A COMPUTING TOOL AND A RELATIONAL DATABASE FOR A MEDICAGO TRUNCATULA MUTAGENESIS PROJECT

By

LIN GE

Bachelor of Science

Suzhou Railway Teachers College

Suzhou, P. R. China

1991

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2007

A COMPUTING TOOL AND A RELATIONAL

DATABASE FOR A MEDICAGO TRUNCATULA

MUTAGENESIS PROJECT

Thesis Approved:

G. E. Hedrick

Thesis Adviser

John P. Chandler

Nohpill Park

A. Gordon Emslie

Dean of the Graduate College

ACKNOWLEDGEMENTS

I wish to express my deep and sincere gratitude to my thesis advisor, Dr. G. E. Hedrick, for his excellent guidance and detailed advice throughout this study. I am also grateful to Drs. J. P. Chandler and N. Park for their encouragement and constructive suggestions as my committee members.

I would like to thank Dr. R. Chen and the research fellows in his laboratory at Noble Foundation for providing me with the opportunity to work on their plant science project, which allowed me to serve biological research employing the knowledge of computer science.

Last but certainly not least, I am deeply indebted to the love, patience and constant support of my family.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

1.1. Motivation

Among crops, legume species such as soybean and alfalfa are unique in their ability to fix atmospheric nitrogen thanks to the formation of root nodules in which they house symbiotic bacteria. Since there is no limitation for nitrogen legumes have developed an ability to accumulate remarkable levels of protein, and contribute nearly 33% of the dietary protein needs of humans. The synthesis of nitrogen fertilizers consumes fossil energy. The use of nitrogen-fixing legumes to produce proteins results in a substantial decrease in the consumption of fossil fuels, and thereby lowers the agricultural contribution to global warming [1]. Legumes also are a rich source of edible oil and diverse natural products with health benefits.

To understand the biological processes unique to legumes, it is logical to concentrate efforts on a model legume that has a small genome. Information gained on the species can be transferred to other related legume species. For this reason, *Medicago truncatula* (*M. truncatula*, hereafter), has emerged as a model legume. It has a small genome, greatly facilitating genetic analysis. The sequencing of the gene spaces in *M. truncatula* is scheduled to be completed by the end of 2006 [2].

Plant biologists have already amassed DNA sequence information for thousands of different genes and gene families in *M. truncatula*. One of the next challenges for plant biologists is to assign biological functions to all theses sequenced genes. Missing from plant biologist's toolbox is a method for generating plant populations that carry "knockout" mutations of sequenced genes. The process would greatly assist in efforts to determine the function of genes in vivo. The research community of *M. truncatula* has initialized several projects in parallel to generate large mutant populations of the legume using various mutagenesis methods. As a part of the efforts, Dr. Rujin Chen's group in the Samuel Roberts Noble Foundation is generating a mutant library of *M. truncatula* using fast neutron radiation (Figure 1.1).

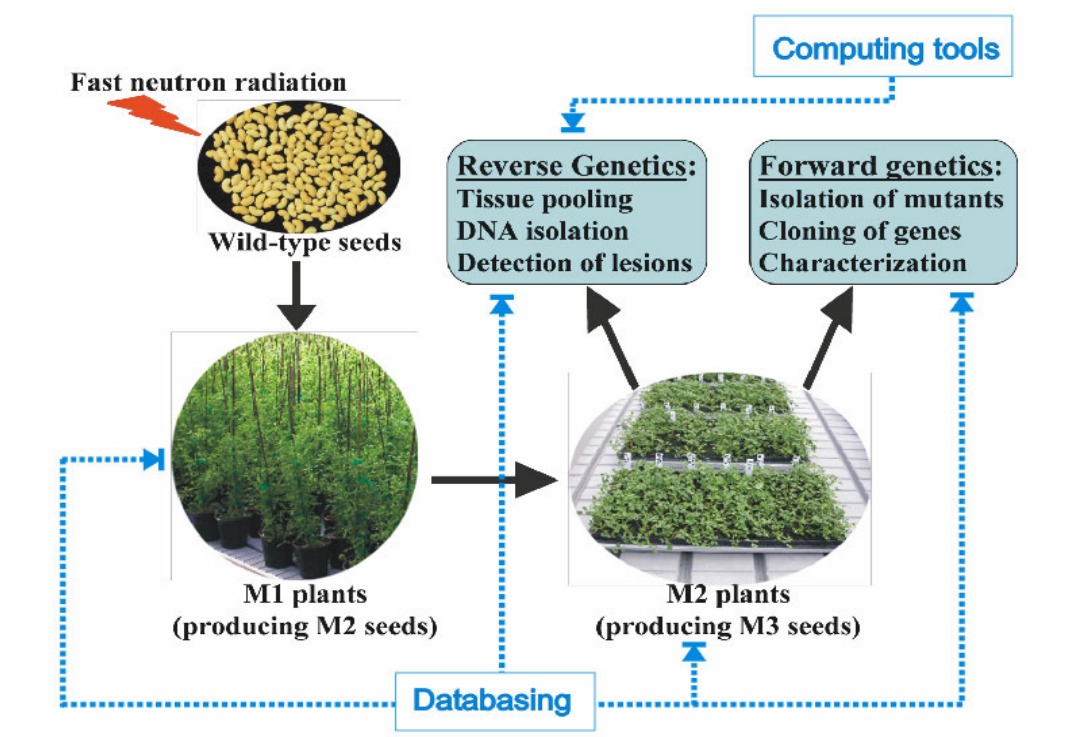


Figure 1.1. Overview of construction of *M. truncatula* mutant population using fast neutron radiation. Steps requiring databasing or the assistance of computing tools are indicated.

Embodied in the project is the recognized need that multiple steps in the process require informatics and laboratory information management tools which monitor the tracking of mutated plants, their DNA and their grains; the design of primers for polymerase chain reaction (PCR) amplification of targeted regions, the interpretation and databasing of mutant alleles; and the integrated analysis of mutant alleles and phenotypic information.

As shown in Figure 1.1., the scientists in the Samuel Roberts Noble Foundation apply the bombardment of fast neutrons to the wild-type seeds of *M. truncatula*. The mutated seeds are grown to generate M1 plants (M1, first generation of mutagenized plants). The M1 plants yield M2 grains (M2, second generation of mutagenized plants). 5 of M1 plants are grown in the same pot. Their M2 grains are harvested together and are stored in numbered bag. The detailed information such as the radiation dosage applied, planting date, grain yield and visible phenotype is recorded and stored in a database. This is the first stage of the project. Plants for mutant screening are usually grown from M2 seeds because most mutant phenotypes result from homozygous recessive mutations, and M1 plants, which are heterozygous for induced mutations, do not show the mutant phenotypes. For this reason, M2 plants are generated from M2 seeds. Again, the detailed information such as the planting date, M2 seeds used, phenotypes and resultant M3 grains is documented in database, while developing M2 plants.

M2 plants are subjected to the research of “forward and reverse genetics” (Figure 1.1.). Forward genetics is used to investigate the mutated plant exhibiting desired phenotype. It starts with a phenotype and moves towards the discovery of the function of the responsible gene. The process demands the assistance of a database to record the data

collected. Varied bioinformatics tools are also required to facilitate the design of experiments in order to characterize responsible gene of which the interruption cause phenotype. Whereas forward genetics starts with the mutant and then leads to the gene, reverse genetics starts with the gene of interest and ends with the corresponding mutant. The approach is to identify the mutation in a particular gene first and then to investigate the consequence of the mutation. Fast neutron radiations cause several types of mutations including deletion; the deletion loci in chromosomes can be detected by PCR analysis using specific primers flanking the targeted genes. This is the basis of revere genetics screening of *M. truncatula* mutants mutagenized by fast neutron bombardments.

The objective of this project is to interrupt genes in *M. truncatula* as many as possible, and then to identify each individual M2 plant carrying the deletion of gene of interest. It has been estimated that approximately 100,000 M2 mutant plants will be generated in order to mutagenize most genes in the model legume. Obviously, it is time-consuming and labor-intensive if the mutation of each mutated plant is identified individually. Therefore, an efficient and high-throughput approach must be applied to this kind of large-scale association study. One recent technology to address the cost, time and labor that are involved in large-scale mutation screening is to carry out analyses not on individual DNA samples, but on pools made up of DNA from many individuals [3].

To reduce the number of PCR analyses as much as possible, the laboratory in the Samuel Roberts Noble Foundation pools the seedling tissues of M2 plants both within a grid and across grids (Figure 1.2.). These pooled tissues are extracted for DNA samples serving as the templates of PCR-based screening. To pool tissue samples within a grid and across grids, the numbers of samples of each pool has to be identified first.

Obviously, computing tools are required to facilitate the large-scale pooling of seedling tissues. On the other hand, computing tool is also required to locate grid address that contains the tissue from a particular plant carrying mutation based on PCR screening.

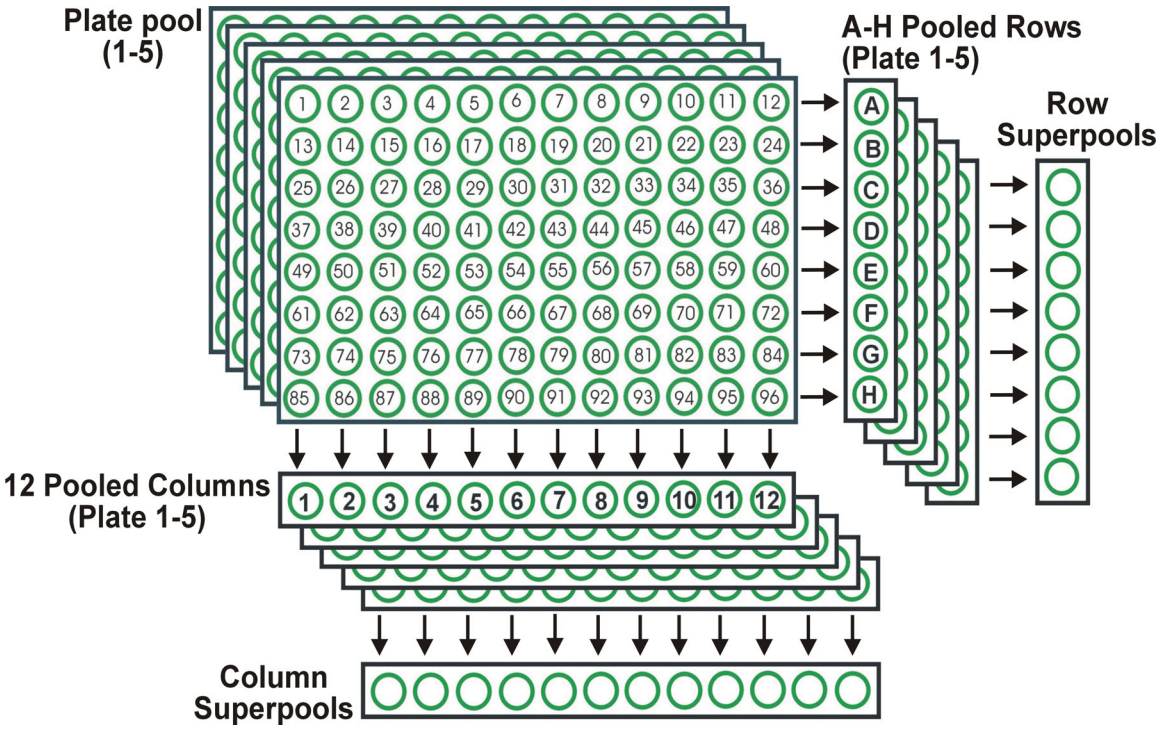


Figure 1.2. Three-dimensional pooling of tissues samples for DNA extraction. Each well of one 96-well plate represents tissues from 5 M2 plants. Tissues from 96-wells are pooled together to form a plate pool. Tissues from the same column and row of 5 96-well plates are also pooled to form column superpool (1-12) and row superpool (A-H).

1.2. Objectives of the thesis

There are two objectives of the thesis:

1) Develop a computing tool to facilitate the construction and subsequent utilization of an *M. truncatula* mutant library.

- Develop a tool that generates the list of tissues samples for plate, column and row pooling (3-dimensional pooling).
- Develop a tool that calculates the grid address (i.e., the numbers of plate, column and row) of a given DNA sample. The tool also shows the grid address graphically.
- Develop a tool that calculates the number of DNA sample according to PCR-screening (i.e., the numbers of plate, column and row that gives rise to positive results). The tool also shows grid address graphically.

2) Develop a relational database to assist in the management of mutant populations and their progenies, and to record, then store and exploit all data generated within the project.

1.3. Organization of the thesis

The thesis consists of the following chapters: Chapter one introduces the background of the biological research project and the embedded needs of computing tools and database management system. Chapter two is the literature review that describes the technological details of computer science, which are applied in the thesis. Chapter three describes the development of computing tools that facilitates the pooling of tissues samples, and the locating the grid address of tissues sample. The practical tests of these tools are performed. Chapter four focuses on the construction of a relational database that manages the day-by-day operation and all information generated whiling creating the mutant library of *M. truncatula*. Chapter five is the summary and proposed future work.

CHAPTER II

LITERATURE REVIEW

2.1 Component Object Model

Component Object Model (COM) is a Microsoft platform for software component introduced by Microsoft in 1993. It provides a standard mechanism by which objects can communicate regardless of what language is used to create the components [4]. COM defines a structure for building program routines (objects) that can be called up and executed in a Windows environment. This capability was built into Windows 95/98 and Windows NT 4.0. Parts of Windows itself and Microsoft's own applications are also built as COM objects. COM provides the interfaces between objects, and Distributed COM (DCOM) allows them to run remotely. COM is used in the following ways.

1) COM Objects: COM objects can be small or large. They can be written in any of several programming languages, and they can perform any kind of processing.
2) Automation (OLE automation): Standard applications, such as word processors and spreadsheets, can be written to expose their internal functions as COM objects, allowing them to be "automated" instead of manually selected from a menu.

3) Controls: Applications can invoke COM objects, called "controls," that blend in and become just another part of the program.
4) Compound Documents and ActiveX Documents: Microsoft's OLE compound documents are based on COM, which lets one document be embedded within or linked to another (OLE). ActiveX Documents are extensions to OLE that allow a Web browser, for example, to view not only Web pages, but also any kind of document.
5) Programming Interfaces: Increasingly, Microsoft is making its standard programming interfaces conform to the COM object model so that there is continuity among all interfaces.

COM includes interfaces and API functions that expose operating system services, as well as other mechanisms necessary for a distributed environment (naming, events, etc.) [5]. These are referred to as COM technologies (or services), and are shown in detail as follow:

- *Type Information*: Some clients need runtime access to type information about COM objects. This type information is generated by the Microsoft IDL compile and is stored in a type library. COM provides interfaces to navigate the type library.
- *Structured Storage and Persistence*: COM objects need a way to store their data when they are not running. The process of saving data for an object is called making an object persistent. COM supports object persistence through "Structured Storage", which creates an analog of a file system within a file. Individual COM objects can store data within the file, thus providing persistence.

- *Monikers*: Clients often require a way to allow them to connect to the exact same object instance with the exact same state at a later point in time. This support is provided via "monikers". A moniker is a COM object that knows how to create and initialize the content of a single COM object instance. A moniker can be asked to bind to the COM object it represents, such as a COM object residing on specific machine on the network, or a group of cells inside a spreadsheet.
- *Uniform Data Transfer*: COM objects often need to pass data amongst themselves. Uniform Data Transfer provides for data transfers and notifications of data changes between a source called the data object, and something that uses the data, called the consumer object.
- *Connectable Objects:* Some objects require a way to notify clients that an event that has occurred. COM allows such objects to define outgoing interfaces to clients as well as incoming interfaces. The object defines an interface it would like to use (e.g., a notification interface) and the client implements the interface.

