

Rochester Institute of Technology

**RIT Digital Institutional Repository**

---

Theses

---

5-18-2016

## **Development of a Data Management System for Grape Breeding Programs**

Connor Fortin  
chf3908@rit.edu

Follow this and additional works at: <https://repository.rit.edu/theses>

---

### **Recommended Citation**

Fortin, Connor, "Development of a Data Management System for Grape Breeding Programs" (2016). Thesis. Rochester Institute of Technology. Accessed from

This Thesis is brought to you for free and open access by the RIT Libraries. For more information, please contact [repository@rit.edu](mailto:repository@rit.edu).

# Development of a Data Management System for Grape Breeding Programs

by

Connor Fortin

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of  
Masters of Science in Bioinformatics

College of Science

Thomas H. Gosnell School of Life Sciences

Rochester Institute of Technology

Rochester, NY

May 18, 2016

Committee Members:

Michael Osier

Lance Cadle-Davidson

Gregory Babbitt

Rajendra Raj

## **Abstract**

Grape production is a significant industry in the U.S. and abroad, and is negatively impacted by diseases such as powdery mildew and downy mildew. Selecting vines that are disease resistant and have good fruit quality requires organized record-keeping and the ability to find useful trends in the data. In the Cornell Grape Breeding Program, and many others, data is stored in outdated systems, excel spreadsheets, and paper books; the need for a new data management system specific to grapes is prevalent. The Breeding Management System (BMS) is a crop breeding data management system that is well suited for the organization and storage of breeding data. The use of the BMS for grape breeding was explored, and several extensions to the functionality of the BMS were developed. The most significant extension is the design of a graphical search tool, allowing users to search for vines based on the name of the vine, parents of the vine, experiments in which the vine appears, and observed values in experiments. An ontology was defined to organize the experimental data collected, and a pipeline was created to allow researchers to record data in an audio format, which will be converted into an excel spreadsheet. The development of the system is successful because it centralizes data storage into a common format and system, allows for the extraction of relevant experimental and vineyard data, and allows for the identification of well performing vines over single and multiple traits.



## Table of Contents

<u>Section</u>	<u>Page Number</u>
Introduction	4
<i>Economic Impact of Grape Production</i>	4
<i>Powdery Mildew and its Impact on Grape Production</i>	4
<i>Previous Powdery Mildew Resistance Research Efforts</i>	5
<i>Currently Available Plant Breeding Management Tools</i>	5
<i>Breeding Management System at a High Level</i>	8
<i>Breeding Management System Benefits and Usage</i>	10
Methods	21
<i>Uploading Data to the BMS</i>	21
<i>Vine search tool</i>	24
<i>Search Types in Detail</i>	26
<i>Trait Filter Processing</i>	29
<i>Information Management Window</i>	33
<i>Options Menu</i>	34
<i>Vine Name Synonyms</i>	35

<i>Report Formatting</i>	37
<i>Voice-to-spreadsheet</i>	40
<i>Scripts for Fieldbook Printing</i>	44
Results	45
<i>BMS Data Upload</i>	45
<i>Vine Search Tool</i>	47
<i>Report Formats</i>	57
<i>Voice-to-spreadsheet</i>	60
<i>Scripts for Fieldbook Printing</i>	60
Discussion	63
<i>Graphical Search Tool</i>	63
<i>Voice-to-spreadsheet</i>	66
<i>Ontology</i>	67
<i>Data Entry</i>	69
Citations	71
Appendices	75





## **Introduction**

### *Economic Impact of Grape Production*

According to a study by the MKF Research Institute, table grapes, wine, and other grape products contributed 162 billion dollars to the U.S. economy in 2007. This number was reached by calculating the total revenue from the sale of wine and other grape products and adding this to the total wages made by those in the grape and wine industry in the same year (MKF Research LLC). Worldwide, 75,866 square kilometers are used for grape production. Most of this land is dedicated to wine production, some is used for the production of table grapes, and a small percentage is used to produce raisins and juice. The area used to grow grapes increases worldwide by about 2% every year (Tonietto, Jorge, and Alain Carbonneau, 2004). The UN Food & Agriculture Organization reports that the U.S. passed Italy in 2012 to become the second highest producer of grapes in the world. China currently leads the world in grape production as of 2012 with 9,600,000 metric tons of grapes produced in that year (Production of Grape by countries, 2014).

### *Powdery Mildew and its Impact on Grape Production*

*Erysiphe necator* is the fungus that causes powdery mildew on all green parts of a grape vine. Vines infected with the fungus do not produce marketable fruit. Thus, grape breeders and vineyard managers have an interest in preventing the fungus from infecting vines. Currently, the most effective way to reduce powdery mildew growth in vineyards is the spraying of fungicides including sulfur and other chemicals (Bowen, 2014). About 30 million pounds of sulfur are used by grape growers every year in the United States (USDA-NASS, 2006). Currently, most grape

vines that grow market quality fruit have little to no host resistance to the fungus; *Vitis vinifera*, the most common species for grape breeding and production because of its high fruit quality, is very susceptible to fungus and other pests (Alleweldt, G., and J.v. Possingham, 1988).

### *Previous Powdery Mildew Resistance Research Efforts*

Breeding of disease resistance into a grape cultivar is a time consuming process for several reasons. Because powdery mildew resistance exists primarily in wild *Vitis spp.* and fruit quality is best in domesticated *V. vinifera*, interspecific cross-hybridization is needed to combine resistance and quality. Breaking the linkage drag between resistance and negative fruit quality may require several generations. Grapes also have a long generation time of up to several years between when a grapevine seed is planted and when it will produce seeds for further propagation. Furthermore, the evaluation of some traits and resistances may take several years (Dalbo, Weeden, Wilcox, Reisch 2001, Fisher et al. 2004) and require specific expertise and equipment. For these reasons it is helpful to develop genetic markers associated with disease resistance and other desirable traits, as well as negative traits, to select elite offspring for further evaluation as soon as enough plant material exists for genotyping.

Over the past few years, research has been done on several fronts using modern techniques to identify sources of resistance and susceptibility in grapes to the fungus. Fung et al. showed that vines susceptible to powdery mildew undergo defense-related changes in the transcriptome when infected; resistant vines do not undergo these changes (Fung et al., 2008). Several other groups have used genetic mapping to identify trait loci involved with resistance to various fungi including powdery mildew (e.g., Welter et al., 2007; Murat et al., 2007). The identification of the *REN4* locus for powdery mildew resistance was published by members of the *VitisGen* project in

2011 (Mahanil et al., 2011). Vines carrying this resistance locus support no development of powdery mildew regardless of race or tissue type (Ramming et al., 2011). Researchers in the *VitisGen* project are using high throughput genotyping by-sequencing techniques to identify new genetic markers associated with powdery mildew resistance and other traits. In addition they are using simple sequence repeat markers linked with known resistance loci to select disease resistant parents and progeny. They hope to use these techniques to develop cultivars that produce market quality fruit, and have a high natural resistance to powdery mildew (Barbra, 2014).

The identification of multiple resistance sources with different mechanisms, and breeding several of these loci into a cultivar is important for the long term durability of the resistance genes. Powdery mildew can adapt within a few years to the resistance of a single locus, creating a virulent strain of the fungus, and reducing the efficacy of the individual resistance gene (Gohre and Robaztek, 2008).

### *Currently Available Plant Breeding Management Tools*

Currently, the methods for recording data pertinent to grape breeding, especially phenotypic data, are unique to each breeder. Breeders may submit excel spreadsheets formatted differently, have different names for certain traits, use different scales for measuring traits, submit text documents, and in some cases submit hand written phenotypic data to the *VitisGen* project. Much of the phenotypic data remains in the format it was originally submitted in, making organization, comparison, and analysis of the data very difficult. In order to effectively utilize and curate the data, the data must be converted to a common format, and stored in a system that has the following important properties: easy to enter new germplasm and fieldmaps but maintain

historical records of germplasm and plantings, easy to update existing records, easy to ensure integrity of data and restrict access, easy to generate reports of existing vines, fieldmaps, and reports based on trait and genotype data, and the ability to export data in a format that facilitates statistical or genetic analysis (The IBP, 2015).

The most popular commercially available breeding software available today is Agrobase. Listed on its website as supported crops are sorghum, sunflowers, groundnuts, potatoes, canola, cotton, forages, vegetables, flowers, and oil palm, though it mentions other breeders use the software for other crops. Agrobase organizes data into plant research trials, and statistical analysis can be performed on each trial, and on multiple trials together. There are currently many optional analysis modules available to suit a large variety of needs. Agrobase is built using a MySQL architecture, and users interact with it through a graphical user interface. Data can be exported into Microsoft Excel spreadsheets, ASCII plaintext, and a variety of other formats.

Unfortunately, this system does not support grape breeding, and thus is not suitable for our use (Mullitz, 1990).

Project Unity, a breeding management system developed by Phenome Networks, provides another potential platform to store, retrieve, and analyze plant breeding data. This software solution is unique in that it encourages users to release their data to all other users of the software for browsing and analysis, though it expects typical users to share data only after they publish their findings. Unfortunately, Phenome does not and cannot guarantee the integrity of data from other users, which may limit its usefulness unless the data comes from known high quality sources. The software, including each breeder's data, is hosted online, thus no software needs to be installed; all interaction with Project Unity is through a web browser. Data is stored in Amazon's cloud environment. An internet connection is required to access data and analysis

tools, which in most cases will not be an issue. Unfortunately, many researchers and companies are unwilling to store their data on servers that they do not control, even though Phenome pledges absolute data privacy and claims no ownership of your data (Unity Project, 2004).

E-Brida may be one of the more basic breeding management tools available online. Unlike most other published breeding management tools, it includes no data analysis support. It specializes in pedigree and ancestor data storage, viewing and retrieval. E-Brida does not appear to offer support for the storage of genetic data. This limits E-Brida's usefulness for many breeding programs who want to track genetic data, and do not want to spread the data out over multiple pieces of software, or require breeders to learn to use multiple breeding management tools (E-Brida).

### *Breeding Management System at a High Level*

Participants on the *VitisGen* project have elected to manage data using the Breeding Management System (BMS). The BMS is “a web-based solution for crop breeders, where registered users can access purpose-built tools to manage their plant breeding projects, obtain support and consulting services, find new knowledge, access training resources and discuss pertinent issues with their peers in various communities of practice.” This system, initially funded by the Bill and Melinda Gates Foundation, has buy-in from a diverse group of breeders who are committed to using the software, which is currently in development, for the future of their breeding programs. The system is built on a MySQL architecture and currently only supports local hosting. The BMS allows for thorough definition of trait ontologies, and when exporting trait data, a metadata tab is generated, populated with useful details of the trait

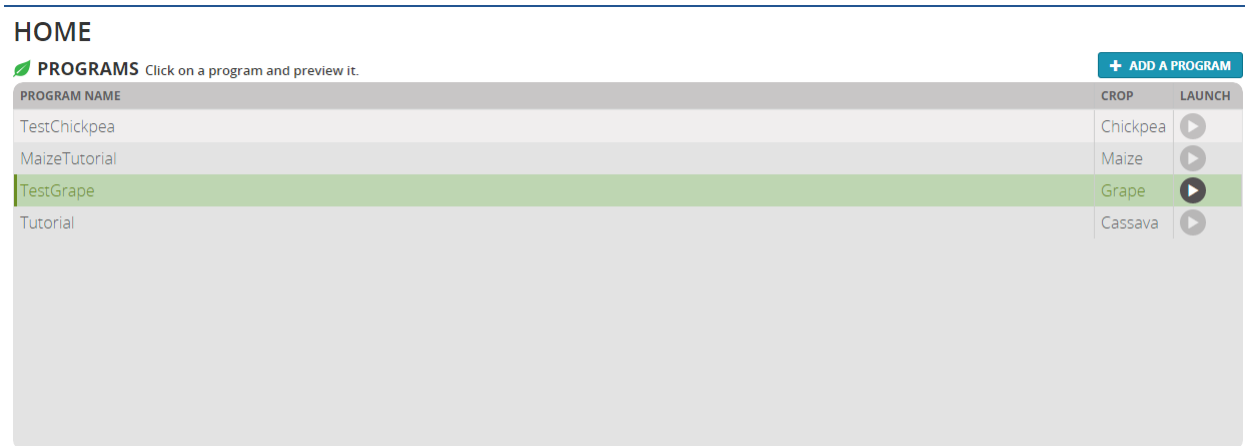
collection, units, description, and collection method. Built in data analysis tools allow for easy computation to be performed with data in the BMS.

The BMS comes with built-in support for a variety of crops including chickpeas, wheat, maize, and groundnuts. The built-in support includes pre-built variable lists and ontologies relevant to each crop, pre-built template files for importing germplasm lists and nurseries, and access to publically available germplasm records. Grape breeding does not have built-in support from the BMS. This means that variable lists and ontologies must be custom-built, and template files must be built manually by the user, or borrowed from other crops. Additional complexities may arise from grapevine being a woody perennial subject to different breeding practices than the built-in crops, which are all annuals.

Understanding the general structure of data in the BMS is important to understanding how it can be used in a breeding program. The BMS is installed locally on a computer; the BMS does not current support connection over a network. The BMS runs in a web browser, and accesses a MySQL database which serves as the model. To gain access to the BMS through the webpage, a user must create an account with a username and password. Anyone with access to the computer can create an account. Users with an account can create breeding programs in the BMS (Fig. 1). Breeding programs contain all of the variables, experimental data, vine information, and other data. The creator of the program can specify other users to have access to read and write data in the program. A user can have access to any number of programs in the BMS.

