain creation stage should involve providing domain name, table names, the fields in the table, the data types of the field, data sources in case of String, and data ranges in others.

RAT: In order to create a new domain.

DEP: None

**Functional requirement 2.2**

ID: FR10

TITLE: Domain Editing

DESC: Given that the admin has opened the web console, then the admin must be able to edit existing domains. The domain editing stage should involve changing domain name, table names, the fields in the table, the data types of the field, data sources in case of String, and data ranges in others. Also adding and removing fields should be supported.

RAT: In order to edit existing domains.

DEP: FR9

**Functional requirement 2.3**

ID: FR11

TITLE: Check Logs

DESC: Given that some action has been performed by the admin or a user, then the admin must be able to check the logs related to the performed action.

RAT: In order to reference the internal processes that were performed by the system.

DEP: FR3, FR5, FR9 or FR10

**Functional requirement 2.4**

ID: FR12

TITLE: Define Blinding logic

DESC: Given that some data files have been stored, then the admin must be able to define blinding logic for individual fields in that file.

RAT: In order for a user to blind data as desired.

DEP: None

**Functional requirement 2.5**

ID: FR13

TITLE: Apply patches and upgrades to system

DESC: Given that the system has been installed, and a patch or update has been issued by the developers, then the admin must be able to apply the patches or execute upgrade commands.

RAT: In order to upgrade and maintain the system.

DEP: none

### 4.1.2 Non-functional Requirement:

**1. Hard drive space**

ID: NFR1

TAG: HardDriveSpace

GIST: Hard drive space.

SCALE: The application's need of hard drive space.

METER: GB.

MUST: No more than 15 GB.

PLAN: No more than 10 GB.

WISH: No more than 8 GB.

MB: DEFINED: Gigabyte

**2. Application memory usage**

ID: NFR2

TAG: ApplicationMemoryUsage

GIST: The amount of Operating System memory occupied by the application.

SCALE: MB.

METER: Observations done from the performance log during testing

MUST: No more than 120 MB.

PLAN: No more than 105 MB

WISH: No more than 90 MB

Operate System: DEFINED: The Operating System on which the application is running on.

MB: DEFINED: Megabyte.

**3. Operating System: Linux, Windows**

**4. Server: Linux**

**5. Programming Language: Java**

**6. Web-Framework: Grails**

**7. Web browsers: Javascript/CSS compatible browsers**

### 4.2 Requirement Analysis

Software testing is one of the indispensable parts of software development in today's world. Proper testing of the software during and after the development phase can save lot of efforts, time and money. Software testing requires proper test data that can meet all the boundaries and constraints. Not only valid data but we also require invalid data for the testing of software.

#### 4.2.1 Functional Requirements

Our project 'Test Data Generator' can serve as an important part in software development life cycle. 'Test Data Generator' takes a domain configuration file and makes the user input a simple query as per the requirement and generates thousands of valid as well as invalid sets of data within a few seconds.

#### 4.2.2 Technical Requirements

The process of generating the required data according to the user specification is the main task of the system. It is not easier to generate data fulfilling all the specifications and collect it as a table. It requires minimum hardware and software setup to establish our system into the organization. We are building our query language and GUI in such a way that even the one with very little technical knowledge can use it easily with the help of a simple documentation.

#### 4.2.3 Operational Requirements

The operation should be independent of any platform. It should understand and parse natural language like queries. It should generate data according to the given query. Randomization of generated data should be retained. Relation among the different fields shouldn't be violated during data generation process. All the generated data should be properly explored. There should be enough security maintained to preserve the privacy of the customers.

### 4.3 Documentation

Documentation is the description of what a particular software does or shall do. It is used throughout development to communicate what the software does or shall do. It is also used as an agreement or as the foundation for agreement on what the software shall do. Requirements are produced and consumed by everyone involved in the production of software: end users, customers, product managers, project managers, sales, marketing, software architects,

usability engineers, interaction designers, developers, and testers, to name a few. Thus, requirements documentation has many different purposes.