The advantages of COM are as follows:

- *COM promotes component-based software development* Before component-based development came, software programs have been coded using procedural programming paradigm, which supports linear form of program execution. But component-based program development comes with a number of advantages, such as the ability to use pre-packaged components and tools from third party vendors into an application and support for code reusability in other parts of the same application.

- *COM promotes code reusability* Standard classes are normally reused in the same application but not easily used in other applications; however, COM components are designed to separate themselves from single applications and hence can be accessed and used by several different applications without any hassle.
- *COM promotes Object-oriented programming (OOP)* The primary characteristics of OOP are encapsulation, which allows the implementation details of an object to be hidden, polymorphism, which is the ability to exhibit multiple behaviors, and inheritance, which allows for the reuse of existing classes in order to design new and more specialized classes. Among these, encapsulation is one of COM's most important characteristics. Encapsulation helps to hide how an object has implemented a method internally. This ultimately helps to incorporate more vigorously implemented or advanced implementation into an object at later time without affecting the client which uses it.
- *COM comprises the necessary mechanisms for COM components to communicate with each other* In the normal case, two components coded using two different programming languages cannot communicate with each other. But COM can make it possible for different language components that adhere to the COM specification to interact with each other, and hence COM is language-independent.
- *COM helps to access components loaded in different machines on the network* COM component can reside anywhere on any computer or computer connected to a network. That is, applications using COM can access and share COM components regardless of their locations. Thus COM provides location transparency and COM components are location independent.

2.2 ActiveX Control

An ActiveX control is an embeddable COM object that is implemented as an in-process server DLL. ActiveX is the name Microsoft has given to a set of "strategic" object-oriented programming technologies and tools. The main technology is the Component Object Model (COM).

One of the main advantages of a component is that it can be re-used by many applications (referred to as component containers) [6]. A COM component object (ActiveX control) can be created using one of several languages or development tools, as Delphi, Visual C++, Borland C++, Visual Basic, and PowerBuilder, or with scripting tools such as VBScript. ActiveX controls can be used in a variety of environments not traditionally associated with programming, such as Microsoft Word, Microsoft Excel, Lotus, Hypertext Markup Language (HTML), and Internet Explorer [6].

ActiveX controls expose themselves to the outside world and can be used in a variety of environments. ActiveX controls are similar to embedded object servers, in that they are embedded in a container and are responsible for providing a user interface. ActiveX controls take advantage of the capability to send events to their container; this capability to send events separates ActiveX controls from other in-process OLE servers [7].

ActiveX controls communicate with the outside world in three ways:

- ***Properties***: Properties are named attributes or characteristics of an ActiveX control. Properties can be marked as read-only, but typically these properties can be set or queried.

- ***Methods***: Methods are functions performed by the control to access the control's functionality. These functions enable an external source to manipulate the appearance, behavior, or properties of the control.
- ***Events***: Events are notifications generated by the control to provide some sort of notification to the container. Usually, this is input by the user, such as a mouse click or keyboard input.

2.3 ActiveX Data Object.NET (ADO.NET)

ADO.NET is a set of computer software components that can be used by programmers to access data and data services. It is a part of the base class library that is included with the Microsoft .NET Framework. It commonly is used by programmers to access and modify data stored in relational database systems, though it can also be used to access data in non-relational sources [8, 9].

ADO.NET consists of two primary parts:

A. Data provider These classes provide access to a data source, such as a Microsoft SQL Server or an Oracle database. Each data source has its own set of provider objects, but they each have a common set of utility classes:

1) Connection: Provides a connection used to communicate with the data source. Also acts as an abstract factory for command objects.
2) Command: Used to perform some action on the data source, such as reading, updating, or deleting relational data.
3) Parameter: Describes a single parameter to a command. A common example is a parameter to a stored procedure.

4) DataAdapter: A bridge used to transfer data between a data source and a DataSet object.
5) DataReader: An object used to efficiently process a large list of results one record at a time without storing them.

B. DataSet DataSet objects, a group of classes describing a simple in-memory relational database, were the star of the show in the initial release (1.0) of the Microsoft .NET Framework. The classes form a containment hierarchy:

1) A DataSet object represents a schema (either an entire database or a subset of one). It can contain tables and relationships between those tables.
2) A DataTable object represents a single table in the database. It has a name, rows, and columns.
3) A DataView object "sits over" a DataTable and sorts the data (much like a SQL order by clause) and filters the records (much like a SQL where clause) if a filter is set. An in-memory index is used to facilitate these operations. All DataTables have a default filter, while any number of additional DataViews can be defined, reducing interaction with the underlying database and thus improving performance.
4) A DataColumn represents a column of the table, including its name and type.
5) A DataRow object represents a single row in the table, and allows reading and updating of the values in that row, as well as retrieving any rows that are related to it through a primary-key foreign-key relationship.

6) A DataRowView represents a single row of a DataView; the distinction between a DataRow and DataRowView is important when enumerating a result set.
7) A DataRelation is a relationship between tables, such as a primary-key foreign-key relationship. This is useful for enabling DataRow's functionality of retrieving related rows.
8) A Constraint describes an enforced property of the database, such as the uniqueness of the values in a primary key column.

A DataSet is populated from a database by a dataAdapter whose Connection and Command properties have been set. However, a DataSet can save its contents to XML (optionally with an XSD schema), or populate itself from XML, making it exceptionally useful for web services, distributed computing, and occasionally-connected applications.

2.4 Relational Database and Structured Query Language

2.4.1 Relational Database

The relational database model, first developed by E.F. Codd (of IBM) in 1970, represents a major breakthrough for both users and designers [10]. The relational database model is implemented through a very sophisticated relational database management system (RDBMS). The RDBMS performs the same basic functions provided by the hierarchical and network DBMS system plus a host of other functions that make the relational database model easier to understand and to implement. Arguably the most important advantage of the RDBMS is its ability to let the user/designer operate in a human logical environment.

The RDBMS manages all of the complex physical details. Thus, the relational database is perceived by the user to be a collection of tables in which data are stored.

The relational database is a single data repository in which data independence is maintained. However, the relational database model adds significant advantages as follow [11]:

- Ensuring data integrity.
- Storing data storage efficiently.
- Giving your database application tremendous room for growth.
- Creating a database that behaves predictably because it conforms to these well-tested rules.
- Enabling other database designers to understand your database because it follows the rules.
- Ensuring that database schema changes are easy to implement.
- Improving the speed of data access.

However, the relational database's substantial advantages over the hierarchical and network databases are purchased at the cost of some disadvantages as follows:

- Substantial hardware and system software overhead.
- Poor design and implementation is made easy.
- May promote "islands of information" problems.

2.4.2 Structured Query Language (SQL)

Structured Query Language (SQL) - A standardized language that approximates the structure of natural English for obtaining information from database, developed by IBM

Research in the mid-1970s [12]. It is a standard interactive and programming language for getting information into and out of relational database management system. The language has evolved beyond its original purpose to support object-relational database management systems. It is an ANSI (American National Standards Institute) and an ISO standard for accessing database systems.

SQL allows users to access data in relational database management systems, such as Access, Sybase, FileMaker Pro, Microsoft SQL Server, Informix, Oracle, and others, by allowing users to query, create, insert, delete, find, modify, retrieve, update, store, manage the data the user wishes to see. SQL also allows users to define the data in database, and manipulate that data [13, 14].

There are many features of SQL as follows:

- ***Simplicity*** – Several problems can be expressed in SQL more easily and concisely than in lower level languages. Simplicity means increased productivity.
- ***Completeness*** – The language is relatively complete. i.e., for a large class of queries users need not use loops or branching.
- ***Nonproceduriality*** – A Language such as the SQL Data Manipulation Language (DML) is known as a "nonprocedural" language. A SELECT statement specifies only what data is wanted, not a procedure for obtaining that data.
- ***Data independence*** – SQL DML statements do not contain any reference to explicit access paths such as indexes or physical sequence. Thus, the SQL

DML provides total “physical” data independence; i.e., independence of the way in which the data is physically stored.

To process an SQL statement, a relational database management system (RDBMS) performs the following five steps:

1) The RDBMS first parses the SQL statement. It breaks the statement up into individual words, called tokens, and ascertains that the statement has a valid verb and valid clauses, and so on. Syntax errors and misspellings can be detected in this step.
2) The RDBMS validates the statement. It checks the statement against the system catalog. Do all the tables named in the statement exist in the database? Do all of the columns exist, and are the column names unambiguous? Does the user have the required privileges to execute the statement? Certain semantic errors can be detected in this step.
3) The RDBMS generates an access plan for the statement. The access plan is a binary representation of the steps that are required to carry out the statement; it is the DBMS equivalent of executable code.
4) The RDBMS optimizes the access plan. It explores various ways to carry out the access plan. Can an index be used to speed a search? Should the RDBMS first apply a search condition to Table A and then join it to Table B, or should it begin with the join and use the search condition afterward? Can a sequential search through a table be avoided or reduced to a subset of the table? After exploring the alternatives, the RDBMS chooses one of them.
5) The RDBMS executes the statement by running the access plan.

2.5 Configuration (config) File and Initialization (INI) File

2.5.1 Configuration (config) File

In computing, configuration files, or config files, are used to configure the initial settings for some computer programs [15]. They are used for user applications, server processes and operating system settings. The files are often written in ASCII and line-oriented, with lines terminated by a newline or carriage return/line feed pair, depending on the operating system.

Some computer programs only read the configuration (config) files at startup. Others periodically check the configuration files for changes. Some can be told to re-read the configuration files and apply the changes to the current process, or indeed to read arbitrary files as a configuration file.

The general format of a configuration file is quite simple. Each line contains a keyword and one or more arguments. For simplicity, most lines only contain one argument. Comment lines are blank lines or lines that start with a '#'.

For example:

```
<name>:<whitespace><value><newline>
```

The <name> contains any alphanumeric character or underline (_). The <value> can include any character except newline. It also cannot start with either spaces or tabs since those are considered part of the whitespace after the colon.

2.5.2 Initialization (INI) File

An initialization file or INI file that has a .INI extension and contains configuration information for MS-Windows based applications [16]. Starting with Windows 95, the INI

file format was superseded but not entirely replaced by a registry database in Microsoft operating systems. Although made popular by Windows, INI files can be used on any system because of their flexibility. They allow a program to store configuration data, which can then be easily parsed and changed.

A typical INI file format might look like this:

```
[section1]

; some comment on section1
var1 = foo
var2 = 451

[section2]

; another comment
var1 = 123
var2 = bar
```

- ***Sections***: Section declarations start with '[' and end with ']' as in [section1] and [section2] above. And sections start with section declarations.
- ***Parameters***: The "var1 = foo" above is an example of a parameter (also known as an **item**). Parameters are made up of a key ('var1'), equals sign ('='), and a value ('foo').
- ***Comments***: All the lines starting with a ';' are assumed to be comments, and are ignored.

2.6 Visual Basic.NET (VB.NET), Relation to Visual Basic and Microsoft Access (MS Access)

2.6.1 Visual Basic.NET (VB.NET)

Visual Basic .NET (VB.NET) is an object-oriented computer language that can be viewed as a evolution of Microsoft's Visual Basic (VB) implemented on the Microsoft .NET framework. Its introduction has been controversial, as significant changes were made

that broke backward compatibility with VB and caused a rift within the developer community [17].

The great majority of VB.NET developers use Visual Studio .NET as their integrated development environment (IDE). SharpDevelop provides an open-source alternative IDE. There are advantages of using VB.NET as follows [18]:

- Problems can be solved easily and effectively.
- It is possible to create web applications with a zero learning curve.

Like all .NET languages, programs written in VB.NET require the .NET framework to execute. The .NET Framework offers a number of advantages to developers as follows:

- It is a consistent programming model.
- It has direct support for security.
- It has simplified development efforts.
- It has easy application deployment and maintenance.

2.6.2 Relation to Visual Basic

Whether Visual Basic .NET should be considered as just another version of Visual Basic or a completely different language is a topic of debate. One simple change that can be confusing to previous users is that of Integer and Long data types, which have each doubled in length; a 16-bit integer is known as a Short in VB.NET, while Integer and Long are 32 and 64 bits respectively. Similarly, the Windows Forms GUI editor is very similar in style and function to the Visual Basic form editor [19].

The things that have changed significantly are the semantics [20]**.** The changes have altered many underlying assumptions about the "right" thing to do with respect to

performance and maintainability. Some functions and libraries no longer exist; others are available, but not as efficient as the "native" .NET alternatives. Even if they compile, most converted VB6 applications will require some level of refectory to take full advantage of the new language. Extensive documentation is available to cover changes in the syntax, debugging applications, deployment and terminology.

2.6.3 Microsoft Access (MS Access)

Microsoft Access is a popular relational database management system for creating, managing desktop and client/server database applications that run under the Windows operating system. It was packaged with Microsoft Office Professional which combines the relational Microsoft Jet Database Engine with a graphical user interface [4].

Microsoft Access can use data stored in Access/Jet, Microsoft SQL Server, Oracle, or any ODBC-compliant data container [21]. It allows relatively quick development because all database tables, queries, forms, and reports are stored in the database. For query development, Access utilizes the Query Design Grid, a graphical user interface that allows users to create queries without knowledge of the SQL programming language.

One of the benefits of Access from a programmer's perspective is its relative compatibility with SQL—queries may be viewed and edited as SQL statements, and SQL statements can be used directly as Macros and VBA Modules to manipulate Access tables [21].

There are four features for using Access as follow:

- *Access Systems are Fast to Develop*. Due to the nature of Microsoft Access it is possible to get a system up and running in much less time than with other development environments.

- *Access Systems are low risk.* Due to the widespread use of Microsoft Systems worldwide one should never find oneself in the situation where the system will have to "be rewritten" because it's out of date.
- *Fast to Modify.* If one has a custom report that must be included as soon as possible, then Microsoft Access can allow the report to be written without disrupting use of the system.
- *Link Systems to Other Applications.* Sometimes this can be done directly, sometimes through an import/export process.

2.7 DNA Pooling

DNA pooling is a method for reducing the burden of genotyping large numbers of individuals. In DNA pooling, individuals' DNA specimens are combined into one sample, and that sample is genotyped to estimate allelic frequencies in the original population. Pooling allows allele frequencies in groups of individuals to be measured using far fewer PCR reactions and genotyping assays than are used when genotyping individuals [3].

Two advantages of DNA pooling are:

- It is a powerful and efficient tool for high throughput association analysis.
- It significantly reduces the consumable and labor costs of a study.

CHAPTER III

DESIGN AND IMPLEMENTATION OF MEDICAGO COMPUTING TOOL

3.1. The design of the computing tool

3.1.1 System functioning

According to the requirement for tissue pooling for DNA isolation as described in Chapter 1, the computing tool is employed to implement the display of a DNA sample location pooled in 3-dimensional manner, and the generation of a DNA sample list for pooling. It includes the following computing functions:

1) Calculate the grid location (i.e., the numbers of plate, column and row) and unit number (5 plates per unit) of each pooled DNA sample according to the number of DNA sample.
2) Calculate the number of DNA samples and its unit number belonging according to the grid address (i.e., the numbers of plate, column and row) resulting from PCR screening.
3) Calculate and generate the list of the numbers of DNA samples in certain column of each unit according to the numbers of unit and column.
4) Calculate and generate the list of the numbers of DNA samples in certain row of each unit according to the numbers of unit and row.