Figure 1: BMS Home Screen

Image of the Home Screen of the BMS. Users can select which program they would like to work with, provided they have the proper permissions.



All of the data uploaded to the BMS must be entered via excel spreadsheets. When uploading experimental data, a template spreadsheet can be exported from the BMS into which the experimental data can be pasted. This spreadsheet is then uploaded to the BMS which populates the MySQL database with data from the spreadsheet. When uploading information about vines, there are template files that come with the software, and can be downloaded from the BMS website.

### *Breeding Management System Benefits and Usage*

The BMS has several advantages over other data management systems. Updates, patches, and product support will be ongoing, which is not the type of support that would be available had an independent system been developed from scratch for the Cornell Grape Breeding Program.

Secondly, the detailed trait ontology and variable management system (Fig. 2) built into the

BMS will be useful for standardizing the data collection process across public grape breeding programs.

Figure 2: BMS Variable and Ontology Management Screen

Shown below is the BMS screen to create or edit ontology variables. Entered are the details of the BERRY\_pH variable. The ontology location is selected in the section labeled “Properties and Trait Classes”. The specific trait for the variable is denoted in the section labeled “Property”. The method and scale of the variable can be seen in the section labeled “Methods and Scales”. The “Role” dropdown list tells the BMS where to store the variable in the MySQL database, and the “Data Type” and “Valid Values” boxes allow specifications for the types of data expected.

#### MANAGE OR ADD VARIABLES

*\* indicates a mandatory field*

**VARIABLE DETAILS:**

Variable Name: \*  Role: \*

Description:  Data Type: \*

Valid Values:  Minimum  Maximum

**PROPERTIES AND TRAIT CLASSES:**

Filter for Trait Class: \*   Property: \*

Description:  Description:

Trait Class Tree View: 

- Agonomic
- Biochemical trait
- ▷ Biotic stress
- General
- Genetic
- Grain quality
- Lineage
- ▲ Morphological
- ▷ **Fruit Quality**
- Passport
- Physiological

 Crop Ontology ID:

**METHODS AND SCALES:**

Method: \*   Scale: \*

Description:  Description:

The trait ontology is part of a larger variable management system which allows breeders to specifically define the measurements they are recording for their breeding programs. Each variable has three primary components: a trait to be measured, a method for measuring the trait,



and a scale on which the measurement is taken. A variable is defined by a unique combination of these characteristics. For example, one could define a variable for measuring grape berry weight to be the weight of twenty dry berries on the gram scale. A separate variable could put the same trait and method on the ounce scale.

The concept of defining variables in this fashion addresses a deeper data collection problem that occurs in breeding programs. There is great overlap in the types of traits that breeders are interested in measuring, such as disease resistance, cold tolerance, and taste, but there are differences in the way each breeder measures and codifies the data they collect. Consider the following two data scales. Cornell codifies powdery mildew infection ratings on a five-point scale wherein 1 = no infection and 5 = full infection. On occasion, Cornell uses a four point scale for powdery mildew infection. Another breeding program uses a seven-point scale for the same trait where 1 = no and 7 = full infection. Additionally, the different breeders may consider different grape tissues in their infection rating; Cornell may only consider the leaves and berries, where another program may consider the leaves, berries, and rachis. There are examples of these differences over many other traits that are evaluated by the breeding programs. Combining data from many programs to gain new insights into disease resistance and other traits is a primary goal of the *VitisGen* project, but the combination of different methods of rating and different rating scales makes comparison of data difficult or inappropriate.

Selecting a trait from the trait ontology is part of the variable definition process in the BMS.

When creating a new variable, you may select an existing trait or create new trait. Creating a new trait involves describing what trait is being created, and where in the trait ontology it will exist.

This organizes variables so that they are easy to find and allows users to easily find similar traits that they may want to evaluate.

There are several other features of a variable that need to be defined as well. A general name and description of the variable is required. We have tried to incorporate the trait, method, and scale into the name of the variable. The format of data being stored in the variable, such as text, numbers, and dates must also be defined. Minimum and maximum values can be specified, or a set of exact values to choose from. New methods and scales that do not exist in the BMS can be created as needed.

There are three main data structures for storing data in the BMS. A list, nursery, and trial are used to store breeding program data. The most basic of the data structures is the list (Fig. 3), which is simply a list of vines. Lists are commonly uploaded from an excel spreadsheet, and can be used to store the vines that are in a vineyard, to store vines for which data was collected in a certain experiment, or to store any grouping of vines however concrete or abstract. Every nursery and trial must have an associated list of vines in the nursery or trial. This relationship is enforced and is the primary use of lists.

### Figure 3: Vine List Example

This image shows the single-dimensional nature of a vine list.

#	DESIGNATION
1	61.0404.01
2	70.0816.05
3	65.0008.01
4	65.0550.04
5	65.0562.01
6	65.0562.02
7	65.0562.03
8	65.0589.01
9	66.0792.01
10	BR 14
11	Castor
12	Cayuga White
13	Chancellor
14	Concord
15	Himrod
16	Himrod
17	Horizon
18	Ill 547-1

The second data structure to consider is a nursery (Fig. 4). A nursery allows a breeder to specify a location in which the vines are stored, environmental information about the location (e.g., weather and soil conditions), and other location descriptors. A nursery must also have a list of germplasm associated with it, most typically selected from an existing list in the BMS, described

above. Typically, each nursery has a unique list of vines for which observations will be taken. This can be uploaded from an excel spreadsheet.

### Figure 4: BMS Nursery Example

Shown here is an example data sheet for a nursery in the BMS. “ENTRY\_NO” and “PLOT\_NO” are assigned by the BMS, and are uninteresting. “DESIGNATION” holds the name of the vine and “CROSS” holds the parental information for the vine. “VINE\_ROW” and “VINE\_”NUMBERS” hold the location data for each vine. The overall format of a nursery is not different from an excel spreadsheet. Note that the traits observed here are traits that require only one observation.

DESIGNATION	ENTRY_NO	CROSS	PLOT_NO	NOTES	ACCESSION_NUMBER	ROOTSTOCK	VINE_ROW	VINE_NUMBERS
95.0300.02	1		1	-/-+/??; also at 35	35-14-052-54; 37-06-052		1	001,2
95.0302.01	2		2	-/-/?/?; also at 35	35-13-049-52; 37-06-093		1	003,4
Chancellor	3		3	Seibel 7053;	33-03-022		1	5
Concord	4		4	;	35-22-029; 33-53-043; 20-19-1		1	6
Steuben	5		5	;	35-22-028; 33-03-017,18; 21-1		1	7
PI 200569	6		6	;	35-22-027; 44-03-112; 35-04-0		1	8
95.0308.01	7		7	-/-+/??; also at 35	35-13-017,18; 37-07-110		1	009,10
95.0308.02	8		8	-/-+/??; also at 35	35-14-033,34; 37-08-081		1	011,12
95.0308.04	9		9	-/-+/??; also at 35	35-16-052-54; 37-07-041		1	013,14
97.0503.02	10		10	-/-/?/?; also at 35	35-15-049-52; 37-10-010		1	015,16
97.0535.03	11		11	-/-+/??; also at 35	35-16-025-27; 37-12-102		1	017,18
97.0551.04	12		12	-/-+/??; also at 35	35-16-035,36; 37-13-060		1	019,20
Eger 99-11.02	13		13	+/-/?/?;	37-16-101		1	021,22
Eger 99-11.03	14		14	+/-/?/?;	37-17-105		1	023,24
Eger 99-11.04	15		15	+/-/?/?;	37-17-110		1	025,26
95.0310.03	16		16	-/-+/??; also at 35	35-17-040-43; 37-08-007		1	029,30

The nursery data structure allows for the specification of multiple traits to be collected for the vines listed. There are several variable types. The most common variable is the observation variate, which is observed in the nursery. These variables can be selected by searching for the name of the trait that the variable is designed to describe, or by browsing the ontology to find the

specific trait with which the variable is associated. The other common type of trait is the germplasm descriptor. This type of trait may be an accession number for the vine, an alternate name for the vine, or other such descriptor. Because germplasm descriptors are stored in a different location in the MySQL database, extracting the information is more complicated than an observation variate.

The most restrictive feature of a nursery is the lack of support for observation replications. Each vine is permitted only one observation per variable. A researcher could not use a nursery to store disease ratings for a vine for different times in the growing season. Instead, nurseries store vineyard records and traits that require only one observation, such as the location of the vines in a vineyard, vine accession numbers, genetic information for the vine, and the color of the berries.

When creating a nursery, an existing nursery in the BMS can be selected as a template. This will copy all variables, environment, and location information from the existing nursery into the new nursery. Data from the existing nursery will not be copied, but the list of variables for which to collect data will be copied; the list of vines will also not be copied from the existing nursery.

After updating the location of the nursery, the vine list and observation data can be added. This approach greatly reduces the time to create a nursery, and ensures uniform information and format across all nurseries.

The final data structure is the trial (Fig. 5). A trial is very similar to a nursery in that it allows for the specification of location and environmental factors, requires that a list of vines be associated with it, and requires that certain variables be collected in it. As with nurseries, it is rare to re-use a list from a previous trial, as each trial typically has a different list of vines.

## Figure 5: BMS Trial Example

Shown here is an image of a BMS Trial measurements screen. This is similar to a nursery; the primary difference is the presence of the “REP\_NO” column, which allows for multiple observations of a trait for each vine in the trial. The overall format is not different from an excel spreadsheet. Note that the traits observed here are traits that can have multiple observations, and are not static for a vine.

 Add Measurements

100 Showing 1 to 58 of 58 entries

DESIGNATION	ENTRY_TYPE	ENTRY_NO	REP_NO	PLOT_NO	WINE_COLOR	YEAST_TREATMENT	NITROGEN_CODE	DATE_OF_PICKING	BERRY_BRIX	BERRY_pH_JU
95.0301.01	Test entry	7	1	1	R	Juice only			20.1	3.33
97.0517.02	Test entry	11	1	2	W	Juice only			22.4	3.29
Rulik	Test entry	21	1	3	W	EC1118	1 g/kg DAP at inoc.		19.5	3.28
84.0100.05/09	Test entry	30	1	4	W	EC1118	1 g/kg DAP at inoc.		20.4	3.39
GR8	Test entry	28	1	5	R	HP/EC1118/Alpha	1 g/kg DAP at inoc.		20.6	3.36
Cserezegi Fueszeres	Test entry	1	1	6	W	CS(24)/EC1118	1 g/kg DAP at inoc.		18.3	3.18
Rubin Tairovskii	Test entry	51	1	7	R	HP/EC1118/Alpha	1 g/kg DAP at inoc.		20.4	3.17
Felicia	Test entry	20	1	8	W	EC1118	1 g/kg DAP at inoc.		17	3.43
La Crescent	Test entry	2	1	9	W	CS(24)/EC1118	1 g/kg DAP at inoc.		20.4	3.06
97.0551.04	Test entry	25	1	10	W	Juice only			20.3	3.21
MN 1200	Test entry	12	1	11	R	HP/EC1118/Alpha	1 g/kg DAP at inoc.		20.8	3.08
Concord	Test entry	44	1	12	W	Juice only			15.5	3.36
81.0329.01	Test entry	50	1	13	W	Juice only			18.7	2.76
95.0302.01	Test entry	34	1	14	W	Juice only			21.8	3.3
81.0315.17	Test entry	45	1	15	W	EC1118	1 g/kg DAP at inoc.		17.3	2.97
97.0527.01	Test entry	49	1	16	W	Juice only			18.3	2.8

A trial allows for the specification of replication numbers for an experiment. The replication number indicates how many observations of each variable will be collected for each vine. This distinguishes a trial from a nursery, and makes trials the more appropriate data structure for storing experimental data that require multiple observations. All experimental data are stored in trials.

Like nurseries, trials permit the specification of an existing trial as a template. This will copy all of the location information, environmental information, and variables from the previous trial into the newly created trial. Like the nursery, the data and vine list will not be copied from the template trial. This is useful in many situations as experiments for disease ratings, cold tolerance, and other traits are repeated each year, and the same variables are assessed. This allows easy creation of a new trial from an existing one to maintain consistency between the type of information collected and the format of the trial.

Nurseries and trials are organized into folders, which allow organization by year or other parameter. Nurseries and trials are browsed for separately, but the folders are visible in either browsing view. This is analogous to only being able to see excel sheets when browsing the file system from Microsoft Excel, and only seeing Word documents when browsing the file system in Microsoft Word.

### *Breeding Management System Shortcomings*

There are numerous shortcomings to the BMS that the usefulness of the software, and need to be addressed. The major shortcoming of the BMS is the lack of a good search and reporting feature. A search feature does exist in the BMS and allows for the searching of vines by certain traits. However, the search is not functional when the BMS is configured for a general crop (i.e., not a built-in crop), which is the case with here. The BMS can be configured with a choice of several different databases specific to certain crops such as corn, groundnuts, and squash which contain breeding methods, and other such information specific to that crop. In the case that there is no specific database for a crop, a general crop database can be chosen. The search feature is not functional when the BMS is configured with a general crop database.

Even if the search function was operational for a general crop, it would not be sufficient; the breeders have desired functionality that is not met by the BMS search tool. The searching capabilities provided by the BMS do not provide the user with a way to identify all of the nurseries trials in which the vine can be found. There is no way to identify all of the vines with a certain parent or set of parents. There is no way to correct naming irregularities (misspellings, improperly formatted names), and there is no way to limit the existing search feature to a certain range of years. The years with which one pulls data from for the search is important, and environmental conditions change from year to year, and trait evaluation practices change over time. The breeding team at Cornell expressed a strong desire to be able to limit their searches to certain ranges of years. This is to control for climate change over several years, and the fact that for certain traits older data may not be as reliable as more recent data.