System documentation includes all of the documents describing the system itself from the requirements specification to the final acceptance test plan. Documents describing the design, implementation and testing of a system are essential if the program is to be understood and maintained. Like user documentation, it is important that system documentation is structured, with overviews leading the reader into more formal and detailed descriptions of each aspect of the system. For large systems that are developed to a customer's specification, the system documentation should include:

1. The requirements document and an associated rationale.
2. A document describing the system architecture.
3. For each program in the system, a description of the architecture of that program.
4. For each component in the system, a description of its functionality and interfaces.
5. Program source code listings.
6. Validation documents describing how each program is validated and how the validation information relates to the requirements.

Program source code listings should be commented where the comments should explain complex sections of code and provide a rationale for the coding method used. If meaningful names are used and a good, structured programming style is used, much of the code should be self-documenting without the need for additional comments. This information is now normally maintained electronically rather than on paper with selected information printed on demand from readers.

Best practices for increasing the liveliness of documentation:

- Writing
  Prefer executable specifications over static documents
  Document stable concepts, not speculative ideas
  Generate system documentation
- Simplification
  Keep documentation just simple enough, but not too simple
  Write the fewest documents with least overlap
  Put the information in the most appropriate place
  Display information publicly
- Determining What to Document
  Document with a purpose
  Focus on the needs of the actual customers(s) of the document
  The customer determines sufficiency
- Determining When to Document

Iterate, iterate, iterate
Find better ways to communicate
Start with models you actually keep current
Update only when it hurts

- General
  Treat documentation like a requirement
  Require people to justify documentation requests
  Recognize that you need some documentation
  Get someone with writing experience

### 4.4 Requirement Review

A requirements review or walk-through is a meeting where all the stakeholders are gathered together and walk-through the requirements documentation, page-by-page, line-by-line, to ensure that the document represents everyone's complete understanding of what is to be accomplished in this particular project. Because requirements specifications are formally in people's minds, requirements validation must necessarily involve the clients and the user. Requirement reviews, in which the SRS is carefully reviewed by a group of people including representative of the clients and the users, are the most common methods of validation. The participants ask questions, share doubts, or seeks clarification. Checklists are frequently used in reviews to focus the review effort and to ensure that no major source of error is overlooked by the reviewers. A good checklist will usually depend on the project.

Following key points are the examples of the document review:

1. Are all hardware resource defined?
2. Have the response times of functions been specified?
3. Have all the hardware, external software, and the data interfaces been defined?
4. Have all the functions required by the client been specified.
5. Is each requirement testable?
6. Is the initial state of the system defined?
7. Are the responses to exceptional conditions specified?
8. Does the requirement contain restriction that can be controlled by the designer?
9. Are possible future modifications specified?

Stages to perform requirement review:

- define reviewers
- work teams
- gates/checkpoints
- baselines of requirement document
- review iteration
- comments handling
- approval procedure

# 5. DEVELOPMENT MODEL

The Prototyping Model is a systems development method (SDM) in which a prototype (an early approximation of a final system or product) is built, tested, and then reworked as necessary until an acceptable prototype is finally achieved from which the complete system or product can now be developed. This model works best in scenarios where not all of the project requirements are known in detail ahead of time. It is an iterative, trial-and-error process that takes place between the developers and the users.

There are several steps in the Prototyping Model:

- The new system requirements are defined in as much detail as possible. This usually involves interviewing a number of users representing all the departments or aspects of the existing system.
- A preliminary design is created for the new system.
- A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.
- The users thoroughly evaluate the first prototype, noting its strengths and weaknesses, what needs to be added, and what should to be removed. The developer collects and analyzes the remarks from the users.
- The first prototype is modified, based on the comments supplied by the users, and a second prototype of the new system is constructed.
- The second prototype is evaluated in the same manner as was the first prototype.
- The preceding steps are iterated as many times as necessary, until the users are satisfied that the prototype represents the final product desired.
- The final system is constructed, based on the final prototype.
- The final system is thoroughly evaluated and tested. Routine maintenance is carried out on a continuing basis to prevent large-scale failures and to minimize downtime.

# 6. SYSTEM SPECIFICATIONS

## 6.1 Improvement in Domain Configuration

In ‘SmarTest’, the domain configuration is confined to a single field which means domain cannot be configured relating different fields, tables and domains. In smartest, there is an admin UI where an admin can configure the domain. For example, for a particular domain, the admin will have to enter the name of columns in the domain, the type of each column (date, number or string), the allowed values in each column (upper and lower range for number and date type and possible values, which may be a file containing values or a list, for string type columns) and the Boolean value “required” which is true if the column is required in all data files.