The scientists at the Noble Foundation prefer to lookup the graphics display of the grid address of the DNA sample in the 96-wells plate format, which consists of 12 columns (C1, C2, C3 … C12) and 8 rows (RA, RB, RC … RH) (Figure 3.1); therefore, the graphics display was programmed to show the grid address of the DNA sample in a 96-well plate according to the number of the DNA sample or the grid address of the DNA sample resulting from PCR-screening.

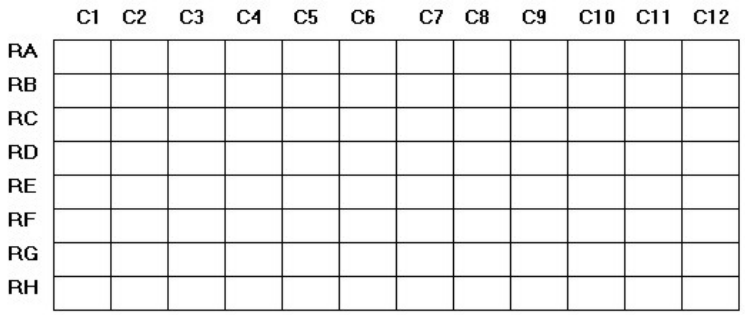


Figure3.1. Graphic display of grid address of DNA samples in 96-well plate format.

3.1.2 Construction of user friendly interface

The computing tool was written in the Visual Basic 6.0 language using an object-oriented methodology. It operates in Windows XP. The application of each function is selectable from the main interface. Tables 3.1-4 show objects designed for 4 function interfaces.

Table 3.1 Properties of interface that implements the display of grid address and unit number of DNA sample

Object	Property	Setting
frmGrid (Form)	Caption	Location
	Height	7995
	Width	12000
Label1	Caption	Display Grid Location of Each Pooled DNA Sample From Mutant Lines
Label2	Caption	Input No. of DNA Sample
Label3	Caption	Location
Label4	Caption	(0 ~ 11): C1 – C 12 (12 ~ 19): RA - RH
Shape	Shape	0-Rectangle (0 ~ 95)
	Height	400
	Width	621
TextBox1	Input	Number of DNA Sample > 0
TextBox2	Data Field	Grid Location (P-Plate; C-Column; R-Row; U-Unit)
CommandButton1	Caption	Display Grid Location of DNA Sample
CommandButton2	Caption	Clear All
CommandButton3	Caption	Back
CommandButton4	Caption	Exit

Table3.2 Properties of interface that implements the display of DNA sample number

Object	Property	Setting
frmCertainLocation (Form)	Caption	Number of DNA Samples
	Height	7905
	Width	10800
TextBox1	Input	Number of plate > 0
TextBox2	Input	0 < column number< 13
TextBox3	Input	From (A or a) to (H or h) Uppercase or Lowercase
TextBox4	Text	No. of DNA Sample (C-Column; R-Row, P-Plate, U-Unit)
CommandButton1	Caption	Display
CommandButton2	Caption	Clear
CommandButton3	Caption	Back
CommandButton4	Caption	Exit
Shape	Shape	0-Rectangle (0 ~ 95)
	Height	400
	Width	621
Label1	Caption	Input Plate Number
Label2	Caption	Input Column
Label3	Caption	Input Row
Label4	Caption	No. of DNA Samples
Label5	Caption	(0 ~ 11): C1 - C12 (12 ~ 19): RA - RH

Table3.3 Properties of interface that generates DNA samples list for column pooling

Object	Property	Setting
frmDNAColumn (Form)	Caption	Column
	Height	6800
	Width	12000
Label1	Caption	Input Unit 1 Unit = 5 Plates = 480 (Wells)
lblCol (Label)	Caption	Input Column
lblColNum (Label)	Caption	No. of DNA Sample
lblTitle (TextBox)	Data Field	Display DNA Samples of Mutant Lines in Certain Column of Each Unit
txtUnit (TextBox)	Input	> 0
txtCol (TextBox)	Input	0 < column number < 13
Text1 (TextBox)	Text	List of No. of DNA sample
CommandButton1	Caption	Proceed
CommandButton2	Caption	Display No. of DNA Samples
CommandButton3	Caption	Clear All
CommandButton4	Caption	Back
CommandButton5	Caption	Exit

Table3.4 Properties of interface that generates DNA samples list for row pooling

Object	Property	Setting
frmDNARow (Form)	Caption	Row
	Height	6880
	Width	12000
Label1	Caption	Display DNA Samples of Mutant Lines in Certain Row of Each Unit
Label2	Caption	Input Unit: 1 Unit = 5 Plates = 480(Wells)
Label3	Caption	Input Row From (A or a) to (H or h)
Label4	Caption	No. of DNA Sample:
Text1 (TextBox)	Data member	List of No. of DNA Sample
txtRow (TextBox)	Data Input	(A or a) to (H or h) Uppercase or Lowercase
txtUnit (TextBox)	Data Input	> 0
Command1	Caption	Proceed
Command2	Caption	Display No. of DNA Sample
Command3	Caption	Clear All
Command4	Caption	Back
Command5	Caption	Exit

3.1.3 Graphic Display of a Grid Address

To achieve the graphics indication of DNA sample in 96-well plate format, I used shape control built in Visual basic to show 96 wells. They were organized into 12 Columns and 8 Rows. The algorithm for graphic displays was the following codes:

```
Select Case Text9.Text
        Case "A"
        Case "a"
           R = 1
           ID = 12 * (R - 1) + Text8.Text + 96 * (Text7.Text - 1)
           If ID Mod 480 = 0 Then
              Unit = ID \ 480
           Else
              Unit = ID \ 480 + 1
```

```
    End If
    Text6.Text = ID & "    " & "(Unit is: " & Unit & ")"
    For I = 0 To 95
     If (I) \ 12 + 1 = R Or (I) Mod 12 + 1 = Text8.Text Then
       Shape1(I).FillColor = &HC000&
       Shape1(I).FillStyle = 7
     End If
    Next I
Case "B"
Case "b"
    R = 2
    ID = 12 * (R - 1) + Text8.Text + 96 * (Text7.Text - 1)
    If ID Mod 480 = 0 Then
       Unit = ID \ 480
    Else
       Unit = ID \ 480 + 1
    End If
    Text6.Text = ID & "    " & "(Unit is: " & Unit & ")"
    For I = 0 To 95
       If (I) \ 12 + 1 = R Or (I) Mod 12 + 1 = Text8.Text Then
         Shape1(I).FillColor = &HC000&
         Shape1(I).FillStyle = 7
       End If
    Next I
Case "C"
Case "c"
    R = 3
    ID = 12 * (R - 1) + Text8.Text + 96 * (Text7.Text - 1)
    If ID Mod 480 = 0 Then
       Unit = ID \ 480
    Else
       Unit = ID \ 480 + 1
    End If
    Text6.Text = ID & "    " & "(Unit is: " & Unit & ")"
    For I = 0 To 95
       If (I) \ 12 + 1 = R Or (I) Mod 12 + 1 = Text8.Text Then
          Shape1(I).FillColor = &HC000&
          Shape1(I).FillStyle = 7
       End If
    Next I
Case "D"
Case "d"
    R = 4
    ID = 12 * (R - 1) + Text8.Text + 96 * (Text7.Text - 1)
    If ID Mod 480 = 0 Then
       Unit = ID \ 480
```

```
    Else
       Unit = ID \ 480 + 1
    End If
    Text6.Text = ID & "    " & "(Unit is: " & Unit & ")"
    For I = 0 To 95
       If (I) \ 12 + 1 = R Or (I) Mod 12 + 1 = Text8.Text Then
          Shape1(I).FillColor = &HC000&
          Shape1(I).FillStyle = 7
       End If
     Next I
Case "E"
Case "e"
    R = 5
    ID = 12 * (R - 1) + Text8.Text + 96 * (Text7.Text - 1)

    If ID Mod 480 = 0 Then
       Unit = ID \ 480
    Else
       Unit = ID \ 480 + 1
    End If
    Text6.Text = ID & "    " & "(Unit is: " & Unit & ")"
    For I = 0 To 95
       If (I) \ 12 + 1 = R Or (I) Mod 12 + 1 = Text8.Text Then
          Shape1(I).FillColor = &HC000&
          Shape1(I).FillStyle = 7
       End If
     Next I
Case "F"
Case "f"
    R = 6
    ID = 12 * (R - 1) + Text8.Text + 96 * (Text7.Text - 1)

    If ID Mod 480 = 0 Then
       Unit = ID \ 480
    Else
       Unit = ID \ 480 + 1
    End If
    Text6.Text = ID & "    " & "(Unit is: " & Unit & ")"
    For I = 0 To 95
       If (I) \ 12 + 1 = R Or (I) Mod 12 + 1 = Text8.Text Then
          Shape1(I).FillColor = &HC000&
          Shape1(I).FillStyle = 7
       End If
    Next I
Case "G"
Case "g"
```

```
        R = 7
        ID = 12 * (R - 1) + Text8.Text + 96 * (Text7.Text - 1)

        If ID Mod 480 = 0 Then
           Unit = ID \ 480
        Else
           Unit = ID \ 480 + 1
        End If
        Text6.Text = ID & "    " & "(Unit is: " & Unit & ")"
        For I = 0 To 95
           If (I) \ 12 + 1 = R Or (I) Mod 12 + 1 = Text8.Text Then
              Shape1(I).FillColor = &HC000&
              Shape1(I).FillStyle = 7
           End If
        Next I
     Case "H"
     Case "h"
        R = 8
        ID = 12 * (R - 1) + Text8.Text + 96 * (Text7.Text - 1)

        If ID Mod 480 = 0 Then
           Unit = ID \ 480
        Else
           Unit = ID \ 480 + 1
        End If
        Text6.Text = ID & "    " & "(Unit is: " & Unit & ")"
        For I = 0 To 95
           If (I) \ 12 + 1 = R Or (I) Mod 12 + 1 = Text8.Text Then
              Shape1(I).FillColor = &HC000&
              Shape1(I).FillStyle = 7
           End If
        Next I
     Case Else
        MsgBox " invalid Row Number, Please Input Again!!!"
        Text6.Text = ""
        Text9.Text = ""
End Select
```

3.2. The implementation of the computing tool

The main interface of the software package in the Windows system is shown in Figure 3.2. It was named the Medicago Computing Tool. The following sections demonstrate each function of the software.

Figure 3.2. Interface of computing tools to facilitate DNA sample pooling and screening of *M. truncatula* mutants.

3.2.1. Lookup of the grid address of a DNA sample of mutant lines

The scientists at the Noble Foundation pool tissue samples from multiple mutant lines prior to DNA isolations. The pooling architecture is described in Chapter 1 (Figure 1. 2). Since the DNA samples of ~100,000 mutant lines are pooled, the localization of a particular DNA sample is difficult without the aid of a computing tool. One of the

functions of the Medicago Tool is to look up the grid address of DNA samples (Figure 3.3). For instance, if a user would like to check the location of a DNA sample, No. 5806 that is input into the software through the input panel (Figure 3.3), the grid address of the DNA sample is displayed when the user clicks the button of "Display Grid Locations of DNA Samples". The software thereafter indicates that the DNA sample is in Plate (P) 61, Column (C) 10 and Row (R) D. As mentioned in Chapter 1, one pooling unit consists of DNA samples of 5 plates. The users would like to know the unit number when they look at the grid address of any specific DNA sample; therefore, the software also is designed to display the Unit number. As shown in Figure 3.3, for example, DNA sample No. 5806 belongs to Unit 13. In addition, the users at the Noble Foundation prefer to view the graphics display of grid address of a DNA sample on a plate. The software, therefore, was written to satisfy this requirement. On the interface of grid address lookup, overlapping shadowed cells indicate DNA sample location on a 96-well plate (Figure 3.3). To look up the location of new DNA samples, users can click the button, "Clear All", to erase existing input and display. The "Back" button allows the user to go back to main menu for other applications. The "Exit" button allows the user to log off the computing tool system.

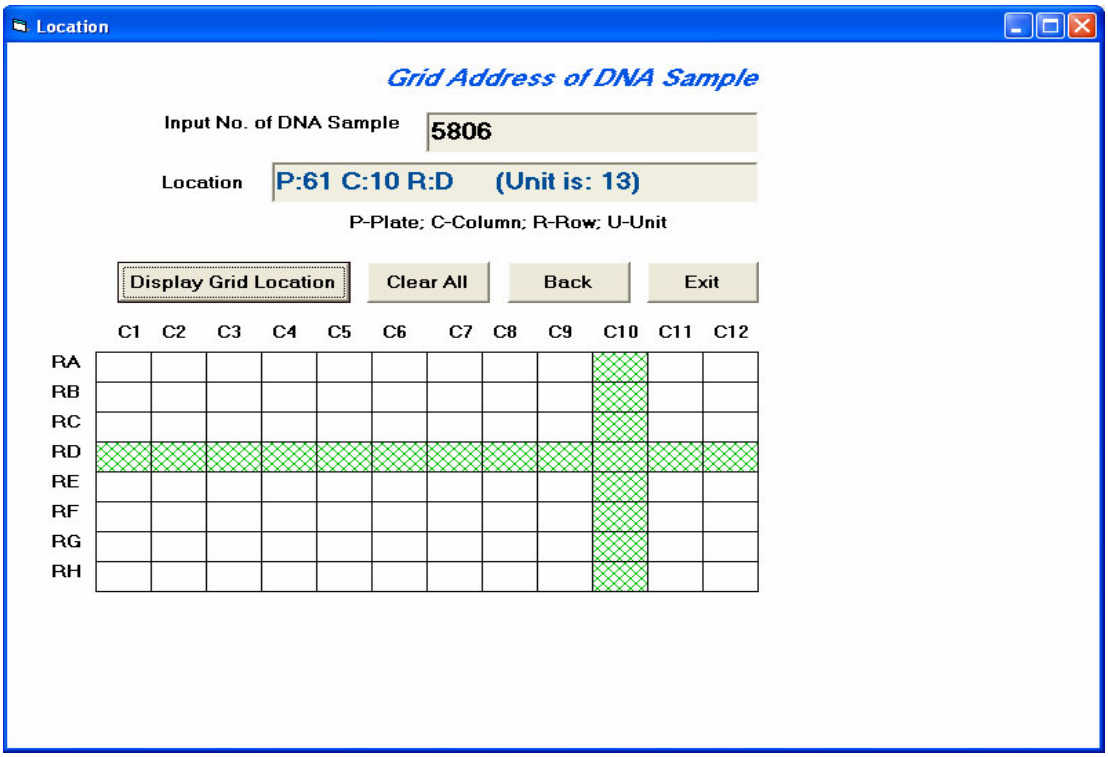


Figure 3.3 Interface of grid address lookup of DNA samples

3.2.2. Lookup of DNA sample Number according to PCR screening results

The scientists at the Noble Foundation conduct PCR reactions to screen pooled DNA samples in order to identify a desired mutant line. As described in Chapter 1, one pooling unit consists of 25 pooled DNA samples; 25 PCR reactions with the 25 pooled DNA samples allow to screening 2400 mutant lines. For each target gene, specific primer pair is designed to run the 25 PCR reactions. Three positive signals; i.e., three identical PCR products appear if one mutant carrying deletion in a given target gene exits in the 2400 mutant lines (Figure 3.4). The Medicago tool allows the user to obtain the DNA sample number of a particular mutant group (5 mutants per well) that gives rise to the positive screening results.