Additionally, there is no way to generate a report from the searches than can be conducted in the BMS. The trait search supported by the BMS returns a list of vines satisfying the specified criteria, but there is no way to export the information for the listed vines. The only way to export data from the BMS is by navigating to each trial or nursery, and exporting the excel sheet that represents that dataset. The breeders at Cornell have expressed a strong desire to be able to export all of the data, or data across multiple nurseries and trials for given vines. This functionality is not supported by the BMS. The trial and nursery reports generated by the BMS are well designed for extracting the data from a specific dataset, but not for consolidating information from multiple datasets.

One of the features that the breeding team likes the most is the metadata sheet that is exported in the report. This sheet contains all of the environmental and location data specified for the nursery or trial, and contains details for each variable collected in the nursery or trial. The variable



information includes the description, trait being assessed, scale, method, and valid values if applicable.

There were a number of technical problems with the BMS that have been addressed in software releases since this project was initiated in June 2013. First, it was previously impossible to delete a trial or nursery. If a mistake was made, one solution was to move the trial or nursery into a mistakes folder. A software update allowed for the deletion of trials and nurseries, but the name for the trial or nursery was not freed in the MySQL database. If a mistake was made in the trial or nursery, a new one had to be made with a slightly different name. Recently, it is now possible to fully delete a trial or nursery.

The need for a voice-to-spreadsheet software arises from a desire from the breeders to be able to record data on a medium that is not paper. Recording data on paper can be difficult in inclement weather. Additionally, a considerable amount of time must be spent recording the observations on paper and then transcribing the observations to a spreadsheet or other electronic medium, a process that is error-prone. Other data recording options were considered, but it was determined that audio recording would be the easiest to implement and most convenient method of data recording. The voice-to-spreadsheet software would also provide a convenient method to digitize paper records. The voice-to-spreadsheet software can be used to create spreadsheets in a format which can be easily uploaded to the BMS, with the goal of saving time and improving accuracy when compared to hand-entering data into a spreadsheet.

The development of scripts to create fieldbooks was a key component of a data system being developed for the breeders. Fieldbooks contain information about each vine planted in a vineyard, and can be used to record notes about vines. Should the audio recording of notes fail to work, the

fieldbooks will serve as an important backup as a data recording medium. The use of the BMS to store vineyard information will render previous scripts to produce the fieldbooks useless; the development of new scripts to produce fieldbooks from data stored in the BMS is a necessary component of the new data system. These scripts can be produced from either the search tool, or by directly retrieving data via the BMS user interface.

## **Methods**

### *Uploading Data to the BMS*

The process of entering data into the BMS starts by creating a trial or nursery. An existing trial or nursery can be used as a template assuming the type of experiment already exists as a trial or nursery in the BMS. Next a vine list must be created to associate with the study. The easiest way to do this is to navigate to a list already in the BMS and export it. This will create an excel representation of the list in the computer's download folder. The information in the excel sheet can simply be deleted to create a template file. This includes the names of vines, ID numbers, parental information, and metadata about the list such as name and description. Next, the names of the vines in the study need to be copied into the excel sheet in the correct column. The parental information for the vines must be pasted into the spreadsheet in the format "[female parent]/[male parent]". This may require the concatenation of columns if the parental information is stored separately, or a search-and-replace operation of the delimiting character is not a forward slash (/). The Entry column must be filled with incrementing numbers starting at 1, and the GID column should be populated with zeros. The name and description for the list must be specified in the metadata tab; the name must not be already in use by a list previously existing in the BMS. The list can then be saved and uploaded to the BMS.

Once the list for the trial or nursery has been saved, a fieldbook for the dataset can be exported. The fieldbook contains a metadata tab and a data tab. The fieldbook data tab has the variables across the top row of the sheet, and the names of the vines descending down the left column. Sorting by the name of the vine in the fieldbook data tab, and in the sheet containing the data will sort the vines in a matching order. The data can then be pasted into the fieldbook, saved, and uploaded to the BMS. The trial or nursery can then be saved in the appropriate folder.

The following MySQL (Fig. 6) query was run to assess how many studies have been uploaded to the BMS.

```
select COUNT(DISTINCT REPLACE(REPLACE(name, '-PLOTDATA', ''), '-  
ENVIRONMENT', '')) as 'STUDY_NAME' from project;
```

Please note that each study has three entries in the table “project”, two of which have “-PLOTDATA” or “-ENVIRONMENT” appended on the end. Furthermore, each folder has two entries in the list, and these counts were subtracted by hand.

The following query was used to count how many unique vines have been uploaded to the BMS.

```
select COUNT(DISTINCT desig) from LISTDATA;
```

A method in the vine search tool below was adapted to also report the total number of observations uploaded to the BMS.

The following query was used to extract all variables in the BMS:

```
select stdvar_name as 'NAME', stdvar_definition as 'DEFINITION', property as 'PROPERTY',  
method as 'METHOD', scale as 'SCALE' from standard_variable_details;
```

The results of this query combined with ontology definitions in a separate spreadsheet were used to calculate the total amount of defined variables and variables with observations or the BMS.. Please note the distance between certain tables such as “germplasm”, where information about each vine is stored, and “phenotype” where information about each observation is stored.(Fig 6.) The motivation for a graphical interface for the search tool (Fig. 7) was to create a user-friendly way to search the data in the Breeding Management System.

## Vine search tool

Figure 6: General BMS Schema

Shown here is the general schema for the BMS. Some tables have been omitted, but all major tables are included. Note the large number of tables that must be joined to gather information for a vine (GERMPLASM table), an study (PROJECT, PROJECTPROP tables), a measurement (PHENOTYPE table), and the trait observed (CV\_TERM table). The complexity of the schema results in long wait times for queries to return results.

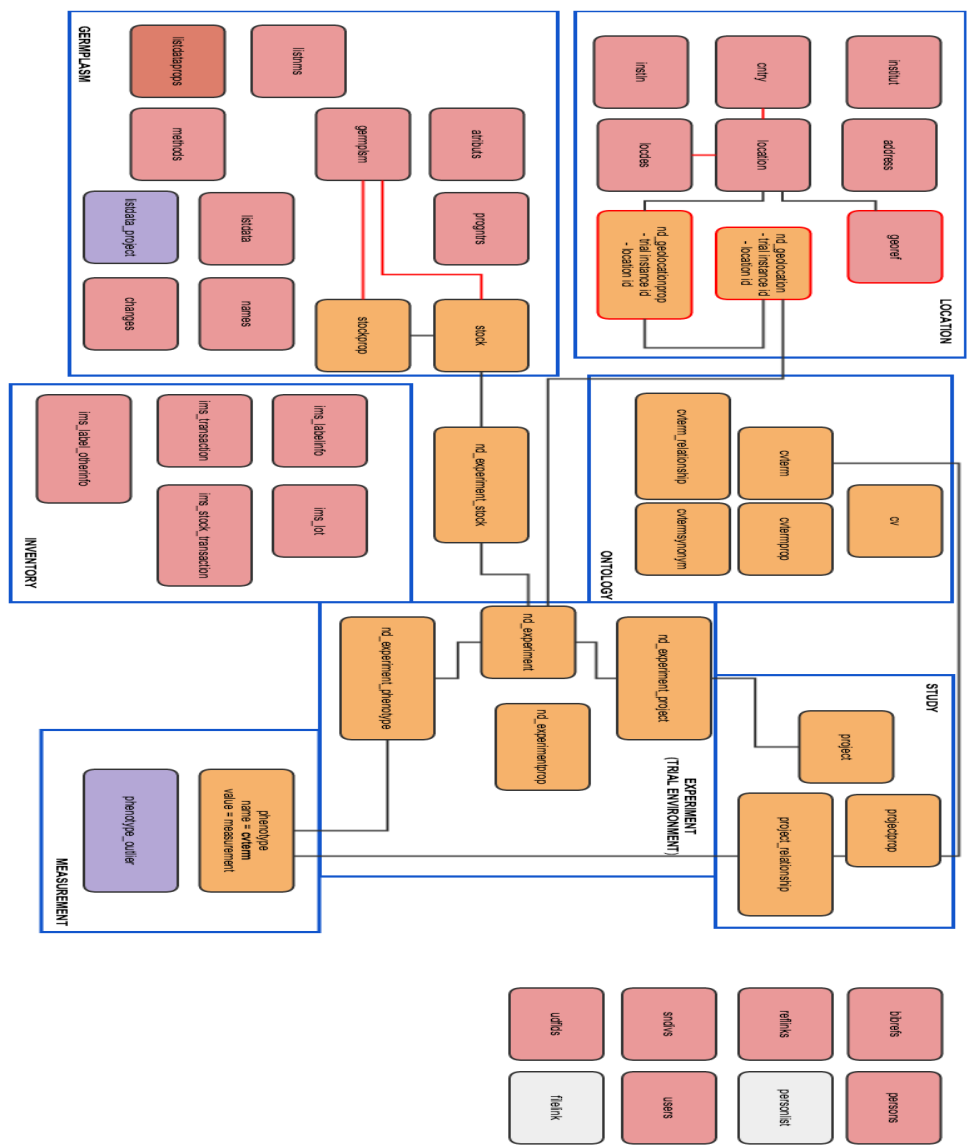
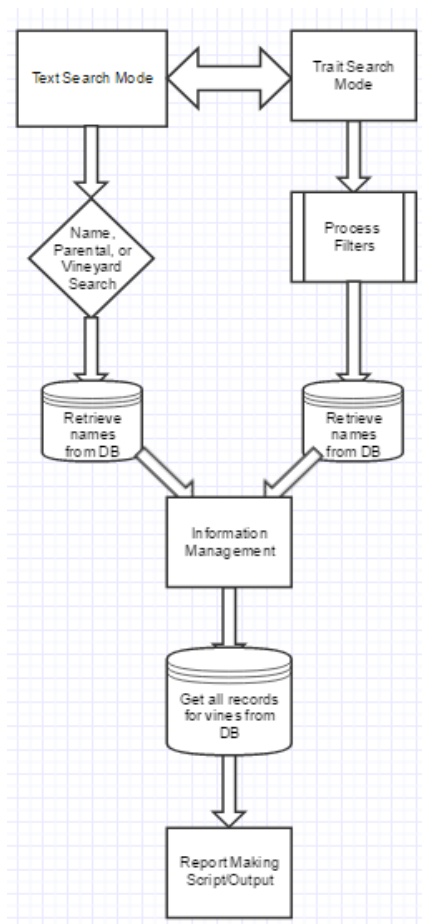


Figure 7: Search Tool General Workflow

Shown here is an image of the general program flow of the search tool. Users can toggle between Text Search Mode and Trait Search Mode. A search can be conducted from Text Search Mode by the name, parents, or studies a vine appears in. A list of vines is retrieved from the database based on the search criteria. The list of vines is given to the Information Management Window where users can specify information to include in the report, including vines not returned from the initial query. A second query is used to extract relevant information for each vine; this information is passed to the report formatting script, which outputs a tab-delimited file with the desired information and proper format.

Trait Search mode allows for the specification of filters to search against. These filters are processed, and then a vine



list is generated by querying the database. This vine list is passed to the Information Management Window

The trait search window allows users to create filters by which to search for traits. In this version of the search tool, the user can supply two extra options to a given filter. First, the user can supply a year range. When supplied, only values that come from studies that fall within the year range will be considered. If the year range specified is 2012-2015, only values from the years 2012, 2013, 2014, and 2015 will be considered. The user can select the type of filter from two options: Single Value and Average Value. Single value filters can be passed by any single value that meets the filter requirements. Average Value filters consider the average value of a vine for a given trait. The average value must pass the filter requirements.

A general message method is called to handle most exceptions. This will create a window to display the message to the user. There is a console that opens with the program, and minimal standard output is produced there. This output informs the user of what query is running, and the completion percentage of report generation.

### *Search Types in Detail*

The specifics of each search type are documented in the technical documentation (Appendix 23), but they will be reviewed in a general sense here.

**Name Search** is one of the primary search methods supported by the tool. When a name is entered, two variations of the name are generated: one with periods, and one with underscores to address the naming convention irregularities. These strings are passed through a function to

remove any MySQL wildcard characters. The WHERE clause of the MySQL uses a “like” operator, and appends the “%” wildcard character to the end of the string to find all names that begin with the string the user provided. What is returned is a list of all vines that match the search criteria, along with their internal ID numbers. The list of vines is used to populate the list of vines in the Information Management Window.

**Parental Search** mode allows users to search for vines by the names of their parents. Parental information is coded as “[female parent]/[male parent]” in a single column of a MySQL table. Users can enter the full parentage in the same format, or enter a whole or partial parent name. In the case of full parentage being supplied, only vines which parentage matches the entire parentage entered will be returned. In the case of a single parent or partial parent being supplied, vines that have a parent that matches the full or partial parent as either a male or female parent will be returned. The result of the search is a list of vines which satisfy the requirements of the parental search. These vines are used to populate the list of vines in the Information Management Window.