We will modify the configuration mechanism such that the admin can set up fields, tables and domains relations as well. Output of the configuration will be a JSON file which would look somewhat like the following,

```
{
   "column1": {
      "id": "1",
      "type": "date",
      "name": "DOB",
      "upper_range": "11/11/2014",
      "lower_range": "11/11/1990",
   },
   "column2": {
      "id": "2",
      "type": "integer",
      "name": "Age",
      "unique": "true"
   }
}
```

Here, the fields DOB and Age are related to each other. We will have similar JSON configuration relating different field, table and domains.

We will basically improve following sections in domain configuration:

### 6.1.1 Field Relation

We will implement the relation between different fields (columns). For eg: There may be two field Age and Date of Birth (DOB). The Age field should be related with DOB directly. We will configure the JSON file accordingly to establish the field relation.

### 6.1.2 Table relation

Not only in field, but the relation between data may also be in different table. We will use foreign key concept while generating such test data. For e.g. Let us consider following Table1 and Table2:

**Table 1**

| ID | Name | Address |
|---|---|---|

**Table 2**

| ID | Salary |
|---|---|
| 1 | 1000 |
| 2 | 2000 |
| 3 | 3000 |

In this case, we will use ID as a foreign key. We will configure the JSON file accordingly so that it can deal with foreign key.

### 6.1.3 Domain relation

We may arrive to the case where data in different domain are related. For Example: There may be two domain Health and Education where data of Health directly affects the data in Education. Hence, considering the requirement, we will generate the test data that represents the multiple domains.

## 6.2 Improvement in query language

We will improve in query language so that it supports all types of business logic that may come across. Our query language will support all type of business logics that a standard SQL supports. Our query will support the table relation as well as domain relation.

## 6.3 Improvement in data generation

We will improve the data generation process so that the data is generated for all cases. For e.g. If a user asks for the 1000 data out of which 500 data meet the condition A and B, then the remaining 500 data include the data which meet the following conditions:

not A and not B
not A and B
A and not B

### 6.4 Test data validation

Test data validation ensures that the generated data is valid. Validation is done with respect to the set of generated data that meets the given conditions and the set of generated data that don’t meet the given conditions.

### 6.5 Use of Randomization

Randomization is the process of making something random. Randomization is not haphazard. Instead, a random process is a sequence of random variables describing a process whose outcomes do not follow a deterministic pattern, but follow an evolution described by probability distribution. Random permutation of sequence when shuffling cards, random sampling of population in statistical sampling, random number generator, transforming data stream, observing atomic delay, etc. are some application of randomization.

In our project Randomization plays a vital role. While generating data with mentioned boundary condition, we have to generate different sets of data each and every time with equal probability of occurrence of each data. In this process data should be constantly random in every trial of data generation. In our project, randomization gives the datasets which meant to represent the whole data.

Basically there are two types of randomization: non-algorithmic randomization and algorithmic randomization. Non-algorithmic randomization are like throwing dice, shuffling cards, roulette wheels, lottery machines, observing atomic decay using radiation counter, etc. While randomized algorithm is an algorithm that employs a degree of randomness as a part of its logic. Las Vegas algorithm, Monte Carlo algorithm and quick sort algorithm are some example of randomized algorithm.

Currently, we are doing research in Monte Carlo algorithm and Metropolis algorithm and how to fit it in our project. Although we have secondary choice of simple randomization, we’re using current timestamp of our computer, which we have successfully tested.

### 6.6 Data Blinding

Data blinding refers to changing the source data in some way as to hide the identity of the data but still have access to original data upon un-blinding. This is somewhat like encryption, but the data types and lengths are preserved. Also the field relations are maintained in the blinded data as well. Basically, the blinding will be for Strings, Numbers, and Dates. The blinding logics are very simple and are defined in a JSON file. For example, for the following table,

**Table 1**

| ID | Name | Address |
|---|---|---|
| 1 | a | x |
| 2 | b | y |
| 3 | c | z |

A blinding logic could be defined as below,

```
{
        'ID': 'increment by 12',
        'Name': 'rotate by 9',
        'Address': 'encrypt xor my_key'
}
```

The data blinder class would take care of the required logics. The main purpose of blinding is to not expose the identity of the data straight away. This is needed to avoid privacy issues.