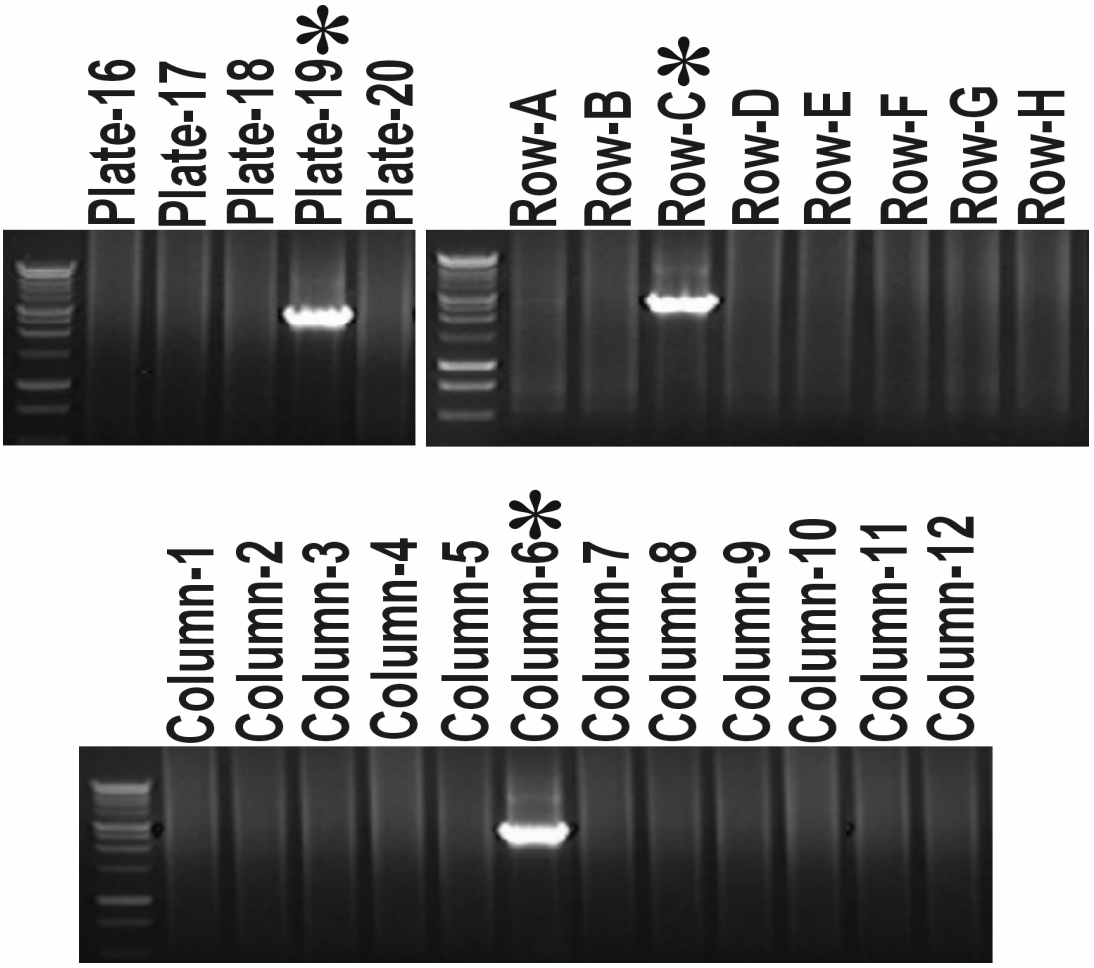
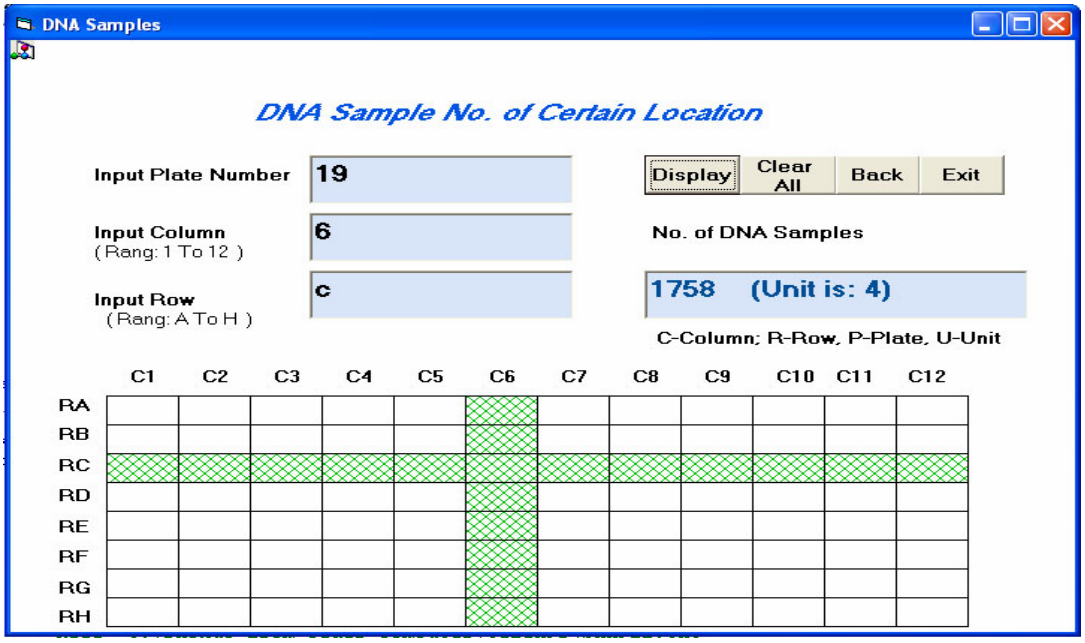


Figure 3.4 Example of 3-dimenional PCR-based screenings of one pooling unit consisting of 2400 mutant lines.

As shown in Figure 3.5, the user is able to input PCR results; i.e., the numbers of plate, row and column that give rise to positive signals. The information is processed when the button, "Display No. of DNA Samples", is clicked. For instance, the PCR screening results shown in Figure 3.4 indicate that the DNA sample from the grid address of Plate-19, Row-C and Column-6 carries a target mutation. To find out the DNA sample number of the grid address for further mutant analysis, user can input the PCR screening results on the interface of DNA sample No. Lookup (Figure 3.5). The DNA sample number and its unit number are displayed when clicking the button of "Display No. of

DNA Samples". The location of the DNA sample on 96-well plate also is indicated graphically on the same interface.



Row-E	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692
Row-F	1693	1694	1695	1696	1697	1698	1699	1700	1701	1702	1703	1704
Row-G	1705	1706	1707	1708	1709	1710	1711	1712	1713	1714	1715	1716
Row-H	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727	1728
Plate19	Col-1	Col-2	Col-3	Col-4	Col-5	Col-6	Col-7	Col-8	Col-9	Col-10	Col-11	Col-12
Row-A	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740
Row-B	1741	1742	1743	1744	1745	1746	1747	1748	1749	1750	1751	1752
Row-C	1753	1754	1755	1756	1757	**1758**	1759	1760	1761	1762	1763	1764
Row-D	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775	1776
Row-E	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788
Row-F	1789	1790	1791	1792	1793	1794	1795	1796	1797	1798	1799	1800
Row-G	1801	1802	1803	1804	1805	1806	1807	1808	1809	1810	1811	1812
Row-H	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823	1824
Plate20	Col-1	Col-2	Col-3	Col-4	Col-5	Col-6	Col-7	Col-8	Col-9	Col-10	Col-11	Col-12
Row-A	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836
Row-B	1837	1838	1839	1840	1841	1842	1843	1844	1845	1846	1847	1848
Row-C	1849	1850	1851	1852	1853	1854	1855	1856	1857	1858	1859	1860
Row-D	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871	1872

Figure 3.5 Example of lookup of DNA sample No. according to PCR screening results (up panel). The DNA sample location on data spreadsheet is shown in bold (low panel).

3.2.3. Display of a DNA sample list for row and column pooling

The plate pool consists of 96 DNA samples of which their numbers are in order. By contrast, column and row pools of each unit (5 plates) consist of 40 and 60 DNA samples across plates, respectively. Therefore, extreme care must be taken when preparing row and column pools in order to avoid mishandling. It is estimated that DNA samples of approximately 100,000 mutant lines are pooled in order to screen mutation in most given genes. The scientists at the Noble Foundation require a computing tool that generates the lists of DNA samples for column and row pooling. The Medicago Computing Tool includes the functions to assist the routine manipulations.

Figure 3.6 demonstrates the column pooling using the computing tool. To generate the DNA sample list for the pooling of column 11 in unit 58, the numbers of the unit (58) and the column (11) are input through the function interface. The numbers of 60 DNA samples are displayed. Users are able to paste and print the list of DNA samples for pooling as follows:

The list of DNA samples for the pooling of column 11 in unit 58: 27371, 27383, 27395, 27407, 27419, 27431, 27443, 27455, 27467, 27479, 27491, 27503, 27515, 27527, 27539, 27551, 27563, 27575, 27587, 27599, 27611, 27623, 27635, 27647, 27659, 27671, 27683, 27695, 27707, 27719, 27731, 27743, 27755, 27767, 27779, 27791, 27803, 27815, 27827, and 27839.

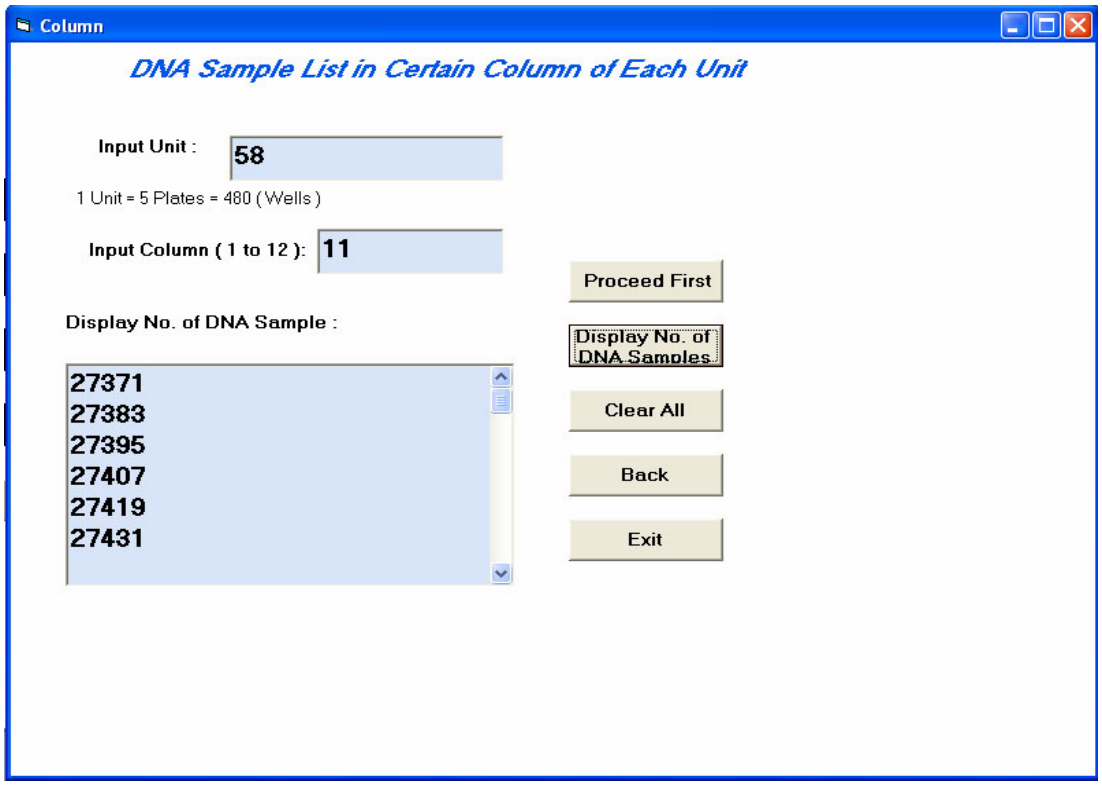


Figure 3.6 Example of generation of DNA sample list for column pooling

Likewise, the function interface as shown in Figure 3.7 was designed to generate the list of DNA samples for row pooling. For the pooling of row "g" in unit 129, the numbers of DNA samples are as follows:

The list of DNA samples for the pooling of row "g" in unit 129: 61513, 61514, 61515, 61516, 61517, 61518, 61519, 61520,61521, 61522, 61523, 61524, 61609, 61610, 61611, 61612, 61613, 61614, 61615, 61616, 61617, 61618, 61619, 61620, 61705, 61706, 61707, 61708, 61709, 61710, 61711, 61712, 61713, 61714, 61715, 61716, 61801, 61802, 61803, 1804, 61805, 61806, 61807, 61808, 61809, 61810, 61811, 61812, 61897, 61898, 61899, 61900, 61901, 61902, 61903, 61904, 61905, 61906, 61907 and 61908.

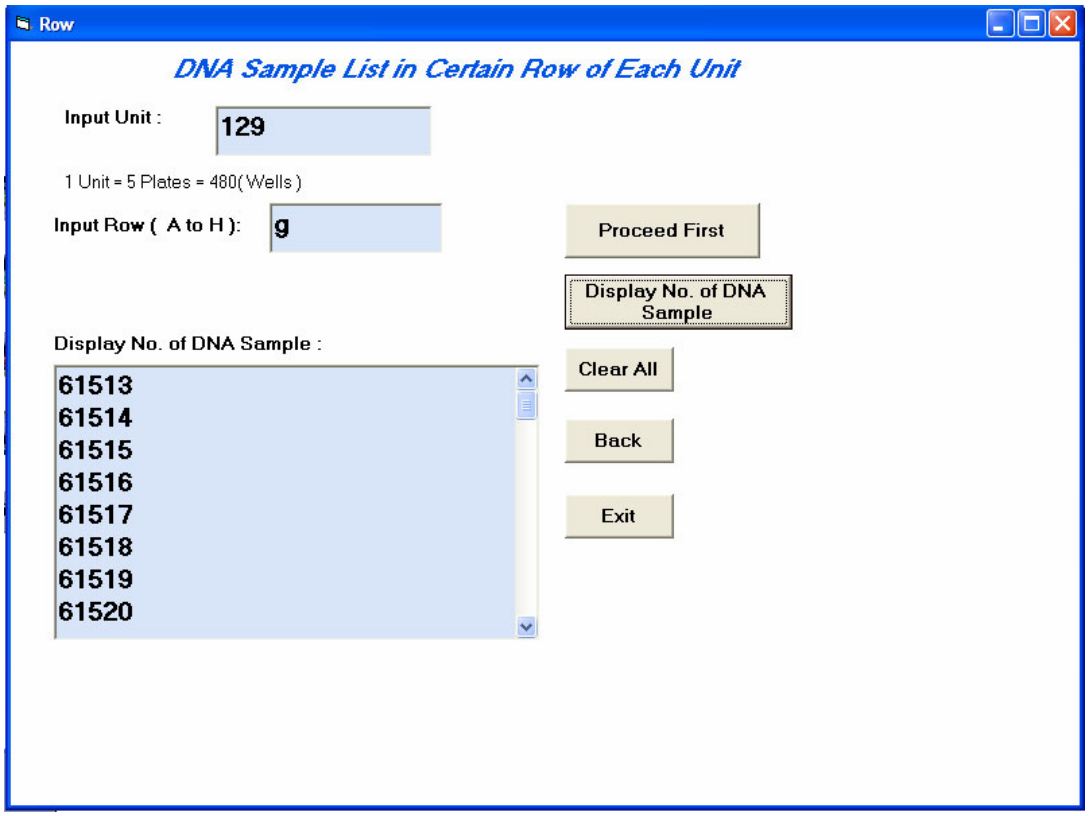


Figure 3.7 Example of row pooling of DNA sample using the computing tool.