The **Vineyard Search** mode allows for the searching of vines based on the trials and nurseries in which the vines appear. Users can select which trial or nursery to search for by specifying the name of the study from a list. The list of studies in the BMS is retrieved from a query that is executed when the program is launched. Users can then start to enter the name of a study. As more characters are entered, the possible studies in the drop-down list is reduced to only those studies that start with the characters entered by the user. The first characters of most studies reflect the year in which they were conducted. The expected functionality is that the user will enter these characters first, and the contents of the drop-down list will be reduced to only studies in the year entered. From there, the user can select the appropriate study.



Once a study is selected from the list or manually entered in full, the user can search for the study. This will prompt the program to will retrieve all vines from the database that occur in the selected study. These vines will populate the vine list window in the Information Management Screen.

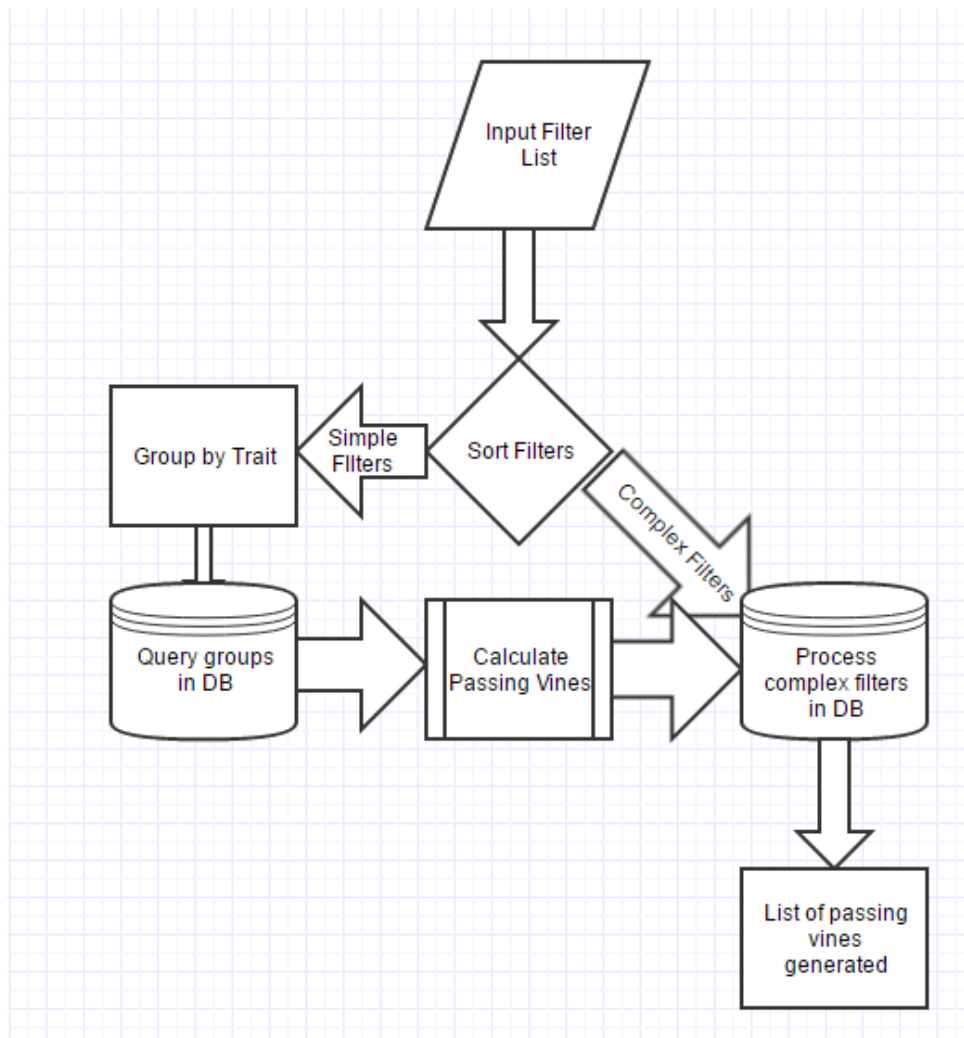
Filters are created in the **Trait Search** mode. Each filter has an operator, value, trait, and type. Optionally, a range of years can be specified. When a filter is created, there are several checks to ensure that the filter is valid. First, all four mandatory fields (trait, operator, value, type) must be defined. Second, the operator type is checked to ensure that it is one of the valid operators from the dropdown list. This check is a holdover from when the search tool was run on the command line and the user could, in theory, enter any set of characters for the operator. Since the operator is selected from a dropdown list now, there is likely no way to fail the check. The contents of each field are checked for MySQL wildcard characters, and will be rejected if they match the specified characters. The trait must be an existing trait in the database. The software maintains a list of all existing traits in the database; this list is retrieved from a query that is performed when the program is launched. If the trait does not exist in this list, the filter will be rejected. If the content of the value entry box is not recognized as a number, it must be surrounded in quotes. The reasoning behind this is to force users to acknowledge that they entered characters that are not numbers into the value box to help prevent them from creating un-passable filters (e.g. Height < i). If the operators “like” or “not like” are selected, the value must be text. Finally, the year range must be in the format #####-##### where the year on the left is less than or equal to the year on the right. Each of the above checks will produce a specific error message to help the user fix the problem.

### *Trait Filter Processing*

The processing of the filters (Fig. 8) was designed carefully to allow for any combination of filters and for minimum time to return results. The first step of processing occurs by simply splitting all of the filters entered into two groups: simple filters which are of type “Single\_Value” and have no year specifications and complex filters which are of type “Average” and/or have a year specification. The simple filters are processed first.

Figure 8: Trait Filter Processing Flow

Shown here is the trait filter processing flow. Filters are initially sorted into complex and simple filters by several attributes. Simple filters are organized into groups so that no group contains a duplicate trait type. Simple filter groups are then converted into MySQL queries, and run in the database. A list of vines is calculated that passed all simple filters. Complex filters are processed on the vines that passed the simple filtering. Each complex filter requires its own database query. If there are no simple filters, complex filters will be processed against every vine in the database. A final list of vines that passed all filters is returned.



In order to understand how simple filters are processed, it is necessary to understand how the MySQL query for the simple filters works. The WHERE clause of the query is constructed from the filter details. Many filters can be part of the same WHERE clause. The filters are linked by “or” operators, thus an observation of a trait can pass any one of the filters. The query then counts the number of unique traits with an observation that passed a filter for each vine. Each row of the query results contain the name of the vine and the number of unique traits with an observation that passed a filter. If a vine passed all filters, the number of unique traits that passed a filter will be equal to be total number of filters. If the number of unique traits is less, the vine did not pass.

With the above algorithm, there is no way to determine which filter a trait passed if there are multiple filters for the same trait, as only the unique traits that pass a filter are counted. For example, if there are two filters for the notes trait, a vine can have observations of notes which satisfy both filters, but this will only be counted once by the query, and the vine would ultimately fail the set of filters.

The solution is to create groups of filters with unique traits. Before adding a filter to a group, the program checks that no filters exist in the group with the same trait. If there are none, the filter is added to the group. If there is a duplicate, the filter is considered for the next group. If there are no groups remaining, a new group is created and the filter is added to it. Each group is processed as its own query, and the number of passed filters is summed from all groups processed. The sum of filters passed from all groups must be equal to the total number of simple filters to pass. If the sum from all groups is less, it indicates the vine one or more filters, and is not returned from the trait query. The list of vines that passed all simple filters is used to expedite the process of the complex filters.

The complex filters are of Average type and/or have a year range specification. The Average type indicates that the average value of the trait in question will be taken for all observations of the trait for each vine, and then compared to the operator and value. This takes a considerable amount of time. To expedite the process, the average value will only be taken for vines that pass the simple filters. This may reduce the number of vines for which an average needs to be taken from thousands to less than a dozen in some cases – a huge increase in speed for the query. The average will still be taken for all vines if no simple filters are created. For this reason it is recommended that an analogous simple filter be created for every average filter. The purpose of the simple filter is to eliminate all vines that have no possibility of passing the average filter before the average is taken. A filter that removes all vines that have zero observations of the trait exceeding the average value desired cannot pass the average filter, and time should not be wasted computing their average value for the trait in question. The program will detect if a user has created an average filter with no analogous simple filter, and will ask the user if one should be created automatically; in almost all cases the user should answer in the affirmative.

Filters which are of type “Single\_Value” but have a year range are considered complex filters because they cannot be grouped into the MySQL queries with filters that have no year requirement. The year requirement is not part of the MySQL query for the filter. For these filters, all observations that pass the trait, operator, and value requirements are returned along with the name of the study from which they occur. The year is extracted from the name of the study, and must be within the year range supplied. The average is then computed from the remaining values if the filter is of Average type, or any of the values that pass the filter requirements will pass a “Single\_Value” filter.

The complex filters are processed individually. If a vine does not pass a complex filter, it is

removed from consideration and is not processed in the remaining complex filters. Vines that pass all complex filters must have also passed all existing simple filters. These vines populate the vine list in the Information Management Window.

### *Information Management Window*

The information management window is created after all search types. All search types produce a list of vines which satisfy the search, and are used to populate the vine list on the left side of the information management window. Originally, this window was only available after a Trait Search, but it seemed natural and useful for all searches to produce this window, and each search method was adapted to do so. There are several options for including or excluding data in the report and specifying formatting options for the report here.

The list of vines that will be included in the report can be altered by using the drop-down box in the “Select Vines for Report” Box. The contents of the drop-down list will be reduced to reflect the characters the user enters into the box. The user can then specify to add or remove a vine from the list. Removing a vine that is not in the list or adding a vine that is already in the list will have no effect. Users may want to add vines to the list to compare against vines that were returned from the search.

Users can then specify which studies and traits to include in the report (Both images). From the trait selection window, users can add or remove specific traits from the list, add all traits to the list, or remove all traits from the list. Only traits listed in this window will be included in the report, so users can specify exactly what traits they would like information for. The study window functions in the exact same way with one exception. The entries in the study list which are only numbers are folders in the BMS. Including a folder will include all studies that are

located in the folder. The software maintains a list of traits and studies to include in the final report. These are written to temporary files for the report-making script. Traits or studies which do not occur in these files will not be included in the final report.

Users have the options to specify the format of the final report. By selecting the check-boxes labeled “Averages Only” and/or “Analysis Format” they can decide select the format they desire. The format requested is passed as an option to the report making script.

When the “Save” button is clicked, the user will be able to specify a file in which to save the final report. The program then processes the list of vines selected for the report. All information for each vine is pulled from the database, and the raw MySQL is written to a temporary file for the report formatting script to use. The file name, formatting options, and traits and studies to include are passed to the report formatting script which constructs the report.

Retrieving the data for each vine is the bottleneck in the search process. Originally, this retrieval was done before the generation of the Information Management Window; for large queries, this left users waiting an unreasonable amount of time to even see what vines were returned from the search. Instead, users should know exactly what vines and information would be included in the report before most of the computation is performed.

### *Options Menu*

From the options menu, a user can specify database connection information, update the trait values information, and manage vine name synonyms. Specifying database connections involves altering the hostname, port, username, and database name which the software uses to connect to the BMS’s MySQL database. These values are written to a file when they are saved, and are read in from the file when the program is launched. The most common item a user may wish to

change is the database name, as this will allow them to search through the data of another breeding program in the BMS. Note that there is no password field, as none is required by the BMS. A password requirement could be easily created if the permissions change. In this case, the password would be stored in memory for a given program session, and not written to a file under any circumstance. The user would have to enter the password each time a connection setting is changed.

The “Update Trait Info” button will prompt the program to re-calculate the values in the “Trait Values” window on the Trait Search Screen. This information includes the average value of the trait for all vines, the standard deviation, minimum and maximum values observed, the number of vines with an observation, and the total number of observations in the database. In the case that observations are stored as text rather than numbers, examples of observations are displayed instead of statistical information. This data is stored in a text file, and loaded into memory when the program is launched. Calculating this information takes a considerable amount of time, and would lead to user frustration if the calculation occurred every time the program was launched, or every time a trait was selected in the Trait Search window. For this reason, the user can decide when the update occurs. The user is recommended to update the information after data is entered into the BMS.

### *Vine Name Synonyms*

Vine Synonyms are a solution to a naming problem common in grape breeding. Vines may be assigned a new name if the variety is released for public use or over time as one commercial variety is disseminated and marketed under various names. Additionally, extra information may be appended to names during data uploads, such as a specific clone (sport) of the vine or passport



information reflecting its source. In any of these cases, the different names represent the same genotype, and all data under synonymous names should be considered for searches and outputted to reports when necessary. The name of the vine is assumed by the BMS and by the program described here to be a unique indicator of genotype. In these special cases, this assumption is broken and must be manually corrected by the user.

To create a synonym, the user must specify the undesirable name as the Hidden Name. The correct name is specified by the Shown Name. Adding the flag “---“ at the end of a Hidden Name tells to program to ignore all characters at the end of a name. The Synonym relationship represents a many-to-one relationship between hidden names and shown names. That is, many hidden names can be mapped to the same shown name, but one hidden name cannot be mapped to multiple shown names. These relationships are saved in a text file, and loaded when the program is launched. When a synonym is created or removed, the “Current Synonyms” pane on the left is updated to reflect the current synonyms that the program recognizes. The synonyms are sorted alphanumerically on the hidden name.