### 6.7 Usability Enhancements

The biggest change from previous system would probably be the adoption of web architecture. This will enable the system to run from a single server, where all the domain knowledge and the databases are stored, for every client to use. This is in total contrast with 'SmarTest' which ran as a desktop app. The users will be able to access the system by an intuitive web based interface that will enable them to configure the system, perform their queries, and get the results. The system will be highly usable for the admin as well as the end user.

# 7. SYSTEM REQUIREMENTS

## 7.1 Software Requirements

**Server Side:**

**Tools:**

- JDK 1.7.0 or above
- Grails 2.2.4 or above
- Maven: It is a build automation tool used primarily for Java projects. This is required to fetch dependencies and build the project.
- IntelliJ IDEA: IntelliJ IDEA is a Java IDE by JetBrains is recommended but not necessary. The IDE will come in handy in case of quick error fix, maintenance and upgrade.

**Platform**

- Linux: It is a Unix-like and POSIX-compliant computer operating system assembled under the model of free and open source software development and distribution.

**Client Side:**

- JavaScript, CSS compatible browser

## 7.2 Hardware Requirements

**Recommended Server Configuration:**

- Intel® Core™ i3 Processor, 5th Gen, 2 GHz
- 8 GB RAM
- 20 GB Storage + Database size

**Client Side Requirements:**

- Computer or a mobile device
- Network connectivity

# 8. APPENDICES

## 8.1 Query Execution from Command Line

**Query #1:** `table_name1.name="a.*"`

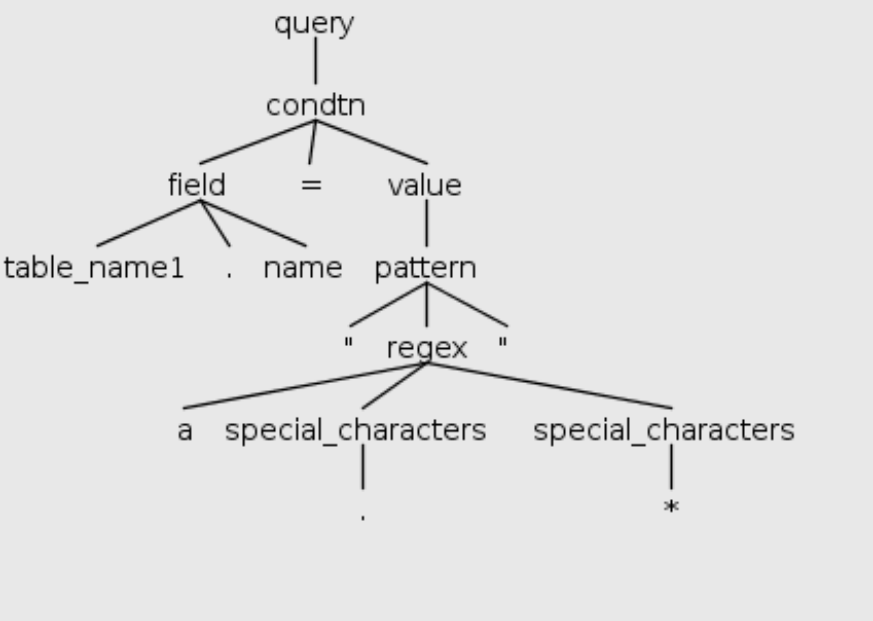


**Figure 9: Generated Tree for Query 1**

| | 1. name | 2. gender | 3. dob | 4. id | 5. Age2 | 6. salary | 7. Age1 |
|---|---|---|---|---|---|---|---|
| 1 | name | gender | dob | id | Age2 | salary | Age1 |
| 2 | Annelle | M | 2000/12/23 | 1 | 50 | 29692.73 | 78 |