CHAPTER IV

DEVELOPMENT AND IMPLEMENTATION OF MANAGEMENT SYSTEM FOR MEDICAGO MUTANT RESOURCES

4.1 Overall Structure Design of the Database Management System

One objective of the thesis is to develop a relational database to assist in the management of the *M. truncatula* mutant populations and their progenies, and to record, then store and exploit all data generated within the project. The relational database consists of three functional components: user management, data management and data analysis (Figure 4.1).

- User management: This component allows users of Noble Foundation to log on, input password and change password in order to maintain the security of the system.
- Data management: This component implements many functions such as day-by-day data input, sorting, bookmark, filter and so on.
- Data analysis and mutant image review: This component allows the users to monitor the progress of the *M. truncatula* mutagenesis project. It also allows to viewing and comparing mutant images in order to pick desired mutant individual for further biological characterization. The data statistics implemented through

this component provides information such as the progress of the project, the basis of the next-step plan.

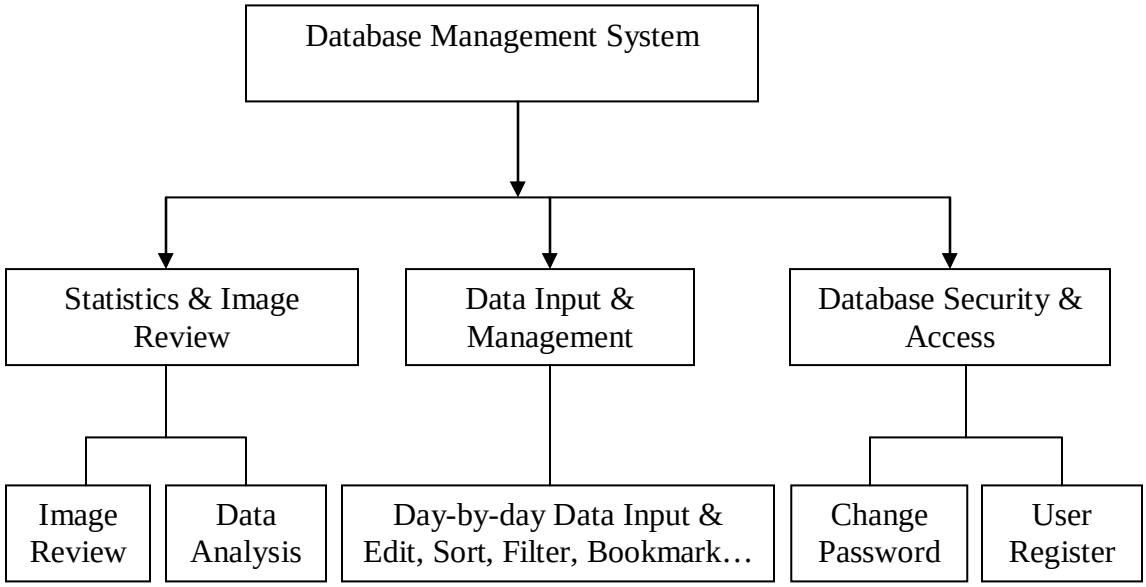


Figure 4.1 Flow Chart of the Management System for *M. truncatula* Mutant Resources

4.2 Database Normalization and E-R Diagram

4.2.1 Database Normalization

Data normalization is the process of designing a database and organizing data to take best advantage of relational database principles. Normalizing allows to reaching the following goals:

- Minimization of redundancy in data.
- Removal of insertion, deletion and updating of anomalies during database activities.
- Reduction of the need to reorganize data when it is modified or enhanced.

Flat Table A flat table as shown in Table 4.1 includes all fields of data generated in the *M. truncatula* mutagenesis project. The database has a designation of “Mutant”. Flat table

is not the most efficient design, and it consumes more physical space on hard drive than a set of normalized database tables. Therefore, normalized tables were generated thereafter.

Table 4.1 “Mutant” flat table

Field Name	Description
MutantM1ID	ID of mutant M1
PlantDate	The date of planting
GerminationDate	The data of seed germination
FlowerDate	The date of seed flowering
NumPlants	The number of plant grown
HarvestDate	The date of seed harvest
WeightofSeedBag	Weight of seed collected in each bag
Phenotype	Mutant phenotype description
OtherComment	Additional information input
ScreenedforNodulation	Mutant screened for nodulation phenotype
NoDNASampling	The number of tissue for DNA isolation
MutantM2ID	ID of mutant M2
DateofSeedTreatment	The date of seed treatment germination
DateofRemovalfromFungicideTreatment	The date of removing the fungicide reagent
Type/AmountofFungicideUsed(mL/Lwater)	Information of fungicide used such as concentration
DateofTransplantingtoPlate	The date of transplanting treated seeds on to germinating plate
DateofSeedlingInoculation	The date of seedling inoculated
DateofTransplantingtoPot	The date of transplanting seedlings to pot
Group	Batch number
ImageID	ID of mutant image
Image	Images of mutant

First Normal Form First normal form **(**1NF) excludes the possibility of repeating groups by requiring that each field in a database hold an atomic value, and that records be defined in such a way as to be uniquely identifiable by means of a primary key. In the flat table designed above (Table 4.1), there are many repeated sets of fields for the data of mutant generations 1 and 2. I identified “MutantM1” and “MutantM2” as its two distinct topics. Taking “Mutant” flat table to the first normal form would mean that I could create

two tables: one for “MutantM1” and one for “MutantM2”. Table 4.2 and 4.3 present the data fields in 1NF.

Table 4.2 “MutantM1” table

Field Name	***Description***
MutantID	A unique ID for mutant
MutantM1ID	ID of mutant M1. This field is primary key.
PlantDate	The date of planting
GerminationDate	The data of seed germination
FlowerDate	The date of seed flowering
NumPlants	The number of plant grown
HarvestDate	The date of seed harvest
WeightofSeedBag	Weight of seed collected in each bag
Phenotype	Mutant phenotype description
OtherComment	Additional information input
ScreenedforNodulation	Mutant screened for nodulation phenotype

Table 4.3 “MutantM2” table

Field Name	***Description***
MutantM1ID	ID of mutant M1
MutantM2ID	ID of mutant M2. This field is primary key
NoDNASampling	The number of tissue for DNA isolation. This field is foreign key.
DateofSeedTreatment	The date of seed treatment germination
DateofRemovalfromFridge/FungicideTreatment	The date of removing the fungicide reagent
Type/AmountofFungicideUsed(mL/Lwater)	Information of fungicide used such as concentration
DateofTransplantingtoPlate	The date of transplanting treated seeds on to germinating plate
DateofSeedlingInoculation	The date of seedling inoculated
DateofTransplantingtoPot	The date of transplanting seedlings to pot
NumPlants	The number of plant grown
Phenotype	Mutant phenotype description
OtherComments	Additional information input
Group	Batch number
ImageID	ID of mutant image
Image	Images of mutant

Second Normal Form Second Normal Form (2NF) requires that all data elements in a table are functionally dependent on all of the table's primary keys. If data elements only depend on part of a primary key, then they are parsed to separate tables. If the table has a single field as the primary key, it is automatically in 2NF. A table is in 2NF if and only if 1) it is in 1NF and 2) each non-primary key attribute is irreducibly dependent on the primary key. In "MutantM2" table, *ID*, *NoDNASampling*, *Location* can become a table called "MutantM2Track" with a primary key of *NoDNASampling*, as shown in Table 4.4.

Table 4.4 "MutantM2Track" table

Field Name	**Description**
ID	AutoNumber
NoDNASampling	The number of tissue for DNA isolation. This field is primary key
Location	Mutant grid location

Third Normal Form The third normal form (3NF) is used to check whether all the non-key attributes of a relation depend only on the candidate keys of the relation. This means that all non-key attributes are mutually independent or, in other words, that a non-key attribute cannot be transitively dependent on another non-key attribute. *ImageID* and *Image* attributes are less dependent upon the *MutantM2ID* then they are on the image attribute. *ImageID*, *NoDNASampling*, *Seedm2ID* and *Image* can become a new table called image. Table 4.5 and 4.6 show the Table 4.3 in 3NF.

Table 4.5 “MutantM2” table

Field Name	***Description***
MutantM1ID	ID of mutant M1
MutantM2ID	ID of mutant M2. This field is primary key
NoDNASampling	The number of tissue for DNA isolation. This field is foreign key.
DateofSeedTreatment	The date of seed treatment germination
DateofRemovalfromFridge/FungicideTreatment	The date of removing the fungicide reagent
Type/AmountofFungicideUsed(mL/Lwater)	Information of fungicide used such as concentration
DateofTransplantingtoPlate	The date of transplanting treated seeds on to germinating plate
DateofSeedlingInoculation	The date of seedling inoculated
DateofTransplantingtoPot	The date of transplanting seedlings to pot
NumPlants	The number of plant grown
Phenotype	Mutant phenotype description
OtherComments	Additional information input
Group	Batch number

Table 4.6 “Image” table

Field Name	***Description***
ImageID	A unique number for image. This field is primary key
NoDNASampling	The number of tissue for DNA isolation
Seedm1ID	ID of mutant M2
Seedm2ID	ID of mutant M2
Image	Images of mutant

4.2.2 Design of Entity-Relationship Modeling

Databases are used to store structured data. The structure of the data, together with other constraints, can be designed using a variety of techniques, one of which is called entity-relationship (E-R) modeling. Figure 4.2 shows an E-R Diagram of the

Management System for *M. truncatula* Mutant Resources. Its components are: 1) rectangles representing entity sets, 2) ellipses representing attributes, 3) diamonds representing relationship sets, and 4) lines linking attributes to entity sets and entity sets to relationship sets.

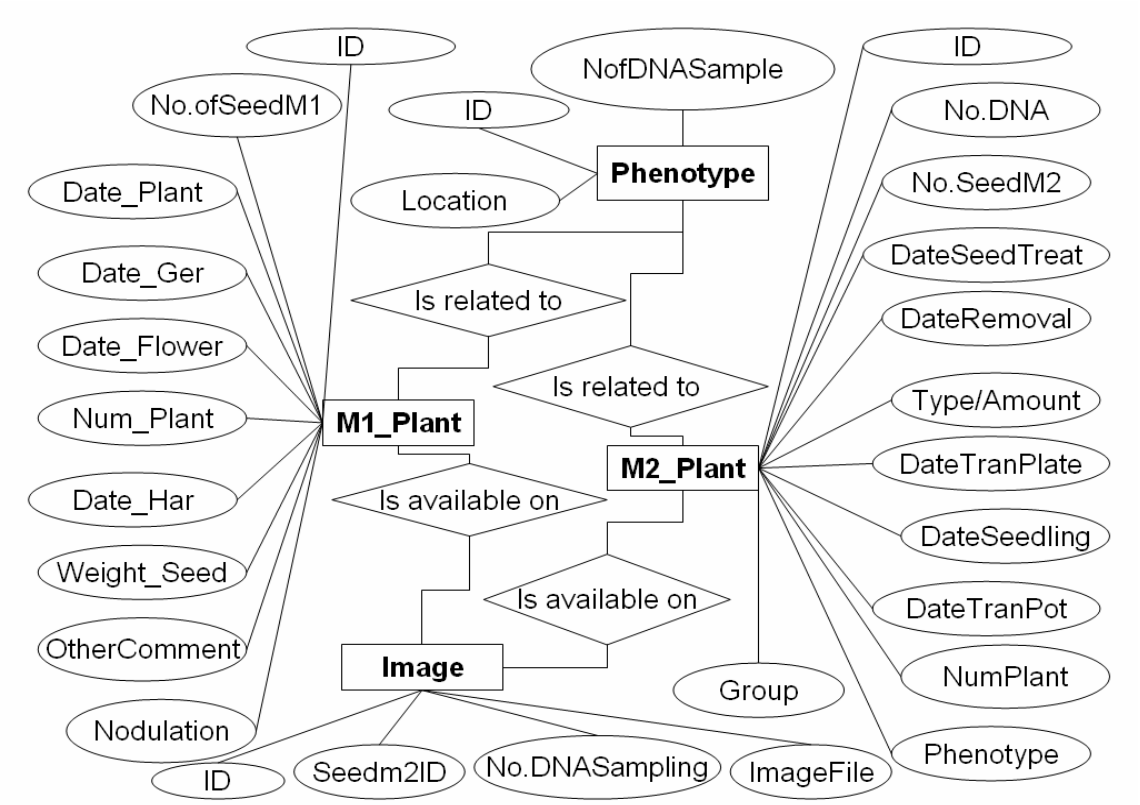


Figure 4.2 E-R Diagram of the Management System for *M. truncatula* Mutant Resources

4.3 Implementation of the Management System for *M. truncatula* Mutant Resources

4.3.1 Database Security and Access

To control what an individual user or group of users can do with a given database, the database has to be secured. It is necessary to consider who will use the database and what types of activities the users should be allowed to perform with the database. These activities might include viewing, modifying or deleting database objects or information.

To control the access, an input interface of user ID and password was generated for the *M. truncatula* mutant management system (Figure 4.3). User is also able to change password through the interface as shown in Figure 4.4 after logging on. Currently, the database is used by the research staff on the same project. In the near future, other researchers will be allowed to assess the database. However, the administrator of the database may authorize limited rights such as viewing only to the user outside.