These synonyms are used to correct the names of vines in multiple locations in the program’s functionality. First, the name for all data checked in the trait search is checked against a list of hidden names. If the name associated with the data is a hidden name, it is corrected to the shown name so that the shown name is returned from the search. In this way, all data under all names can be used to pass trait filters. The Information Management Window checks to make sure that no hidden names are displayed in the vine name window. Any hidden name passed to it will be replaced with the correct shown name. When adding vines in the “Select Vines for Report” Box, if a hidden name is selected to be added from the list, the shown name will be added instead. Similarly, if a hidden name is selected to be removed from the list, the shown name that actually

exists in the list will be removed. Finally, when the user decides to create the report, the list of vines is passed to a function to retrieve all data for the vines. This function adds necessary hidden names or shown names to the list of vines to conduct queries so that all information is retrieved for the report. The report making script corrects all hidden names to shown names as well.

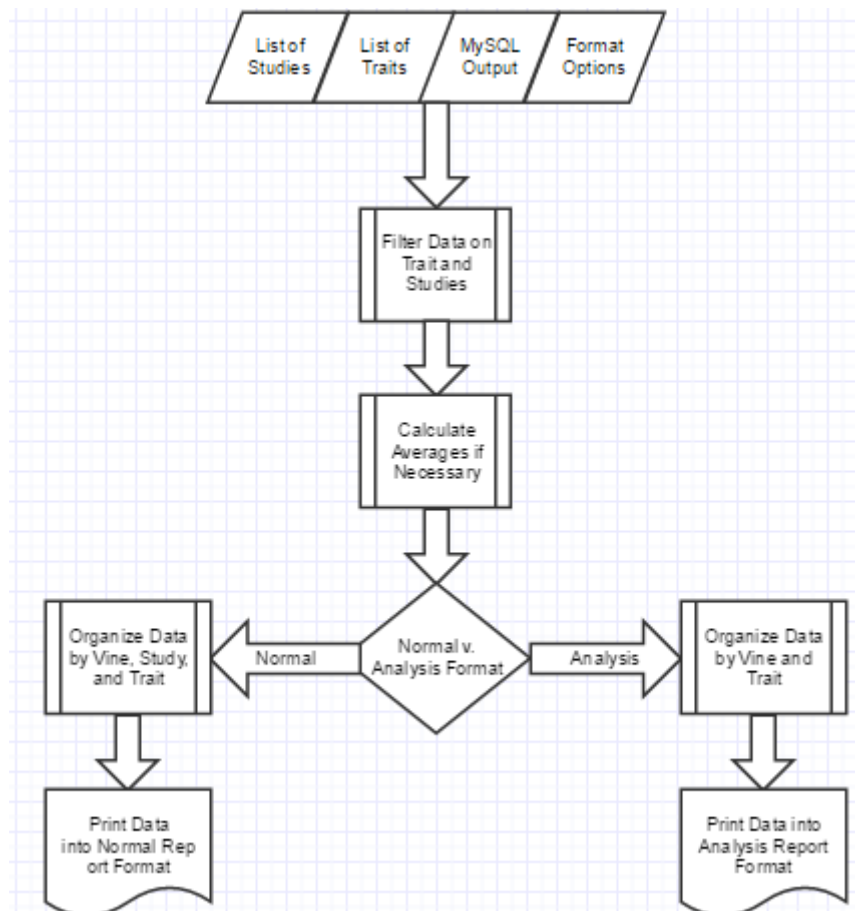
The Search Tool eliminates the possibility of cycles by only looking one step forward or backwards in the relationship. For example, if a hidden name is encountered, data for the shown name, and other hidden names that point to the shown name will be drawn. If a shown name is encountered, all data from all hidden names will be drawn. In no case will the program check to see if a shown name is also a hidden name, thus the possibility of chaining synonyms is eliminated.

### *Report Formatting*

Formatting the output (Fig. 9) became an important issue as the raw MySQL results were not readable or useful for a breeder. The first iteration of the formatting was to organize the report in the same way that the BMS stores data. This involves grouping all observations with the same “RELATED\_IDENTIFIER” onto the same line in the report. This would later become a selectable format for the reports. Using this format is recommended only when the results of a single nursery or trial are being reported. Otherwise, breeders desire a format which shows well-organized data for each individual vine.

Figure 9: Report Formatting Program Flow

Shown here is the program flow for the report making script. The script takes a number of inputs which include raw MySQL output, a list of studies to include in the report, a list of traits to include in the report, and formatting options. First the raw MySQL output is read in and organized in program memory. Data is filtered by the study and trait it appears in. Averages are calculated across vines and traits if specified in the formatting options. Data is then organized and converted into strings in different ways depending on whether normal or analysis format was specified. The strings are then printed to a file as the final output.



The report is broken up into sections for each vine, and subsections for each trial or nursery the vine appears in. For each trial or nursery, the variables collected are listed across a row. The variables are first sorted by numeric or text based information, with a notes column always appearing as the last variable (the length of the notes section can be very long). Each group of variables is alphabetized. Observations for each variable appear in the column below the variable. The subsections for trials and nurseries are sorted based on the data in which they occur; the date is extracted from the name of the study. At the end of the report, the variables that appear in the report are listed. The variables are sorted by type and alphabetized as before. The description and scale of the variable are also listed to provide the reader with more information about each variable.

Feedback to this style of report indicated that the high number of observations reported for each vine can make identifying important trends difficult. For this reason, a final averages format was created, which also contains sections for each vine as before. In this report though, there are no subsections for trials and nurseries. Instead, the average value of each variable is calculated for a vine, and displayed in the report. When a text-based variable is encountered, an observation from the most recent study available is used. The order of the variables is determined through the same sorting that has been described above. Appended at the end of the name of each variable is the number of observations that were used to calculate the average value. The purpose of the report is to show the general performance for each vine over the variables that have been observed for it.

When the analysis and average report options are both specified, the vines are displayed descending in the left-most column of the excel sheet. All of the variables across all vines are sorted according to the previously stated rules and displayed across the top row of the sheet. The

average value of each variable for each vine is displayed in the appropriate cell. The number of observations that were used to calculate each average value is not displayed.

The report formatting script is called by the graphical search tool. Once the user decides to save a report, a raw MySQL report is saved in a temporary file for the report-making function. The list of studies and traits to include in the report are also included in temporary files, and the formatting options are specified on the command-line. The search tool will remove the temporary files once the report is made, or when the program is launched.

### *Voice-to-spreadsheet*

The voice-to-spreadsheet pipeline (Fig. 10) allows data recorded in an audio format to be converted to an excel spreadsheet. The pipeline was developed with the use of Dragon Software, though any voice-to-text software could be used.

The first step in the pipeline is the audio recording of data. This must be done in a structured manner so that the resulting text can be parsed by software. The requirements for this pipeline are that the name of the vine, location, or some other identifying information about a sample be spoken to identify the sample. Next, the name of the trait being observed should be spoken, as well as a value for the trait. The pipeline works best when numerical scales are used for traits, though there is support for binary traits and notes fields. The speaker should speak slowly and clearly for the best translation to text.

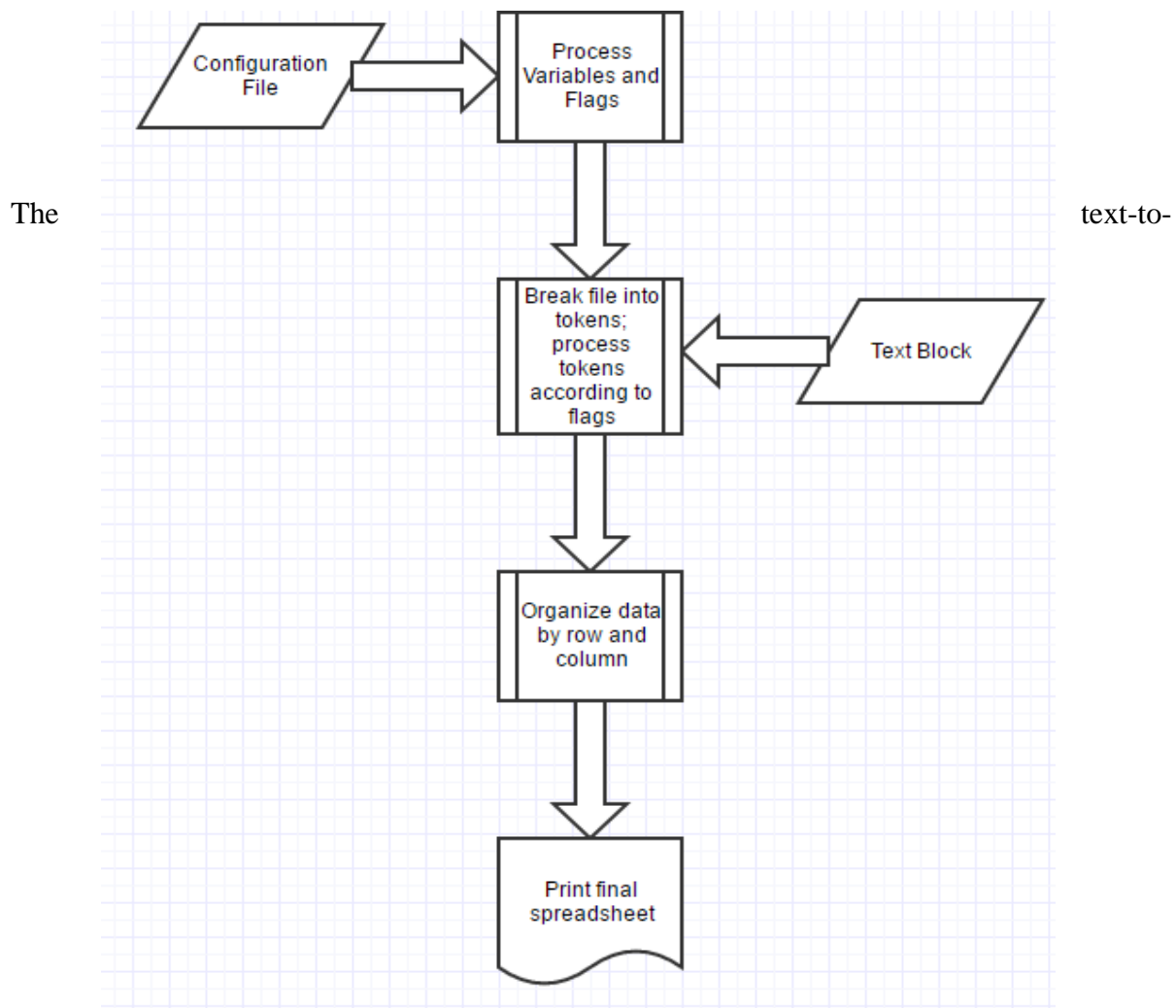
The audio can be recorded on any recording device. A cellphone and USB microphone were used in testing. Higher quality recording equipment may produce better results. This may be desired for noisy environments; wind, mechanical background noise, and other interference are

likely to limit the ability of Dragon to translate the recording. A key component of the recording device is the ability to transfer the audio files to a computer for Dragon to use.

Once the audio file has been uploaded to the computer, Dragon can be used to translate the audio into text. The result is a large block of text. This will be converted to a spreadsheet by the software described here.

Figure 10: Text-to-spreadsheet workflow

Shown here is the workflow of the Text-to-spreadsheet program. First a configuration file is read in which specifies expected traits and behaviors when they are encountered in the text block. The text block is then read in by the program, and broken into tokens. The tokens are then processed according to actions specified in the configuration file, and by determinations made by the program about what each type of token is (trait or value). The data is then organized into a spreadsheet format and printed to produce the final spreadsheet output.



spreadsheet software needs a configuration file to interpret the text it is given. The configuration file contains lines defining expected traits. Each line has several components: 1) the name of the trait as it will appear in the raw text, 2) the name of the trait as it should appear in the spreadsheet, and 3) flags to declare special behavior.

The flag “-l” indicates the start of a new entry. This flag should be associated with the identifying information of each individual. There can be traits with this flag. A new entry will not be created unless another trait has been observed that does not have this flag. This eliminates a situation where “row” and “plot” both have this flag, both traits are read in, and two entries are created. One entry would contain only “row” and the value, the other entry would contain the other information for the vine.

The “-N” flag indicates a notes field. This indicates that all data after this point should be considered a note. This allows users to make arbitrary comments that will be included in the spreadsheet.

The “-n” flag indicates the end of a notes field. Notes fields will continue until a trait with this flag is encountered. It is recommended to record notes as the last trait for each individual, and add the “-n” flag to traits with the “-l” flag. Thus the notes section ends with the start of the next individual. This will prevent runaway notes fields as well.

The “-w” flag indicates that there will be no value associated with the trait. This can be used for traits which only the presence/absence of is important.

The text-to-spreadsheet program begins operation by reading in the configuration file specified by the user. Known tokens are created based on the traits specified in the configuration file, and

value specified in a pre-loaded dictionary. The text (Fig. 11) is then parsed according to the traits in the configuration file. There are several warnings and errors that can occur during processing.

First, if a trait does not exist in the configuration file, a warning will appear. The surrounding text around the unknown trait is supplied so users can easily find the trait in the text file. The trait will be included in the resulting spreadsheet despite the fact that it was not recognized.

If a trait does not have a value afterwards, a warning will be produced (this warning will not occur if the trait has the “-w” flag). The cell where the value should exist is blank in the resulting spreadsheet.

If a value is not recognized by the program, a warning will be produced. The program recognizes all numbers as values, as well as a prebuilt dictionary of common numbers and homonyms, such as “one”, “won”, “two”, “too”, “to”, “tew”, “for”, “four”, “twenty”, “hundred”, “thousand”. If a value is not a valid number, or a recognized number in the dictionary, this error will be produced.