| 3 | Aron | F | 1995/07/27 | 2 | 20 | 43042.71 | 82 |
| 4 | Alishia | F | 2014/07/29 | 3 | 47 | 12767.94 | 71 |
| 5 | Arlie | F | 1993/09/03 | 4 | 34 | 7367.48 | 93 |
| 6 | Altha | M | 1998/02/13 | 5 | 47 | 5838.06 | 26 |
| 7 | Arielle | F | 2008/12/23 | 6 | 36 | 49078.86 | 82 |
| 8 | Alejandra | M | 1993/03/23 | 7 | 25 | 48833.10 | 90 |
| 9 | Alexander | M | 1998/09/07 | 8 | 35 | 28663.07 | 43 |
| 10 | Angel | M | 2010/01/15 | 9 | 28 | 17349.71 | 71 |
| 11 | Adela | M | 2000/12/23 | 10 | 42 | 30619.12 | 90 |
| 12 | Arturo | M | 2010/04/16 | 11 | 33 | 18525.69 | 84 |
| 13 | Aja | F | 2007/08/20 | 12 | 47 | 46372.34 | 30 |
| 14 | Ardath | F | 1994/10/25 | 13 | 46 | 29684.13 | 71 |
| 15 | Amina | M | 1993/09/03 | 14 | 46 | 13934.28 | 66 |
| 16 | Angelic | M | 1998/09/07 | 15 | 26 | 28663.07 | 66 |
| 17 | Ashlee | F | 2010/01/15 | 16 | 49 | 32944.63 | 66 |
| 18 | Andrew | M | 1999/10/11 | 17 | 38 | 7231.32 | 34 |
| 19 | Azzie | F | 2009/10/11 | 18 | 36 | 35818.30 | 73 |
| 20 | Adrianne | F | 1995/03/21 | 19 | 33 | 44113.46 | 79 |
| 21 | Alaina | F | 2004/03/09 | 20 | 25 | 13934.28 | 34 |
| 22 | Angel | F | 2007/08/20 | 21 | 50 | 18944.42 | 43 |
| 23 | Andree | M | 2000/06/10 | 22 | 20 | 12767.94 | 48 |
| 24 | Amal | F | 2010/04/16 | 23 | 27 | 44960.64 | 82 |
| 25 | Arturo | F | 1992/04/12 | 24 | 36 | 32944.63 | 66 |
| 26 | Angle | F | 1993/09/03 | 25 | 25 | 46128.06 | 68 |
| 27 | Arnulfo | M | 2008/12/23 | 26 | 36 | 48833.10 | 34 |
| 28 | Antony | M | 1995/03/21 | 27 | 36 | 28663.07 | 93 |
| 29 | Ayako | F | 1992/04/12 | 28 | 24 | 49078.86 | 96 |
| 30 | Arlene | F | 1997/08/15 | 29 | 24 | 24595.92 | 34 |
| 31 | Alton | F | 2007/12/19 | 30 | 25 | 48833.10 | 90 |
| 32 | Annika | F | 2007/08/20 | 31 | 29 | 39963.14 | 89 |
| 33 | Aida | M | 2000/12/23 | 32 | 50 | 45292.34 | 53 |
| 34 | Ardell | M | 2000/12/23 | 33 | 28 | 29692.73 | 69 |
| 35 | Aliza | M | 2010/10/25 | 34 | 49 | 29692.73 | 74 |