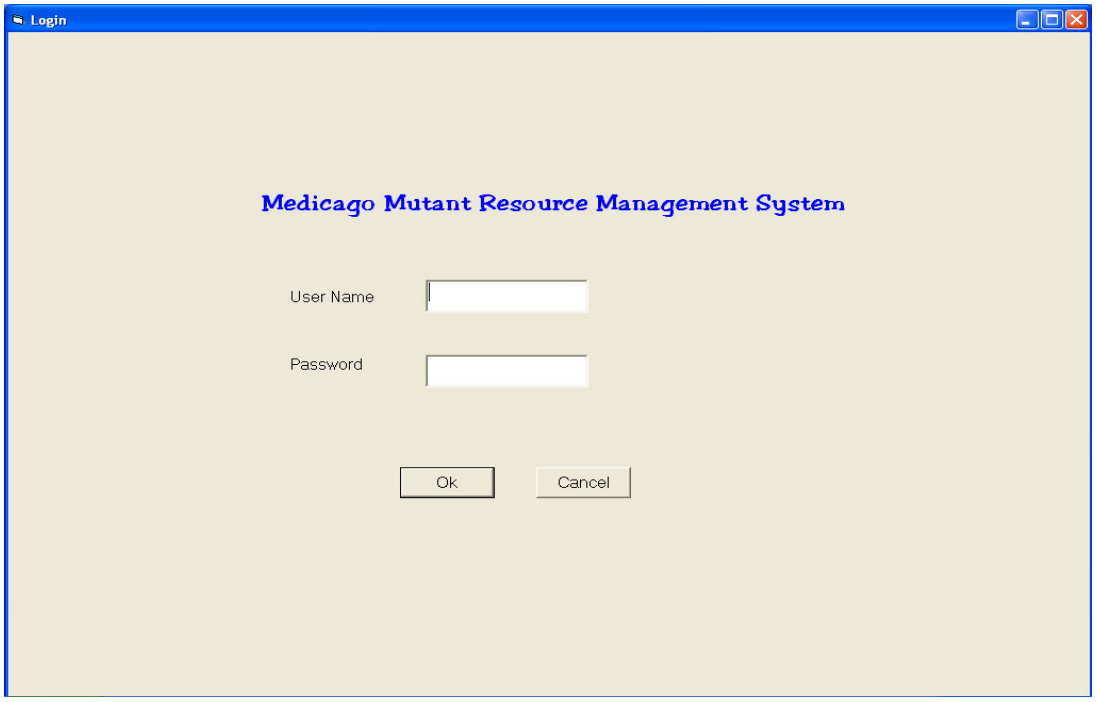


Figure 4.3 Login Interface

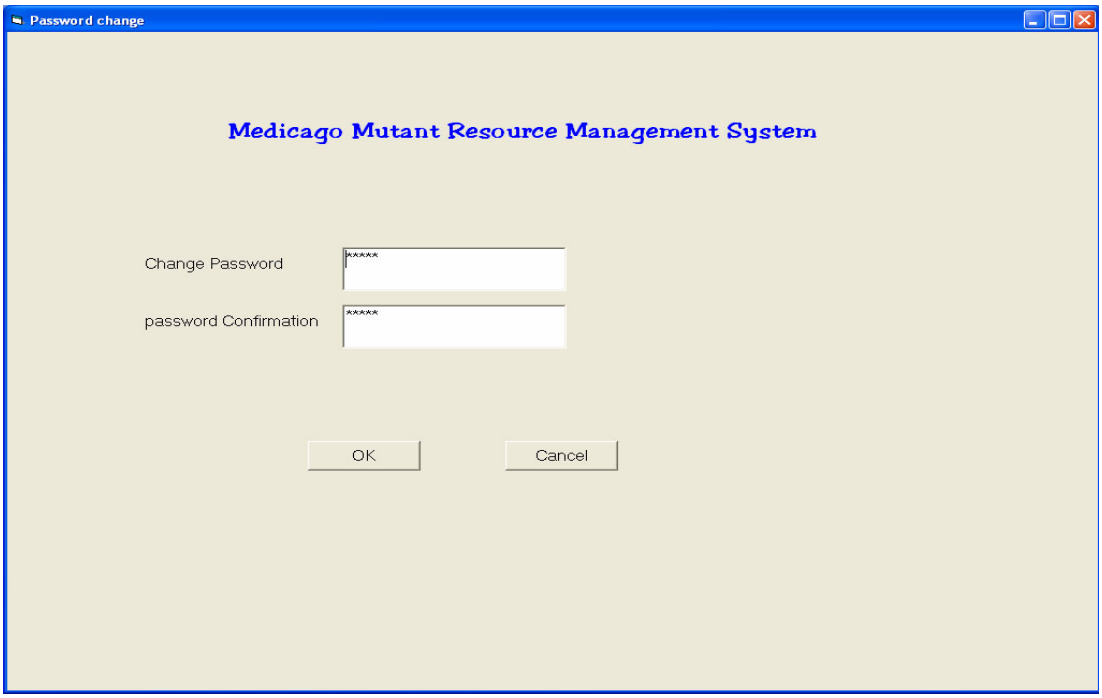


Figure 4.4 Change Password Interface

4.3.2 Data Input and Management

According to users' requirement, the component of "Data Input and Management" achieves many functions such as day-by-day data input, sorting, bookmark, filter and so on. The runtime interface of the component is shown in the Figure 4.5

Data manipulation The component allows users to conduct data manipulation including adding, updating, deleting, canceling, editing and refreshing. Other functions of the component includes navigating (move first, move previous, move next, and move last), finding (first and next), filtering, sorting and as well as adjusting of datagrid width based on the longest field in underlying source of data.

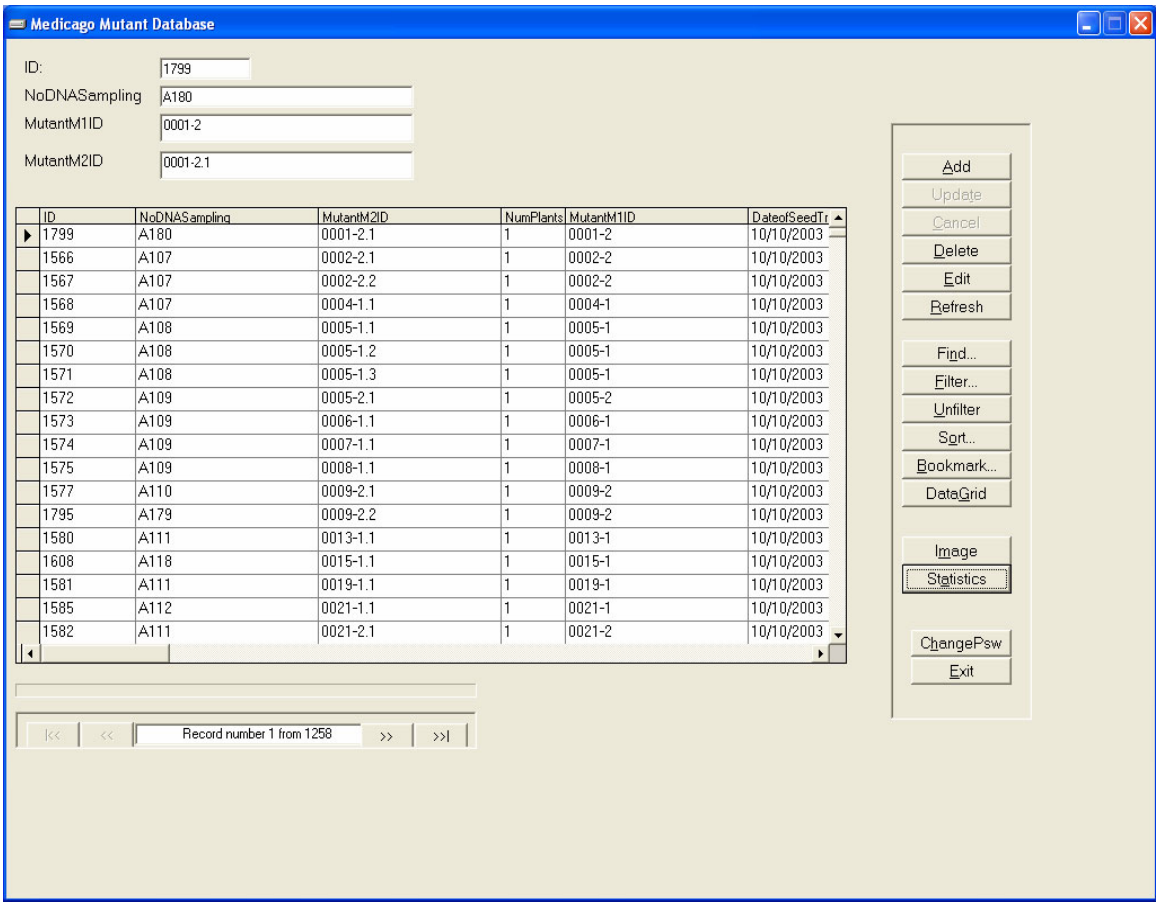


Fig 4.5 Interface of “Data Input and Management”

- ***“Add”*** Button: Click on the button to display the interface of data adding for users.
- ***“Update”*** button: If the user is either adding or editing a record, the ***“Update”*** button is enabled and can be pressed.
- “***Cancel***” button: Click on the button to cancel last operation.
- “***Refresh***” button: Click on the button to back to the very first record.
- “***Delete***” button: Click on the button to display “delete command” working window for user to delete data that will not be use anymore with a pop-up window to remind users to make sure to delete the record.

- "***Edit***" button: Click on the button to displays "edit command" working window for user to edit data in any column field and highlight the edited record for user.
- "***DataGrid***" button: Click on the button to display "DataGrid command" working window for users, which allow to adjusting datagrid width based on the longest field in underlying source.

Bookmark Pop-up Window Bookmark is a way to mark a record in a recordset so that users can go back to the record later quickly without remembering the position of that record. The users of the database can implement the function as follows (Figure 4. 6):

- Select the record to be bookmarked by clicking it in DataGrid or through Navigation button in *M. truncatula* mutant interface, then input the bookmark name in the textbox above, and then press "Enter" key or click "***Add***" button to add this name to the list box below.
- Click bookmark name in the listbox, then click "***Jump***" button; alternatively, double-click the bookmark name in the listbox, if user wants to go back to record bookmarked.
- Click bookmark name in the listbox, then click "***Delete"*** button, if user wants to delete the bookmark name,
- Click bookmark name in the listbox, then click "***Help***" button, if user wants to get the direction about using "bookmark" function.

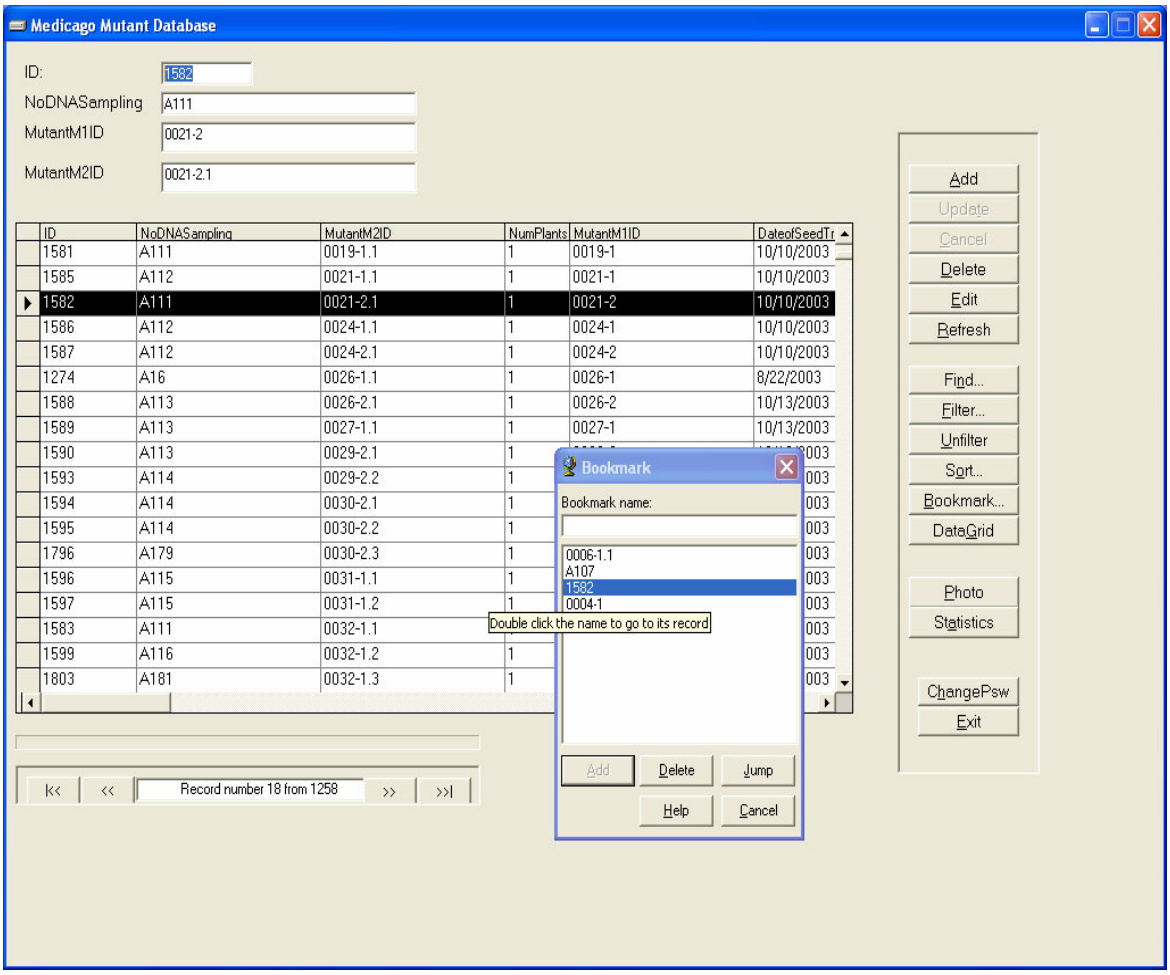


Figure 4.6 Bookmark Pop-up Window.

Filter Pop-up Window "Filter" function is used to specify which records in the database will be included in the specified project. The user of the database can implement the function as follows (Figure 4.7):

- Select the field name from the drop down menu.
- Enter the value of the field to filter.

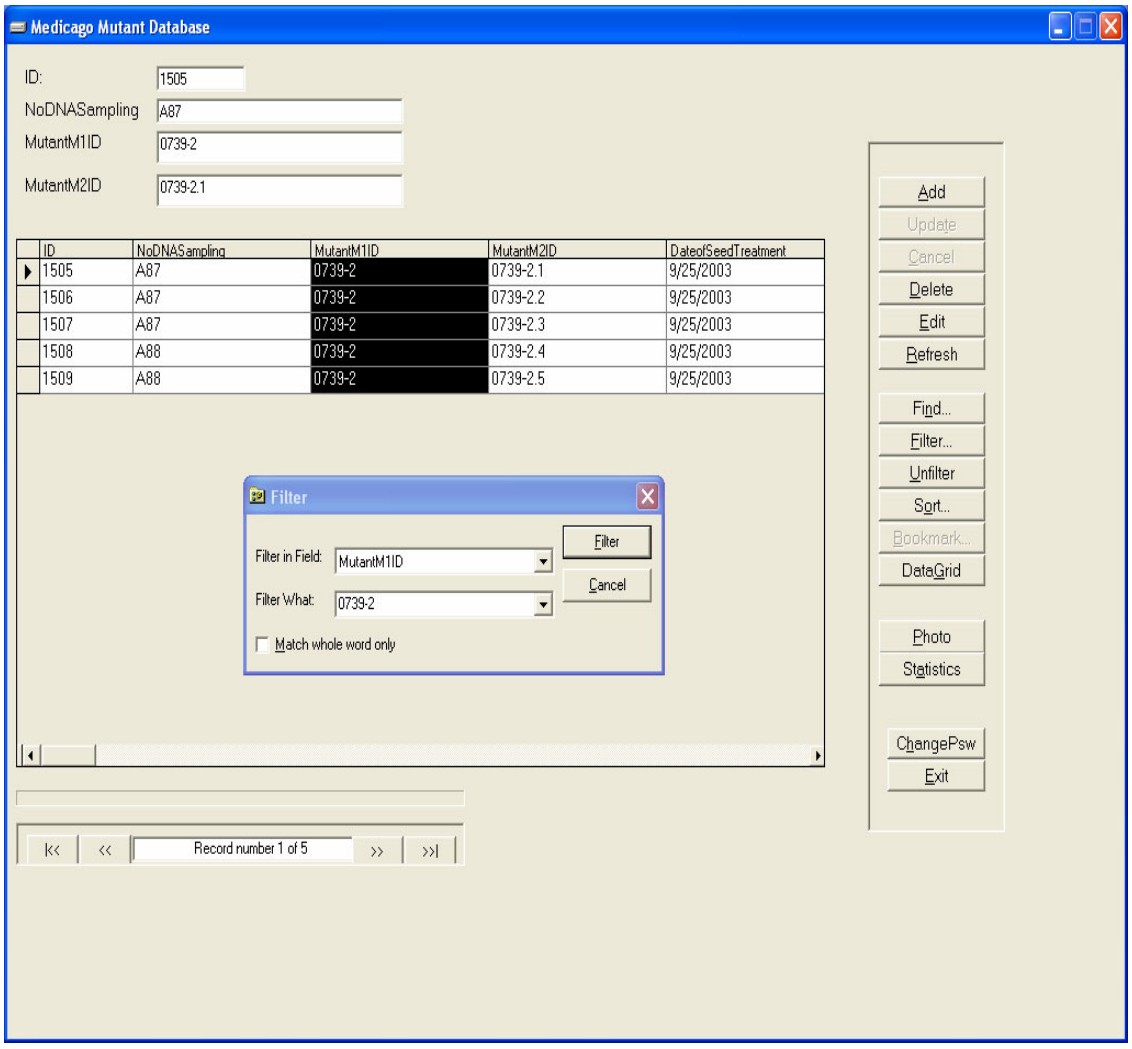


Figure 4.7 Filter with Drop-down menu Window

Find Pop-up Window "Find" function was designed quickly to locate any record in the database. The user of the database can implement the function as follows (Figure 4.8):

- Select the field name from the drop down menu.
- Enter the value of the field to find.