The script then organizes the data into a tab-delimited series of strings representing the rows of a spreadsheet. The result of the script is a tab separated file that can be opened in excel. Across the top row of the sheet are the variables that were present in the input text. Below is an example of text that was transcribed from audio to by Dragon software. The final transcription to a spreadsheet was performed by the software described above and can be seen in the results section.



---

## Figure 11: Sample Text Block

Shown here is a sample text block produced by Dragon software. This is converted by the Text-to-spreadsheet program into a spreadsheet.

```
seedling 26 runt downy mildew 1 seedling three downy mildew three seedling seven runt downy
mildew to seedling 1 downy mildew for seedling 15 note dead seedling 19 downy mildew 1 seedling
27 runt downy mildew 1 seedling eight downy mildew three seedling 12 downy mildew for seedling 16
downy mildew three seedling 28 runt downy mildew 1 seedling 24 note dead seedling 30 downy mildew
three seedling 34 downy mildew to seedling 38 downy mildew for seedling 42 downy mildew three
seedling 46 runt downy mildew for seedling 50 runt downy mildew for seedling 51 downy mildew for
note defoliant is likely due to downy mildew no sporulation seedling 47 downy mildew for seedling
43 downy mildew for seedling 39 downy mildew to seedling 35 runt downy mildew 1 note accidentally
uprooted seedling 31 runt downy mildew 1 seedling 52 downy mildew for seedling 49 downy mildew to
note small leaves seedling 40 downy mildew to note extensive necrosis could be downy mildew
seedling 44 note dead seedling 45 downy mildew to seedling 32 downy mildew seedling 36 downy
mildew for seedling 41 downy mildew three seedling 37 downy mildew 1 seedling 33 downy mildew
three seedling 76 downy mildew to seedling 73 downy mildew for seedling 69 downy mildew three
seedling 72 downy mildew 1 seedling 65 downy mildew 1 seedling 68 downy mildew to seedling 61
downy mildew 1 seedling 57 runt downy mildew 1 seedling 75 downy mildew 1 note tiny leaves
seedling 74 downy mildew 1 seedling 71 downy mildew to seedling 64 downy mildew for seedling 60
downy mildew three seedling 56 downy mildew for seedling 70 downy mildew three seedling 67 downy
mildew for seedling 63 downy mildew for seedling 66 downy mildew for seedling 62 downy mildew 1
seedling 59 downy mildew three seedling 58 downy mildew three seedling 54 downy mildew for
seedling 55 downy mildew three
```

---

### *Scripts for fieldbook printing*

One of the primary requirements for my system is the ability to print fieldbooks for the breeders in the two formats shown in appendix. There are two methods to generate these reports. First, a user can go directly to the nursery in the BMS and export the excel sheet. After navigating to the second tab, the user can save the sheet as a comma-separated file (.csv). This file can be specified for one of the two scripts to generate the files. The name and year of the vineyard must be specified on the command line for both scripts. The result is a file of the same name as the input, with “\_2ndTestFormat” or “\_SeedlingFormat” appended to the end of the filename.

The second method to produce the books is to use the vine search tool, though this method takes considerably longer. Users can search for a specific vineyard using the vineyard search mode. In the information management window, they can specify to only include data from the vineyard desired. The report should be formatted in “Analysis Format” only. This file can be fed to either script to produce the fieldbook.

For both fieldbook formats, the script must determine what row the vines are in, and create a page-break every time a new row is started. When plots are grouped together (e.g. 11-13,15), a descending list of numbers must be printed to represent the plots that the vines occupy. This is accomplished by splitting the group of numbers on commas. This produces a series of tokens. A token that is a single number represents a single plot. A token that contains two numbers separated by a hyphen represents a range of numbers. A token that contains a number followed by just a hyphen (12-) represents a single plot, and is the product of a formatting error.

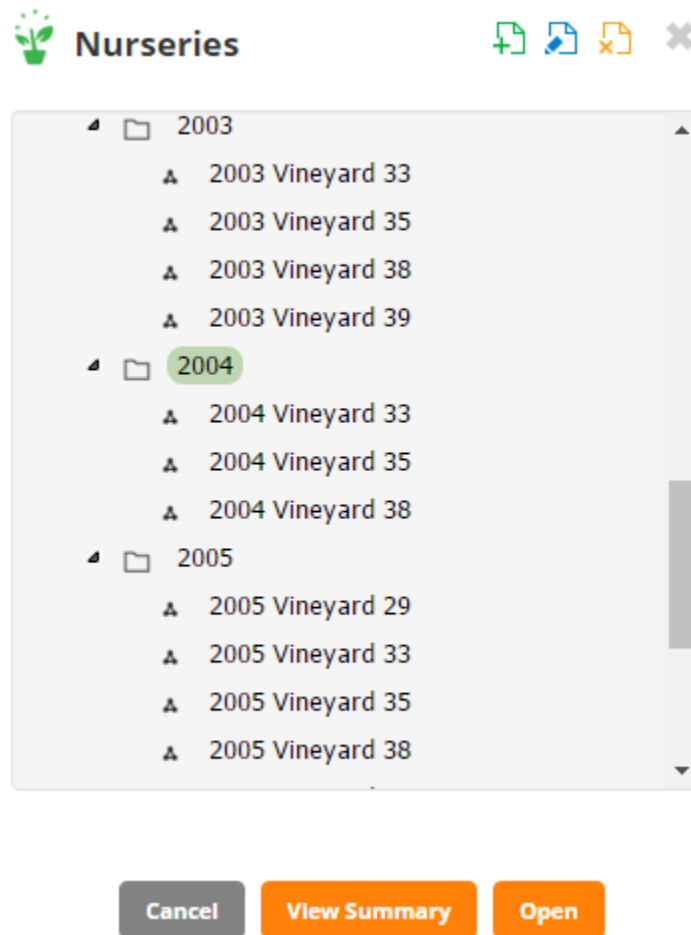
## **Results**

### *BMS Data Upload*

181 different nurseries (Fig. 12) and trials have been created in the BMS and populated with data. Data uploaded to the BMS dates back to 1992. There is currently a gap where years 2011-2014 are missing from the BMS. This data currently exists only in hard copy breeding notebooks and has never been entered into any electronic system.

## Figure 12: Nursery Browsing in the BMS

Shown here is an image of the nursery browsing screen in the BMS. Here, users can navigate through a file system that contains the vineyards of the breeding program, organized by year.



1806 genotypes have been uploaded to the BMS. This number is larger than the total unique genotypes in the BMS. As discussed earlier, there are sometimes multiple names for the same genotype.

There have been a total of 6,157,071 unique observations of variables (Fig. 13) uploaded to the BMS.

## Figure 13: Ontology Documentation

Shown here is a snapshot of the spreadsheet that is maintained to track ontology variables that have been defined.

All aspects of a variable that can be defined in the BMS are tracked in the spreadsheet. Fields highlighted in yellow are those that have been marked for review. This spreadsheet is maintained separately from the BMS, and is used for the records of the breeding program.

Variable Name	Description	Filter for Trait Classes
PM_DET_CAT	Categorical data describing relative powdery mildew severity in detached leaf assay	All->Crop Trait Ontology->Biotic Stress->Powdery Mildew
PM_ISOLATE	Type of isolate used in resistance assay (see above trait)	All->Crop Trait Ontology->Biotic Stress->Powdery Mildew->PM Isolate
PM_GH_INC	Percentage of leaves infected with Powery Mildew	All->Crop Trait Ontology->Biotic Stress->Powdery Mildew->Leaves Infected
PM_GH_MAX_SEVERITY	Percentage of leaf area covered by Powdery Mildew on most heavily infected leaf.	All->Crop Trait Ontology->Biotic Stress->Powdery Mildew->Maximum Severity
PM_GH_TYPICAL_SEVERITY	Percentage of leaf area covered by Powdery Mildew on a typical leaf.	All->Crop Trait Ontology->Biotic Stress->Powdery Mildew-> Typical Severity
PM_GH_INDEX	Combine resistance rating based on Powdery Mildew incidence and severity.	All->Crop Trait Ontology->Biotic Stress->Powdery Mildew Resistance Index
DM_DET	Categorical data describing relative downy mildew severity in detached leaf assay	All->Crop Trait Ontology->Biotic Stress->Downy Mildew->Downey Mildew Severity
PM_DET_INC	Powdery Mildew Incidence in detached leaf assay.	All->Crop Trait Ontology->Biotic Stress->Powdery Mildew->Powdery Mildew Incidence
PM_DET_SPOR	Powdery Mildew sporulation index in detached leaf assay.	All->Crop Trait Ontology->Biotic Stress->Powdery Mildew->Powdery Mildew Sporulation
PM_LEAF_P4	Powdery Mildew Parlier California 4 Point Scale on leaves.	All->Crop Trait Ontology->Biotic Stress->Powdery Mildew->Powdery Mildew on Leaves
PM_STEM_P4	Powdery Mildew Parlier California 4 Point Scale on stems.	All->Crop Trait Ontology->Biotic Stress->Powdery Mildew->Powdery Mildew on Stems
PM_RACHIS_P4	Powdery Mildew Parlier California 4 Point Scale on rachis.	All->Crop Trait Ontology->Biotic Stress->Powdery Mildew->Powdery Mildew on Rachis
PM_BERRIES_P4	Powdery Mildew Parlier California 4 Point Scale on berries.	All->Crop Trait Ontology->Biotic Stress->Powdery Mildew->Powdery Mildew on Berries
BERRY_WEIGHT	20 Berry Weight	All->Crop Trait Ontology->Morphological->Berry Weight
BERRY_BRIX	Berry Brix data	All->Crop Trait Ontology->Morphological->Berry Brix
BERRY_DIAMETER	Fresh weight of berry	All->Crop Trait Ontology-> Morphological-> Berry Diameter

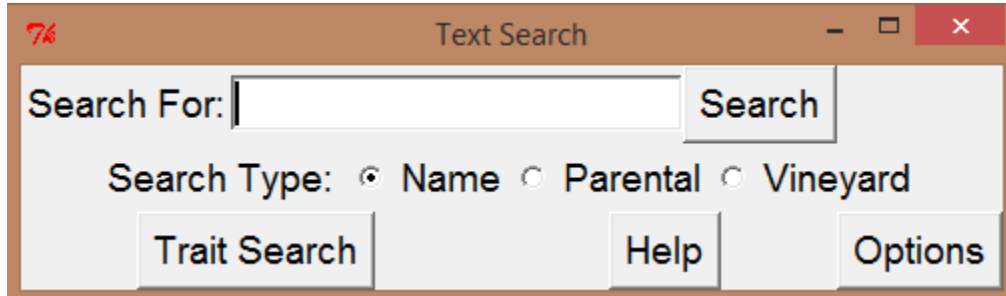
171 unique ontology variables have been described for use, and 69 variables have at least one observation in the BMS.

### *Vine Search Tool*

The Text Search screen is shown below (Fig. 14). The radio buttons at the bottom of the screen allow the user to toggle between **Name Search** mode, **Parental Search** mode, and **Vineyard Search** mode. Pressing the button labeled **Trait Search** (Fig. 15) will open the trait search window.

Figure 14: Text Search Screen

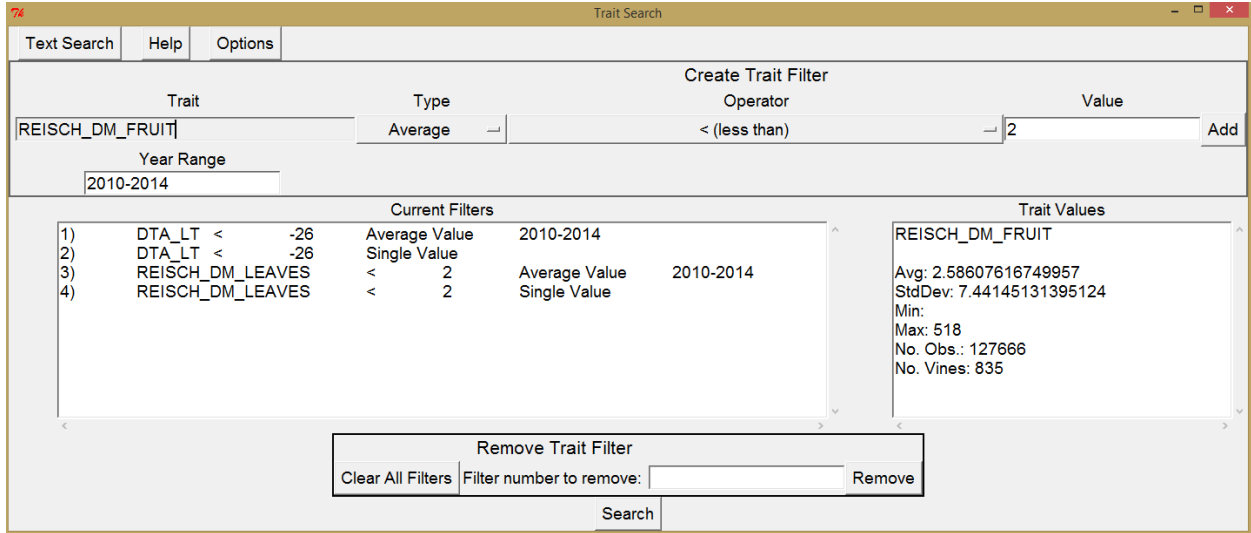
An image of the Text Search Screen. This screen is produced when the program is started.