**Figure 10: Output for Query 1**

**Query #2:** table_name1.name="c.*" and table_name1.dob>"2005/5/1"

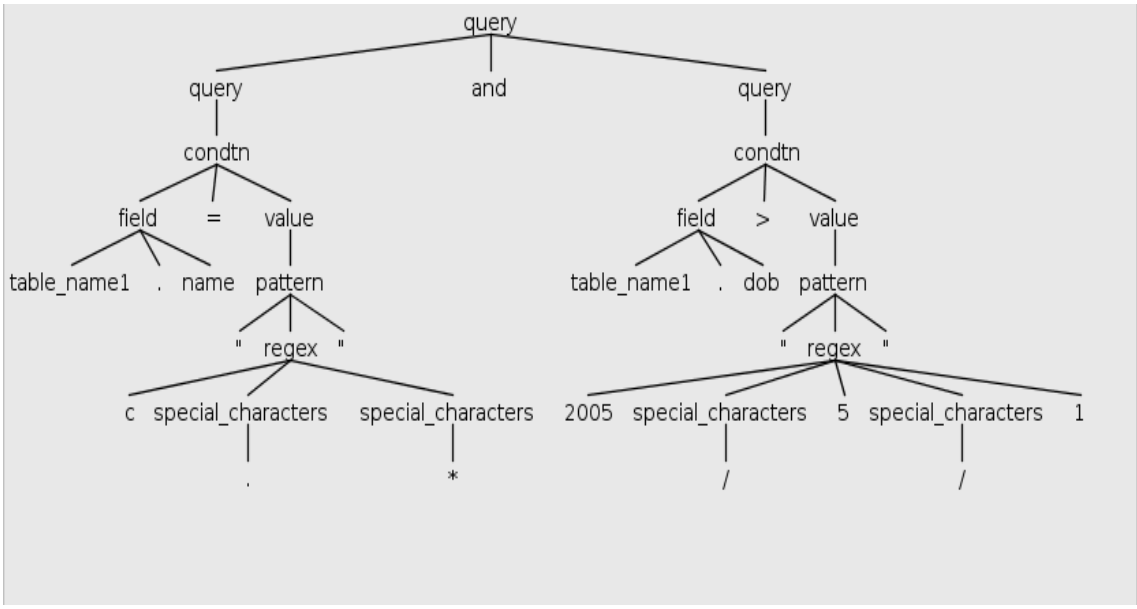


**Figure 11: Generated Tree for Query 2**

| | 1. name | 2. dob | 3. gender | 4. id | 5. Age2 | 6. salary | 7. Age1 |
|---|---|---|---|---|---|---|---|
| 1 | name | dob | gender | id | Age2 | salary | Age1 |
| 2 | Casey | 2014/08/18 | F | 1 | 41 | 5576.63 | 97 |
| 3 | Calandra | 2008/09/27 | M | 2 | 41 | 11806.54 | 31 |
| 4 | Cordie | 2005/11/03 | M | 3 | 35 | 42706.65 | 35 |
| 5 | Chan | 2007/06/28 | M | 4 | 36 | 7439.21 | 87 |
| 6 | Cristine | 2006/03/03 | F | 5 | 50 | 40663.15 | 45 |
| 7 | Cari | 2014/08/18 | F | 6 | 49 | 20793.39 | 81 |
| 8 | Codi | 2010/01/31 | M | 7 | 36 | 24951.36 | 78 |
| 9 | Charlene | 2009/10/03 | M | 8 | 45 | 38983.01 | 39 |
| 10 | Christia | 2012/04/17 | M | 9 | 34 | 11873.85 | 97 |
| 11 | Carly | 2011/02/16 | F | 10 | 45 | 9296.87 | 41 |
| 12 | Carline | 2012/05/09 | M | 11 | 42 | 11873.85 | 25 |
| 13 | Carolann | 2013/08/08 | F | 12 | 49 | 44211.90 | 97 |
| 14 | Claribel | 2013/05/19 | M | 13 | 33 | 35087.64 | 87 |
| 15 | Cammy | 2009/10/20 | F | 14 | 44 | 13023.11 | 94 |
| 16 | Cuc | 2005/12/05 | F | 15 | 50 | 20793.39 | 40 |
| 17 | Clemencia | 2011/03/24 | M | 16 | 49 | 44211.90 | 78 |
| 18 | Chae | 2005/10/29 | F | 17 | 20 | 41119.58 | 94 |
| 19 | Cordelia | 2011/12/02 | M | 18 | 34 | 11806.54 | 70 |
| 20 | Corrin | 2009/11/30 | M | 19 | 34 | 41119.58 | 25 |
| 21 | Charley | 2007/03/22 | M | 20 | 49 | 41249.74 | 74 |
| 22 | Coral | 2008/08/06 | M | 21 | 35 | 43682.97 | 52 |
| 23 | Coletta | 2011/05/18 | M | 22 | 49 | 39234.46 | 25 |
| 24 | Chanelle | 2013/04/25 | M | 23 | 50 | 9444.81 | 40 |
| 25 | Crysta | 2013/10/17 | M | 24 | 23 | 24951.36 | 71 |
| 26 | Colby | 2008/09/23 | M | 25 | 21 | 11873.85 | 94 |
| 27 | Chery | 2013/04/14 | M | 26 | 26 | 41119.58 | 40 |
| 28 | Chase | 2011/10/31 | M | 27 | 44 | 5576.63 | 66 |
| 29 | Cassondra | 2014/01/05 | M | 28 | 45 | 40663.15 | 79 |
| 30 | Claribel | 2010/11/22 | F | 29 | 33 | 43682.97 | 34 |
| 31 | Catharine | 2005/08/21 | M | 30 | 24 | 17096.37 | 73 |
| 32 | Charity | 2010/08/25 | M | 31 | 44 | 18877.41 | 66 |
| 33 | Cythia | 2008/05/31 | F | 32 | 33 | 17096.37 | 36 |
| 34 | Corey | 2006/03/08 | M | 33 | 20 | 11873.85 | 29 |
| 35 | Cher | 2013/06/21 | M | 34 | 30 | 33950.75 | 29 |