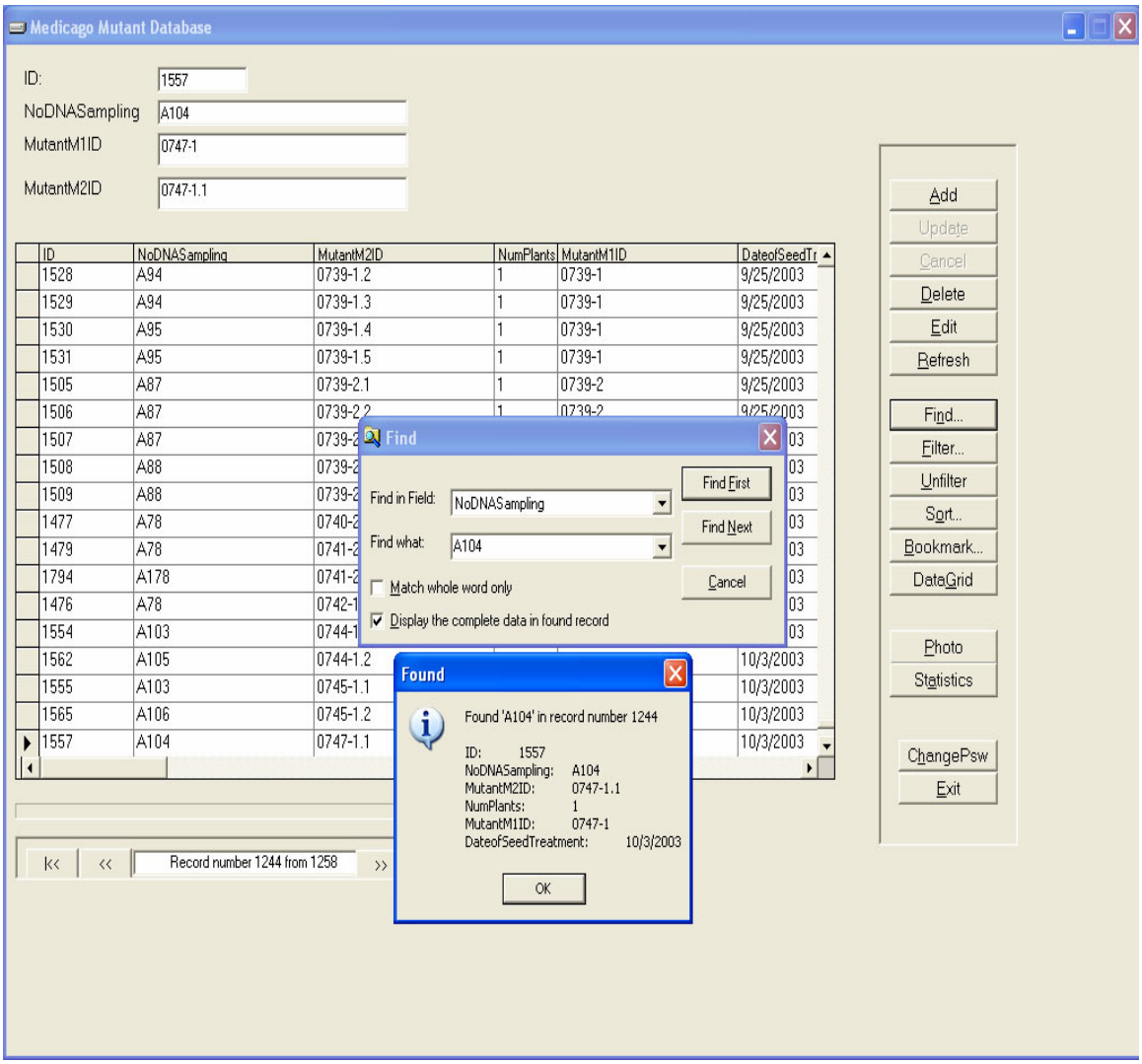


Figure 4.8 Find with Drop-down menu Window

Sort Pop-up Window The user of the database can sort a selected recordset as follows (Figure 4.9):

- Choose sort in field.
- Choose sort type. Click “Ascending” or “Descending” to sort.
- Click sort button on the sort for *M. truncatula*

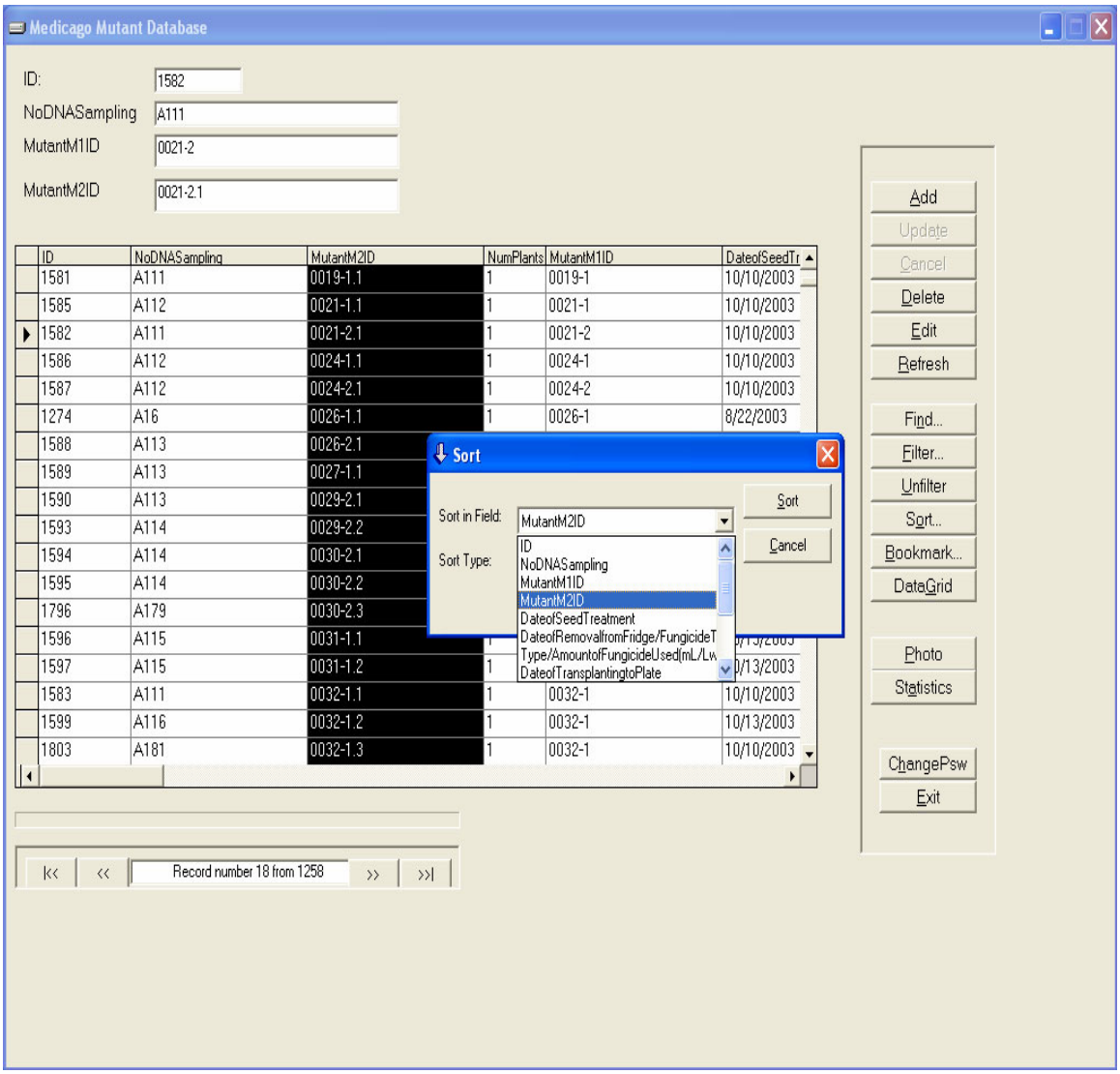


Figure 4.9 Sort with Drop-down Menu Window

4.3.3 Statistics and Mutant Image Review

Statistics: The users at the Noble Foundation desire to monitor the progress of the *M. truncatula* mutagenesis project through the database. Therefore, the "reporting statistics" function was designed to meet the need. The users of the database are able to generate charts for reporting statistics through the interface as shown in Figure 4.10 and 11. They can prepare graphic presentations that show the number of *M. truncatula* mutants generated monthly. Mutant populations generated in multiple years can be presented in

parallel in the same chart. The users can generate three types of plots of which Figure 4.10 and 11 show two. Graphic files prepared can be saved in different file formats.

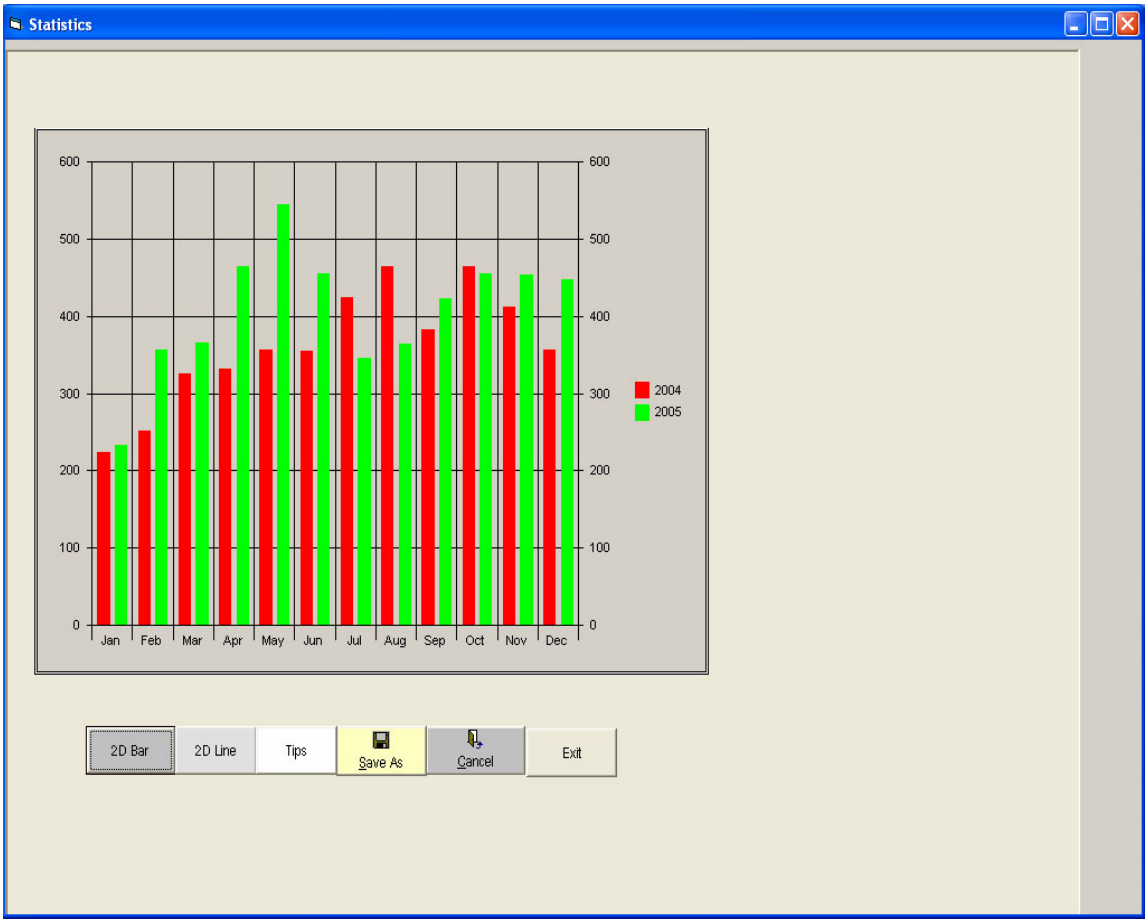


Figure 4.10 Reporting chart in two-dimensional bar

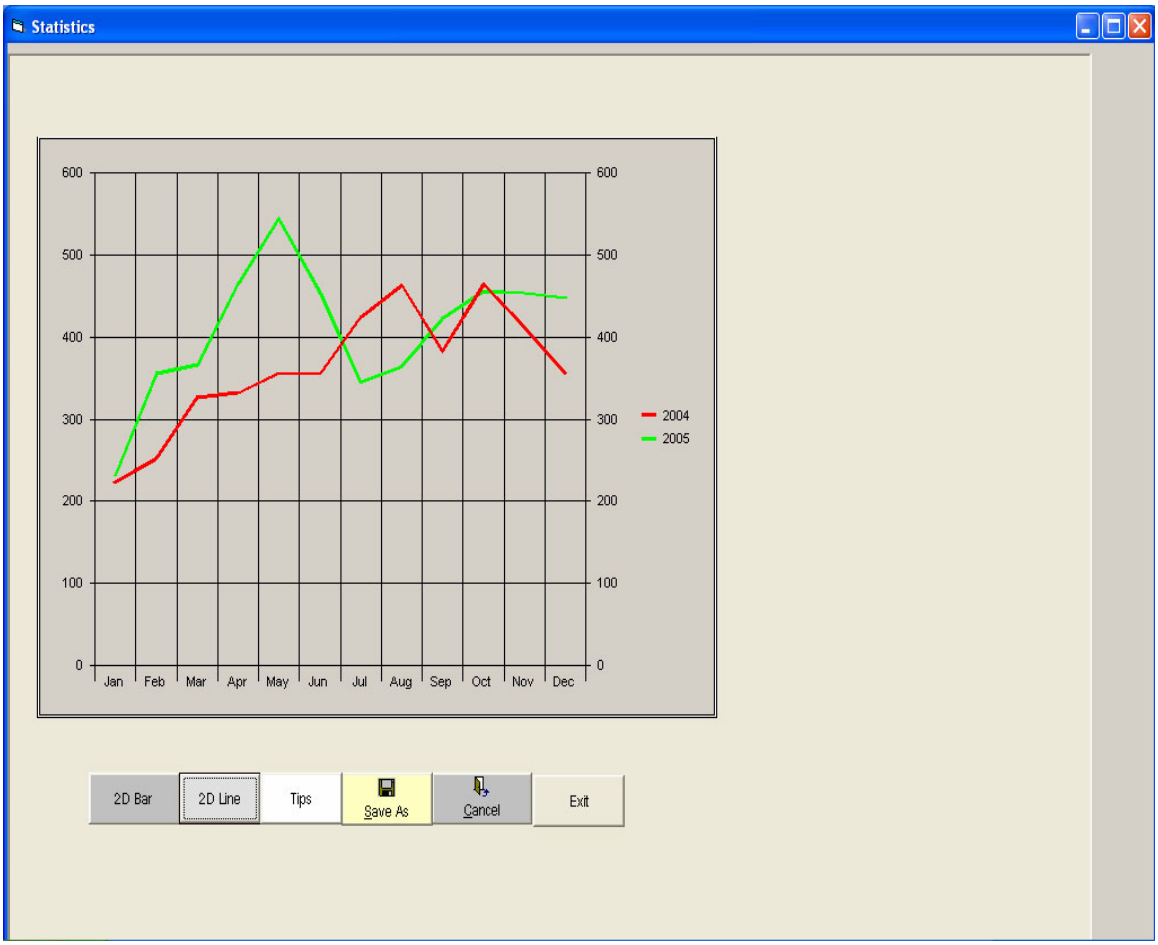


Figure 4.11 Reporting chart in two-dimensional line

Mutant Image Review: Morphological alternations are visible phenotypes that indicate the mutations carried by individual plant. The phenotypes of M2 plants become stable. The users of Noble Foundation requested to store the images of *Medicag* M2 plants in the database. Figure 10.12 shows the interface of the mutant image input and review of the database. Users are able to input, browse and compare the phenotypes of mutant individuals. This will facilitate to select mutant plants showing interesting phenotypes for further biological characterizations (forward genetics).