Filters are created in the above screen by entering/selecting a trait from the dropdown list labeled “Trait”; selecting a trait will update the information in the “Trait Values” box. The type of filter can be selected from the dropdown list labeled “Type”. The operator can be selected from the dropdown list labeled “Operator”. The value for the filter can be entered into the field labeled “Value”. The optional year restriction can be entered into the box labeled “Year Range”. Clicking “Add” will add the filter to this list in “Current Filters”. Filters can be removed by entering the filter index (seen on the far left-hand side of the “Current Filters” list) into the box labeled “Filter number to remove”. All of the filters will be removed by pressing “Clear All Filters”.

Figure 15: Trait Search Mode

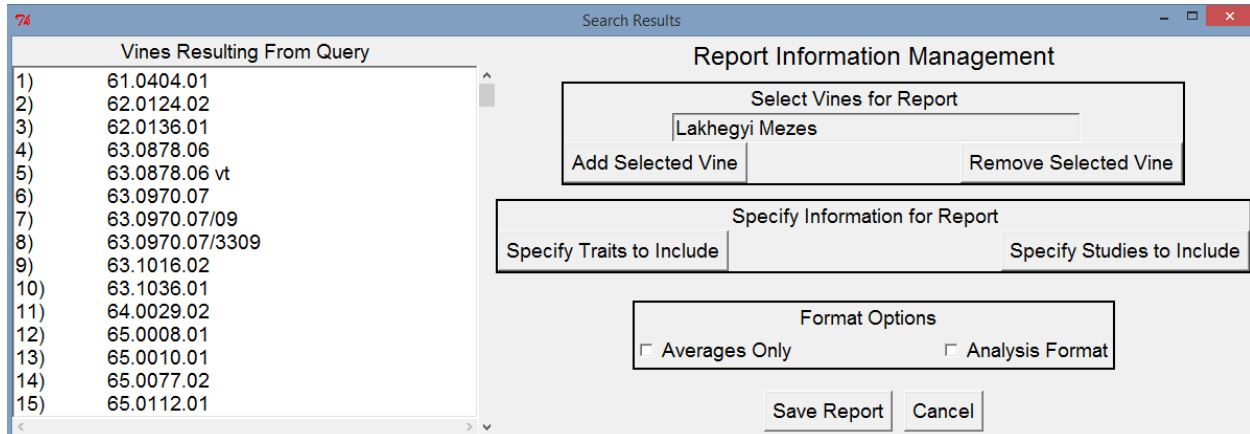
An image of the Trait Search screen. Filters are entered to search for vines with an average low temperature exotherm below -26, and an average Downy Mildew rating less than 2 for the years 2010-2014.



The results of all search types will result in the Information Management Screen (Fig. 16). From this screen, users can add or remove vines from the report as they choose, specify which traits and studies will be included in the report, and specify formatting options for the report. Saving the report will write a temporary file with the raw MySQL output which is then processed by the report maker script to produce the desired report.

Figure 16: Information Management Window

An image of the Information Management Window. This window is produced after every search. Users can specify information for the report in this window.

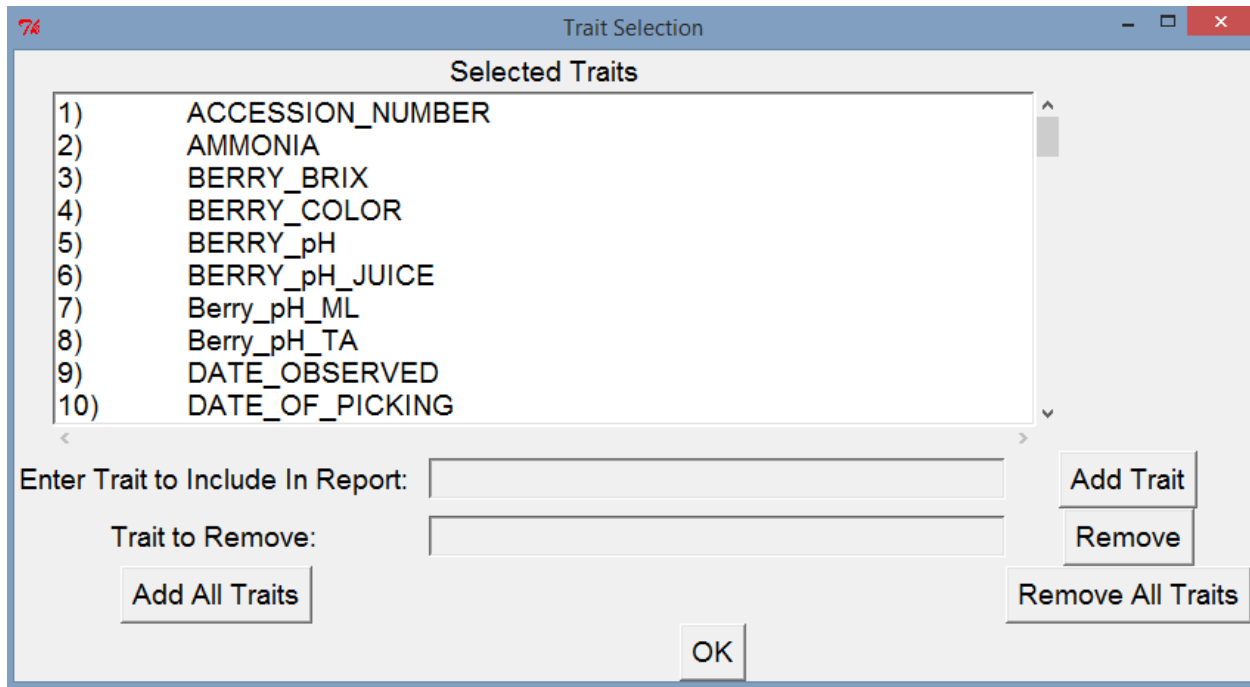


*Trait and Study Select Windows:*

The trait selection window (Fig. 17) allows users to specify which traits should appear in the final report. Users can add individual traits from the top drop-down bar. Users can remove specific traits from the report by selecting them from the lower drop-down bar. The contents of the lower drop down bar reflect the traits that appear in the “Selected Traits” list. Users can also add or remove all traits from the report with the appropriate buttons.

Figure 17: Trait Selection Window

This is the Trait Selection Window. Users can specify which traits should be included in the report here. This window is accessed from the Information Management Window.

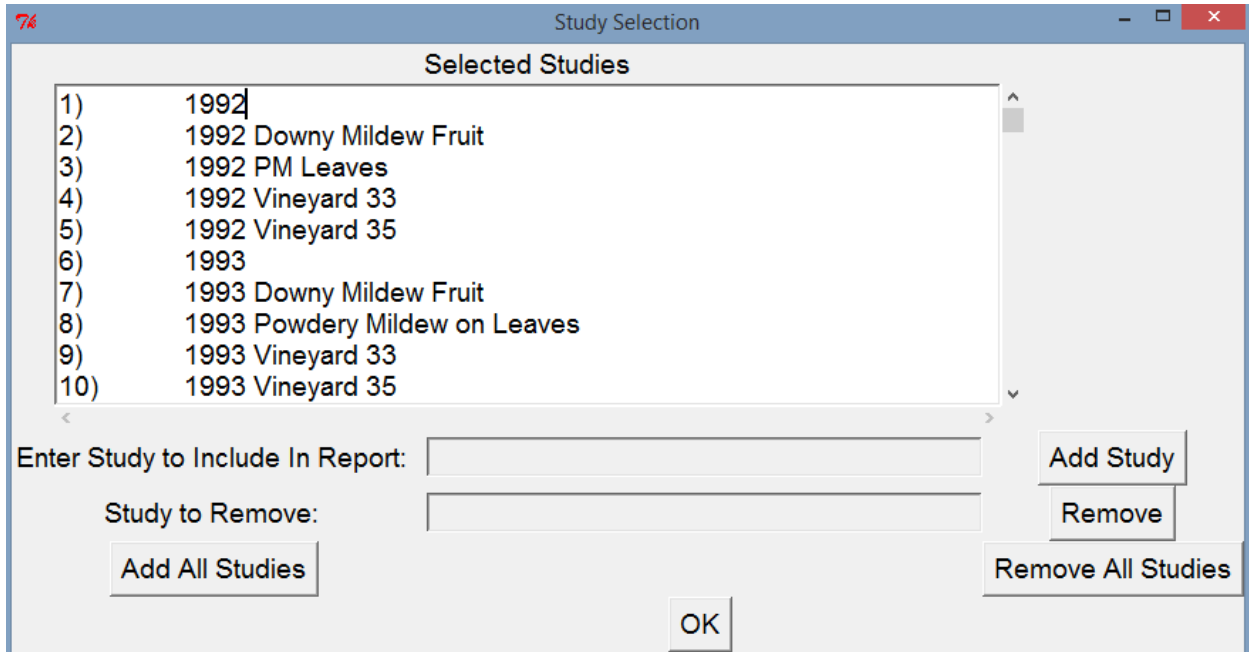


The study selection window (Fig. 18) allows users to specify which studies should appear in the final report. Users can add individual studies from the top drop-down bar. Users can remove specific studies from the report by selecting them from the lower drop-down bar. The contents of the lower drop down bar reflect the studies that appear in the “Selected Studies” list. Users can also add or remove all studies from the report with the appropriate buttons.



Figure 18: Study Selection Window

This is the Study Selection Window. Users can specify which studies should be included in the report here. This window is accessed from the Information Management Window.

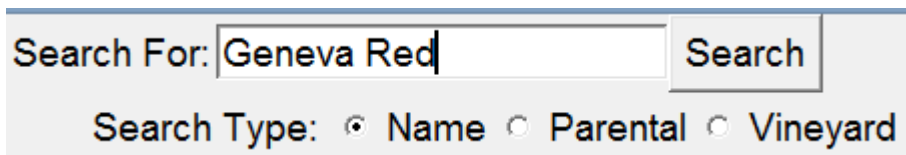


From the Text Search screen, a name search (Fig. 19) can be conducted when the "Name" radio button is selected.

*Text Search Example:*

Figure 19: Name Search Example

An example of a Name Search being conducted

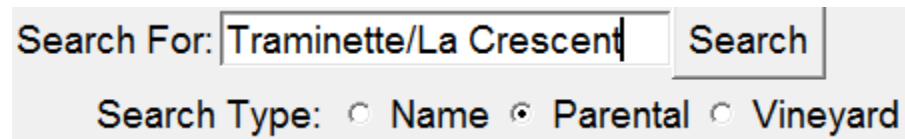


*Parental Search Example:*

The parental search mode (Fig. 20) allows for the searching of vines by their parental information. Parental search mode can be selected by selecting the “Parental” radio button from the list of radio buttons in the Text Search screen.

Figure 20: Parental Search Example

An example of a Parental Search being conducted



Search For:

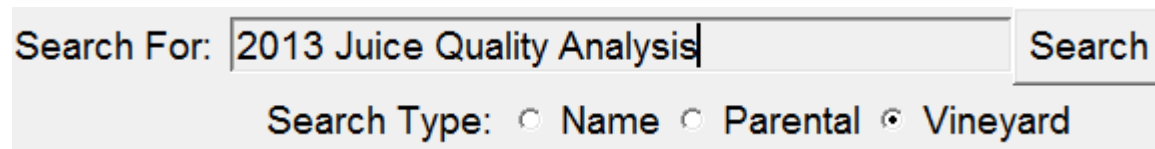
Search Type:  Name  Parental  Vineyard

*Vineyard Search Example:*

Vineyard search mode (Fig. 21) can be selected by clicking the radio button labeled “Vineyard” on the Text Search screen. When this radio button is clicked, the search bar that is displayed for the name and parental search types is replaced by a fill-in list. The drop-down list is populated with the names of all studies in the BMS.

Figure 21: Vineyard Search Example

An example of a Vineyard Search being conducted.



Search For:

Search Type:  Name  Parental  Vineyard

The expected format of a study name is [year] [study type]. It would be helpful to be able to enter the study type into the dropdown list, and reduce the contents of the list to only those studies of the same type. This functionality proved difficult to implement, and is ultimately not a

part of the current search tool.

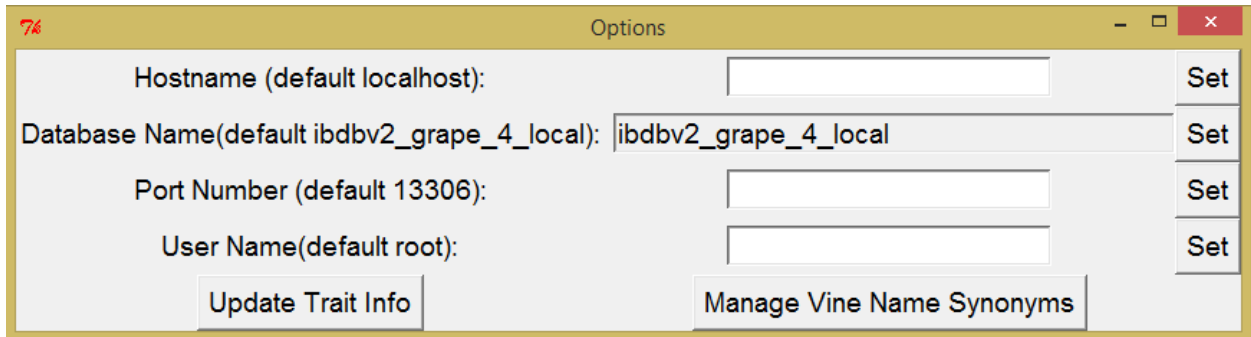
Report results of the vine search tool and report making function. Seen below are snapshots of reports in the four different report configurations. The full reports can be seen in appendix X and XI. All reports were generated from the same trait query with a single filter of  $DTA < -27$  Single\_Value. Concord was removed from the vine list in the Information Management Window. All studies and traits were included in the reports.