**Figure 12: Output for Query 2**

**Query #3:** table_name1.salary>5000.00 OR (table_name1.Age2>30 AND table_name1.dob<"2004/1/1")

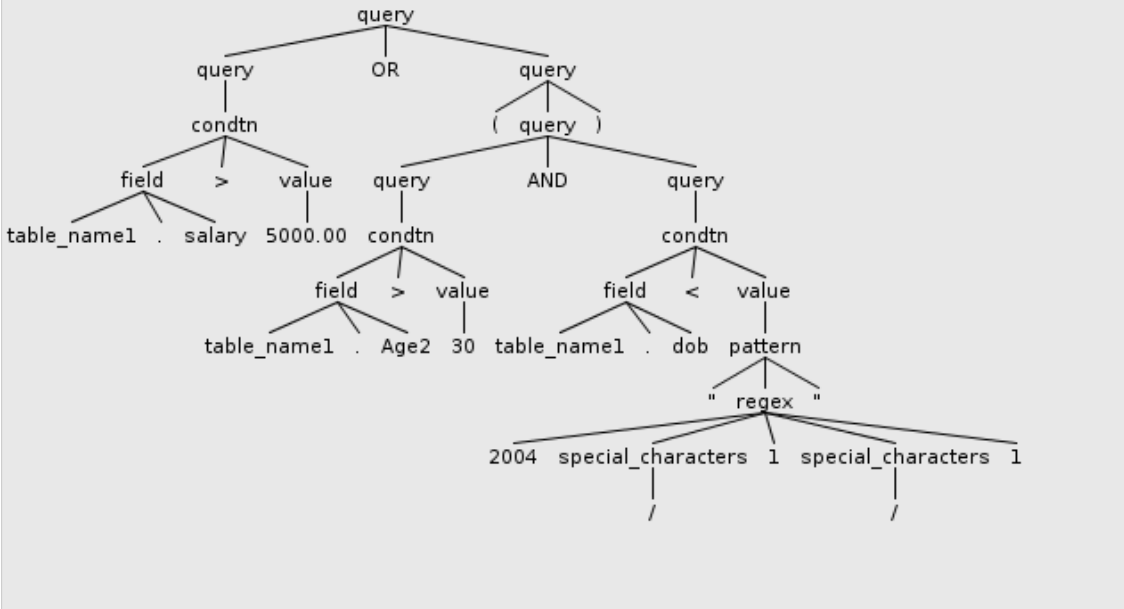


**Figure 13: Generated Tree for Query 3**

| | 1. salary | 2. Age2 | 3. dob | 4. gender | 5. name | 6. id | 7. Age1 |
|---|---|---|---|---|---|---|---|
| 1 | salary | Age2 | dob | gender | name | id | Age1 |
| 2 | 18871.27 | 35 | 1991/10/21 | F | Khadijah | 1 | 67 |
| 3 | 36760.49 | 45 | 1992/09/23 | F | Clyde | 2 | 87 |
| 4 | 45322.75 | 35 | 1994/06/25 | F | Nanette | 3 | 77 |
| 5 | 38359.87 | 48 | 1997/05/13 | F | Freda | 4 | 97 |
| 6 | 34406.04 | 43 | 1996/05/19 | F | Shanika | 5 | 55 |
| 7 | 21780.73 | 43 | 1991/04/25 | M | Dorie | 6 | 87 |
| 8 | 5749.62 | 41 | 1996/02/24 | F | Ethel | 7 | 94 |
| 9 | 22155.76 | 37 | 2000/07/24 | F | Lillian | 8 | 39 |
| 10 | 33873.90 | 33 | 1995/02/14 | F | Ricardo | 9 | 28 |
| 11 | 9511.36 | 47 | 1993/09/03 | F | Glynis | 10 | 77 |
| 12 | 39384.18 | 50 | 1991/11/27 | F | Ayanna | 11 | 28 |
| 13 | 5743.60 | 38 | 1995/02/14 | M | Vannesa | 12 | 85 |
| 14 | 27160.22 | 39 | 1999/08/14 | F | Ricardo | 13 | 48 |
| 15 | 29842.72 | 34 | 1996/06/16 | M | Blaine | 14 | 28 |
| 16 | 31535.83 | 33 | 1998/08/03 | M | Cyndy | 15 | 27 |
| 17 | 8972.88 | 34 | 1998/04/21 | F | Lucio | 16 | 34 |
| 18 | 31535.83 | 20 | 1994/06/25 | M | Cedric | 17 | 76 |
| 19 | 26151.86 | 23 | 1993/09/03 | M | Graig | 18 | 26 |
| 20 | 5749.62 | 20 | 1992/03/10 | M | Howard | 19 | 44 |
| 21 | 27843.56 | 21 | 2005/02/04 | M | Katelynn | 20 | 67 |
| 22 | 15995.33 | 20 | 2009/02/26 | F | Willy | 21 | 87 |
| 23 | 18871.27 | 27 | 2000/08/31 | M | Freda | 22 | 33 |
| 24 | 36126.44 | 45 | 2011/09/21 | M | Ayanna | 23 | 49 |