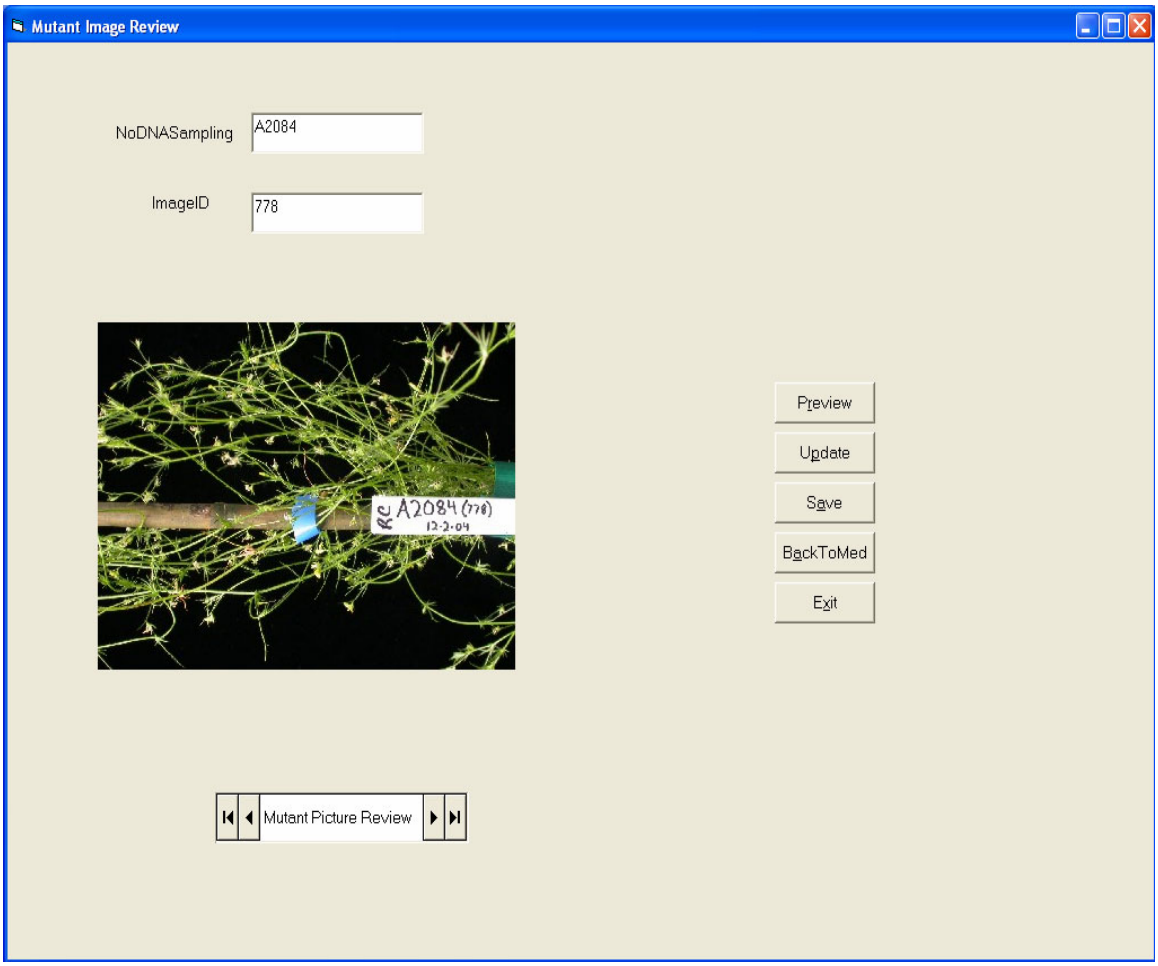


Figure 4.12 Mutant Picture Review

CHAPTER V

SUMMARY AND FUTURE WORK

5.1 Summary

This thesis describes the design and implementation of a computing tool and a relational database for the *M. truncatula* mutagenesis project conducted by the research fellows in Dr. Rujin Chen's laboratory at the Noble Foundation. The research group plans to generate approximately 100,000 mutant lines in order to interrupt most genes in the model legume, *M. truncatula*. To meet the needs of computer science embodied in the biological project, various computer technologies such as VB, VB.NET programming, relational database, SQL and Microsoft Access were used. The major achievements described in this thesis are as follows:

- The computing tool, *M. truncatula* Computing Tool, has been developed to generate error-free tissue sample lists for 3-dimentional pooling prior to DNA isolation. The tool greatly facilitates the routine operation of tissue pooling. Furthermore, *M. truncatula* Computing Tool facilitates the utilization of pooled mutant populations. It provides lookup functions that allow the scientists in the Noble Foundation to localize the grid address of DNA sample according to tissue sample number, and to identify the DNA sample number according to biological screening results.

- *M. truncatula* Mutant Management System described in the thesis, provides a powerful laboratory database to manage all resources and information generated within the long-term project. It assists in the management of the *M. truncatula* mutant populations and their progenies, and to record, then store and exploit all data generated within the project. The statistics function of the system allows the users to monitor the day-by-day progress of the *M. truncatula* mutagenesis project, which is the basis of next-step plan. The database also provides the review of mutant images to facilitate the selection of individual mutant for further biological analysis (Forward genetics).

5.2 Future Work

The objective of the *M. truncatula* mutagenesis project is to generate mutant populations for the research community of *M. truncatula*. To share research resources, one of emerging needs in the near future is to develop an external database, which will allow the authorized scientist outside to access the *M. truncatula* mutant resources generated at the Noble Foundation via the Internet. *M. truncatula* Mutant Management System developed in the present work will be the basis for the development of external database. On the other hand, the utilizations of the *M. truncatula* mutant resources will be steadily grown with the increase of mutant populations. To further facilitate biological research, one of other recognized needs is to provide various bioinformatics tools connected with *M. truncatula* Mutant Management System. Those bioinformatics tools could be already well developed such as sequence blast searching. New bioinformatics tools will also be developed according to specific requirements of the ongoing project.

REFERENCES

1. Vance, C.P., *Symbiotic nitrogen fixation and phosphorus acquisition. Plant nutrition in a world of declining renewable resources.* Plant Physiology, 2001. **127**(2): p. 390-7.
2. Young, N.D., et al., *Sequencing the genespaces of Medicago truncatula and Lotus japonicus.* Plant Physiology, 2005. **137**(4): p. 1174-81.
3. Sham, P., et al., *DNA Pooling: a tool for large-scale association studies.* Nature reviews. Genetics, 2002. **3**(11): p. 862-71.
4. Prague, C. N. , M. Irwin, R. and J. Reardon, *Access 2003 Bible.* 2003: Wiley.
5. Santiago, C.D., *Component Object Model (COM), DCOM, and Related Capabilities*. 2001.
6. Williams, M. and M. Williams, *Visual C^{++} 6 Unleashed*. 2000: Sams.
7. Appleman, D., *Developing COM/ActiveX Components with Visual Basic 6*. 2000: Sams.
8. Williams, C., *Professional Visual Basic 6 Databases with VB, ADO, SQL Server and MTS.* 1999, Birmingham: Wrox Press Ltd.
9. Johnson, G., *Programming Microsoft ADO.NET 2.0 Applications: Advanced Topics*. 2005: Microsoft Press.
10. Rob, P. and C. Coronel, *Database Systems: Design, Implementation and Management*. Sixth Edition ed. 2004: Course Technology.

11. Forte, S., T. Howe, and K. Wall, *Access Database Design and Normalization*. 2002: Sams.

12. Lans, R.F.., *Introduction to SQL*. 1988: Wokingham, England ; Reading, Mass. : Addison-Wesley Pub. Co.

13. Din, A.I., *Structured Query Language (SQL): a Practical Introduction*. 1994: NCC Blackwell.

14. McFadyen, R. and V. Kanabar, *Introduction to Structured Query Language*. 1991: William C. Brown Pub.

15. *How To Create the Web.config File for an ASP.NET Application http://support.microsoft.com/kb/815179*.

16. McComb, G., *Using INI Files to Store Data*. 1997.

17. Barwell, F., *Professional VB.NET*. 2 ed. 2002: Wiley, John & Sons, Incorporated.

18. Grimes, R., *Developing Applications with Visual Studio.NET*. 1st edition. 2002: Addison-Wesley Professional.

19. Appleman, D. and D. Appleman, *Moving to VB.NET: Strategies, Concepts, and Code*. 2001: Apress.

20. MacDonald, M., *The Book of VB.NET: .NET Insight for VB Developers*. 1 edition ed. 2002: No Starch Press.

21. Viescas, J.L., *Building Microsoft Access Applications*. Pap/Cdr edition ed. 2005: Microsoft Press.

APPENDIX

A. Acronym and Abbreviation

ADO	Active Data Objects
ANSI	American National Standards Institute
API	Application Programming Interface
CGI	Common Gateway Interface
COM	Component Object Model
CONFIG File	Configuration File
DAO	Data Access Objects
DFD	Data Flow Diagram
DNA	Deoxyribonucleic acid
E-R Diagram	Entity Relationship Diagram.
GA	Genetic Algorithm
IDL	Interface Description Language
INI file	Initialization file
MS Access	Microsoft Access
MSDASQL	The Microsoft OLE-DB provider for ODBC
OCX	OLE Control Extension
ODBC	Open Database Connectivity
OLE	Object Linking and Embedding

OLE DB	Object Linking and Embedding for Databases
OOP	Object-oriented Programming
PCR	Polymerase chain reaction
RDBMS	Relational Database Management System
RDO	Remote Data Objects
SQL	Structured Query Language
UDA	Universal Data Access
VBXs	Visual Basic Controls

B. Glossary

Application Program Interface (API) An abbreviation of application program interface, a set of routines, protocols, and tools for building software applications.

Basic Local Alignment Search Tool (BLAST) finds regions of local similarity between sequences. The program compares nucleotide or protein sequences to sequence databases and calculates the statistical significance of matches. BLAST can be used to infer functional and evolutionary relationships between sequences as well as help identify members of gene families.

Component Object Model (COM) A model for binary code developed by Microsoft that enables programmers to develop objects that can be accessed by any COM-compliant application.

Configuration (config) File In computing, configuration files, or config files, are used to configure the initial settings for some computer programs. They are used for user applications, server processes and operating system settings. The files are often written in ASCII (rarely UTF-8) and line-oriented, with lines terminated by a newline or carriage return/line feed pair, depending on the operating system. They may be considered a simple database. Some files are created and modified using an ASCII editor. Others are created and modified as a side-effect of changing settings in a graphical user interface (GUI) program. Some computer programs only read the configuration files at startup. Others periodically check the configuration files for changes. Some can be told to re-read the configuration files and apply the changes to the current process, or indeed to read arbitrary files as a configuration file.

Data flow diagram (**DFD**) is a graphical representation of the "flow" of data through an information system. A data flow diagram can also be used for the visualization of data processing (structured design). It is common practise for a designer to draw a context-level DFD first which shows the interaction between the system and outside entities. This context-level DFD is then "exploded" to show more detail of the system being modelled.

Deoxyribonucleic acid (**DNA**) is a nucleic acid that contains the genetic instructions for the biological development of a cellular form of life or a virus. All known cellular life and some viruses have DNAs. DNA is a long polymer of nucleotides (a polynucleotide) that encodes the sequence of amino acid residues in proteins, using the genetic code: each amino acid is represented by three consecutive nucleotides (a triplet code).

Dynamic-link library (**DLL)** is a library of executable functions or data that can be used by a Windows application. Typically, a DLL provides one or more particular functions and a program accesses the functions by creating either a static or dynamic link to the DLL. A static link remains constant during program execution while a dynamic link is created by the program as needed. DLLs can also contain just data. DLL files usually end with the extension .dll, .exe, .drv, or .fon.

Entity Relationship Diagram (E-R Diagram) A model provides a high-level description of a conceptual data model. Data modeling provides a graphical notation for representing such data models in the form of *entity-relationship diagrams* (ERD).

Initialization file (**INI file)** INI file is a configuration file that contains configuration data for Microsoft Windows based applications.

Interface Description Language (IDL) An Interface Description Language (or alternately, Interface Definition Language), or IDL for short, is a computer language used

to describe a software component's interface. IDLs describe an interface in a language-neutral way, enabling communication between software components that do not share a language – for example, between components written in C and components written in Pascal.

M1 plants Plants grown from seeds mutated by fast-neutron mutagenesis.

M2 seeds Seeds produced by M1 plants.

M2 plants Plants grown from M2 seeds.

M3 seeds Seeds produced by M2 plants.

Object Link Embedded (OLE DB) It is a set of interfaces implemented using the Component Object Model (COM). OLE DB separates the data store from the application that needs access to it through a set of abstractions, such as connections, record sets and attributes.

Object-oriented Programming (OOP) in computer science, object-oriented programming is a computer programming paradigm. Many programming languages support object-oriented programming. Many programming frameworks, like the Java platform and the .NET Framework, are built on object-oriented principles. Object-oriented programming is often abbreviated as OOP.

Polymerase chain reaction (**PCR**) is a molecular biology technique, for enzymatically replicating DNA without using a living organism, such as *E. coli* or yeast. Like amplification using living organisms, the technique allows a small amount of the DNA molecule to be amplified exponentially. However, because it is an *in vitro* technique, it can be performed without restrictions on the form of DNA and it can be extensively modified to perform a wide array of genetic manipulations.

Relational Database Management System (RDBMS) A type of database management system (RDBMS) that stores data in the form of related tables. Relational databases are powerful because they require few assumptions about how data is related or how it will be extracted from the database.

VITA

LIN GE

Candidate for the Degree of

Master of Science

Thesis: A COMPUTING TOOL AND A RELATIONAL DATABASE FOR A MEDICAGO TRUNCATULA MUTAGENESIS PROJECT

Major Field: Computer Science

Education:

Received the Degree of Bachelor of Science at Suzhou Railway Teachers College, Suzhou, P. R. China in 1991.

Received the Certificate of Geographic Information Systems at Oklahoma State University, Stillwater, Oklahoma in 2003.

Completed the requirements for the Master of Science in Computer Science at Oklahoma State University, Stillwater, Oklahoma in December, 2007.