| 25 | 36760.49 | 38 | 2007/10/07 | M | Mellisa | 24 | 43 |
| 26 | 35580.93 | 34 | 2006/08/06 | M | Valentina | 25 | 79 |
| 27 | 45322.75 | 30 | 2013/03/11 | F | Ronald | 26 | 32 |
| 28 | 5743.60 | 22 | 1998/08/03 | M | Katelynn | 27 | 39 |
| 29 | 5162.38 | 26 | 2001/02/03 | F | Ricardo | 28 | 28 |
| 30 | 27160.22 | 36 | 2014/11/04 | M | Clyde | 29 | 38 |
| 31 | 38193.93 | 23 | 2009/03/07 | M | Onie | 30 | 32 |
| 32 | 8972.88 | 22 | 1996/04/22 | F | Ayanna | 31 | 85 |
| 33 | 48703.69 | 28 | 1991/02/13 | M | Ronald | 32 | 37 |
| 34 | 5000.00 | 38 | 1995/02/14 | M | Ethel | 33 | 27 |
| 35 | 5000.00 | 43 | 1991/04/25 | M | Perla | 34 | 63 |
| 36 | 5000.00 | 34 | 1992/04/26 | F | Renae | 35 | 87 |

**Figure 14: Output for Query 3**

## 9. REFERENCES


1. Wikipedia “Test Data Generation”, http://en.wikipedia.org/wiki/Test_data_generation, 2014.
2. R. Prakash “Tips to design test data before executing your cases”, http://www.softwaretestinghelp.com/tips-to-design-test-data-before-executing-your-test-cases/, 2008.
3. M. K. Yong “JSON-SIMPLE in java”, http://www.tutorialspoint.com/json/json_java_example.html, 2011.
4. Maven “Read-write csv files using open csv”, http://opencsv.sourceforge.net, 2011.
5. M. K. Yong “Tutorial on Maven installation and concepts”, http://www.mkyong.com/tutorials/maven-tutorials/, 2011.
6. TimSporcic “Getting started with grails”, https://grails.org/tutorials, 2012-2014.
7. Scott Stanchfield “ANTLR 3.x Tutorial”, http://javadude.com/articles/ANTLR3xtut/, 2014.
8. Wikipedia “Monte Carlo Algorithm”, http://en.wikipedia.org/wiki/Monte_Carlo_algorithm, 2014.
9. Midusha Shrestha, Anup Shrestha, Krishna Parajuli, Ritu Bafna “SmarTest”, http://flipkarma.com/project/smartest-test-data-generation-and-exploration/