#### *Options Menu Screen:*

The options menu (Fig. 22) allows users to perform several different actions. First, database connection settings can be specified through the four fields at the top of the screen. The Database Name can be selected from a drop-down list. Entering a value and pressing “Set” will change the value for the current program session. A window will then ask the user if the setting should be saved for future sessions. Pressing “Update Trait Info” will update the information displayed for traits in the Trait Search screen. Vine synonym management can be accessed by pressing “Manage Vine Name Synonyms”.

Figure 22: Options Menu

An image of the options menu. This menu can be accessed from the Text Search and Trait Search windows.

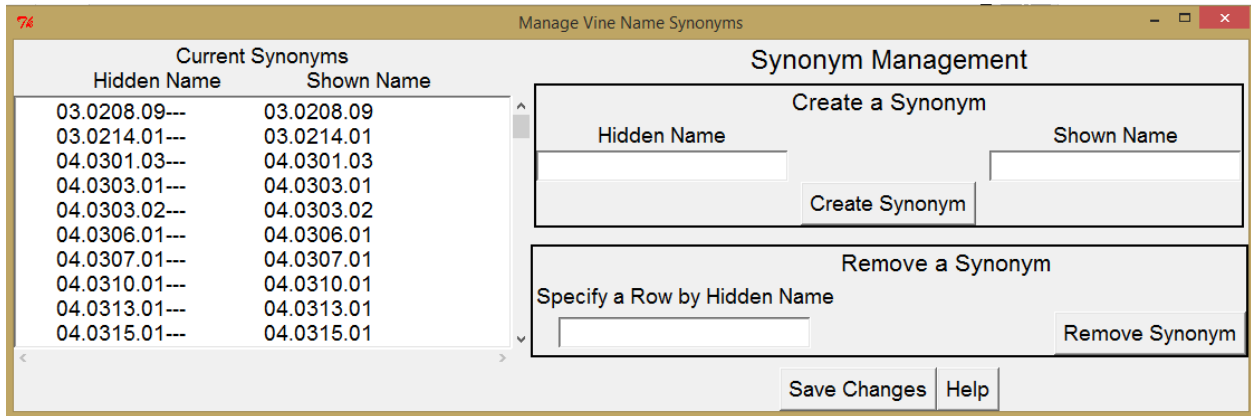


### *Vine Name Synonyms Management Window*

Synonyms can be created in the synonym management window (Fig. 23) by entering a Hidden Name and Shown Name into the appropriate boxes in the Create Synonym section. A Synonym can be removed by entering the Hidden Name of a row into the appropriate box in the Remove a Synonym section. Pressing “Save Changes” will save the synonyms to a file for later sessions. Existing synonyms are shown in the “Current Synonyms” list.

Figure 23: Synonym Management Window

Image of the Synonym Management Window. Here users can create and delete vine name synonyms recognized by the program.



## *Report Formats*

*Normal:*

The normal report format (Fig. 24) is produced when a user selects no additional formatting options in the Information Management Window.

Figure 24: Normal Report Format

A snapshot of the Normal report format. The format has major sections for each vine. Subsections are created for each study the vine participates in. The observations for each trait in the study are listed in each study sub-section. This format is specified in the Information Management Window by selecting none of the check-boxes in the “Format Options” section. The entire report can be seen in appendix NUMBER.

Study	Traits																		
2012 Juice	AMMONIUM	BERRY_BR	BERRY_pH	Berry_pH	Berry_pH	ENZ_PAN	HPLC_ACE	HPLC_CITR	HPLC_LAC	HPLC_MAI	HPLC_TAR	TITRATABI	TITRATABI	TITRATABI	TITRATABI	WINE_COI	YEAST_AV	YEAST_TREATMENT	
		1.6	20.9	2.86	2.83	2.75	100	0.183254	0.254336	0.496636	3.344855	6.497002	14.2	13.1	11.7	W	102	EC1118	
2012-2013	DTA_LT_AVG																		
		-25.9792																	
2013 Juice	AMMONIUM	BERRY_BR	BERRY_pH	Berry_pH	Berry_pH	ENZ_PAN	HPLC_ACE	HPLC_CITR	HPLC_LAC	HPLC_MAI	HPLC_TAR	TITRATABI	TITRATABI	TITRATABI	TITRATABI	WINE_COI	YEAST_AV	YEAST_TREATMENT	
		9.9	19.4	2.92	3.02	2.94	115	0.053122	0.304865	0.642425	5.843051	5.265963	13.7	14.78	13.73	W	125	EC1118	
2013 Wine	DATE_OF	BERRY_CC	DEFECTS	HYBRID	LABRUSCA	MUSCAT	NUMBER	REISCH_N	REISCH_PI	RESICH_PI	WINE_COI	YEAST_TREATMENT							
	#####	W	0	0	0	0	11	4.818182	4.818182	4.909091	W	EC1118							
2013-2014	DTA_LT_AVG																		
		-26.8913																	
2014 Juice	AMMONIUM	BERRY_BR	BERRY_pH	Berry_pH	Berry_pH	ENZ_PAN	HPLC_ACE	HPLC_CITR	HPLC_LAC	HPLC_MAI	HPLC_TAR	TITRATABI	TITRATABI	TITRATABI	TITRATABI	WINE_COI	YEAST_AV	YEAST_TREATMENT	
		0	20.4	3.18	3.03	2.93	210	0.15	0.4	0.58	5.55	4.9	13.75	12.91	13	W	210	EC1118	
2014-2015	DATE_OBS	DTA_LT	DTA_LT_AVG																
	#####	-26.1	-26.7583																
	#####	-26.3	-26.7583																
	#####	-27.3	-26.7583																
	#####	27.5	26.7583																

*Average:*

The average report format (Fig. 25) will be produced when a user selects the “Averages Only” box in the Information Management Window.

Figure 25: Average Report Format

A snapshot of the Average Report format. Here, the sub-sections for each study are collapsed as only the average value of a trait for each vine is reported. The number of observations is listed after each trait header. This format can be specified in the Information Management Window by selected the “Averages” check-box in the “Format Options” section. The entire spreadsheet can be seen in appendix NUMBER.

Vine	Cross							
03.0207.04		AMMONIA n=3	BERRY_BRIX n=3	BERRY_pH_JUICE n=3	Berry_pH_ML n=3	Berry_pH_TA n=3	DATE_OBSERVED n=3	
		4.4	20.2	2.893333333	3.09	2.966666667	42649.33333	
03.0207.06		AMMONIA n=3	BERRY_BRIX n=3	BERRY_pH_JUICE n=3	Berry_pH_ML n=3	Berry_pH_TA n=3	DEFECTS n=1	
		3.833333333	20.23333333	2.986666667	2.96	2.873333333	0	
03.0208.02		AMMONIA n=4	BERRY_BRIX n=4	BERRY_pH_JUICE n=4	Berry_pH_ML n=4	Berry_pH_TA n=4	DATE_OBSERVED n=3	
		3.9	18.725	3.0325	3.1475	3.0225	42649.33333	
04.0303.02		AMMONIA n=2	BERRY_BRIX n=4	BERRY_pH_JUICE n=4	Berry_pH_ML n=4	Berry_pH_TA n=4	DEFECTS n=1	
		3.7	19.6	3.0325	3.03	3.0575	0	
04.0310.03		AMMONIA n=2	BERRY_BRIX n=2	BERRY_pH_JUICE n=2	Berry_pH_ML n=2	Berry_pH_TA n=2	DTA_LT n=4	
		0.8	19.45	2.99	3.355	3.595	-26.15	
05.0403.01		AMMONIA n=2	BERRY_BRIX n=3	BERRY_pH_JUICE n=3	Berry_pH_ML n=3	Berry_pH_TA n=3	DTA_LT n=4	
		5.75	21.96666667	3.116666667	2.92	2.913333333	-26	
05.0403.02		AMMONIA n=2	BERRY_BRIX n=3	BERRY_pH_JUICE n=3	Berry_pH_ML n=3	Berry_pH_TA n=3	DTA_LT n=4	
		18.55	22.33333333	3.046666667	2.986666667	3.053333333	-26.1	
05.0403.09		AMMONIA n=2	BERRY_BRIX n=4	BERRY_pH_JUICE n=4	Berry_pH_ML n=4	Berry_pH_TA n=4	DEFECTS n=1	
		3.7	23.275	2.8875	2.7925	2.78	0	

*Analysis:*

The analysis report format (Fig. 26) will be produced when a user selects the “Analysis” box in the Information Management Window.

Figure 26: Analysis Report Format

A snapshot of the Analysis Report format. Here, all traits observed for all vines are listed across the top of the spreadsheet; vines are listed down the left-hand column. Each replication for a vine in a trial, and entry for a vine in a nursery is given a unique row. This format can be specified in the Information Management Window by selected the “Analysis” check-box in the “Format Options” section. The entire spreadsheet can be seen in appendix NUMBER.

VINE_NAME	CROSS	ACCESSION_NUMBER	AMMONIA	BERRY_BRX	BERRY_COLOR	BERRY_pH_JUICE	Berry_pH_ML	Berry_pH_TA	DATE_OBSERVED
03.0207.04		34-50-085	NA	NA	NA	NA	NA	NA	NA
03.0207.04		NA	NA	NA	W	NA	NA	NA	NA
03.0207.04		NA	2.5	21.1	NA	2.95	3.02	3.03	NA
03.0207.04		NA	NA	NA	NA	NA	NA	NA	1/12/2015
03.0207.04		NA	NA	NA	NA	NA	NA	NA	NA
03.0207.04		NA	NA	NA	NA	NA	NA	NA	42634
03.0207.04		NA	10.7	18.7	NA	2.93	3.2	3.02	NA
03.0207.04		NA	NA	NA	NA	NA	NA	NA	1/26/2015
03.0207.04		NA	NA	NA	NA	NA	NA	NA	NA
03.0207.04		NA	NA	NA	NA	NA	NA	NA	NA
03.0207.04		NA	NA	NA	NA	NA	NA	NA	1/12/2015
03.0207.04		34-50-085	NA	NA	NA	NA	NA	NA	NA
03.0207.04		NA	NA	NA	NA	NA	NA	NA	NA
03.0207.04		NA	NA	NA	NA	NA	NA	NA	42657
03.0207.04		NA	NA	NA	NA	NA	NA	NA	NA
03.0207.04		NA	NA	NA	NA	NA	NA	NA	1/26/2015

*Average and Analysis:*



The average-analysis report format (Fig. 27) will be produced when a user selects the boxes labeled “Averages Only” and “Analysis” in the Information Management Window.

Figure 27: Average-Analysis Format

A snapshot of the Average-Analysis Report format. Here, all traits observed for all vines are listed across the top of the spreadsheet; vines are listed down the left-hand column. Each vine is given only one row in the spreadsheet. The average value of each trait for each vine is reported here. This format can be specified in the Information Management Window by selected the “Analysis” check-box in the “Format Options” section. The entire spreadsheet can be seen in appendix NUMBER.

VINE_NAME	CROSS	AMMONIA	BERRY_BRIX	BERRY_pH_JUICE	Berry_pH_ML	Berry_pH_TA	DATE_OBSERVED	DEFECTS	DTA_LT	DTA_LT_AVG	ENZ_PAN_UNITECH	HPLC_ACETIC_ACID_ML	HPLC_ACETIC_ACID_TA
03.0207.04	f	4.4	20.2	2.893333333	3.09	2.966666667	42649.33333	0	-26.75	-27.41006729	228	NA	0.387655337
03.0207.06	f	3.833333333	20.23333333	2.986666667	2.96	2.873333333	1/12/2015	0	-26.8	-26.65063406	141.6666667	NA	0.128792013
03.0208.02	f	3.9	18.725	3.0325	3.1475	3.0225	42649.33333	0	-26.2	-26.37052763	94.75	NA	0.143467842
04.0303.02	f	3.7	19.6	3.0325	3.03	3.0575	1/12/2015	0	-25.7	-26.00992754	221.5	0.34	0.216226719
04.0310.03	f	0.8	19.45	2.99	3.355	3.595	1/11/2015	NA	-26.15	-26.22083333	201.5	0.330597568	0.607124553
05.0403.01	f	5.75	21.96666667	3.116666667	2.92	2.913333333	1/13/2015	NA	-26	-26.35992754	143	0.42	0.17482985
05.0403.02	f	18.55	22.33333333	3.046666667	2.986666667	3.053333333	1/27/2015	NA	-26.1	-26.38326087	229	0.22	0.161460831
05.0403.09	f	3.7	23.275	2.8875	2.7925	2.78	1/12/2015	0	-27.25	-27.09242754	166	0.34	0.185197456
05.0403.10	f	4.090666667	19.63333333	3.066666667	3.17	3.06	1/12/2015	0	-26.175	-26.12742754	149.6666667	NA	0.079244759
05.0403.11	f	7.666666667	21.53333333	3.233333333	3.36	3.28	1/12/2015	0.5	-27.725	-27.55242754	205	NA	0.230904604
05.0403.13	f	4.1	20.95	2.88	3	2.86	1/13/2015	0	-26.8	-26.23742754	115	NA	0.192234848
06.0504.01	f	16.5	19.35	2.925	3.165	3.705	1/11/2015	NA	-25.9	-25.97083333	150.5	0.814075626	1.036790688
06.0517.01	f	8.65	21.2	2.84	2.815	2.77	1/11/2015	NA	-25.4	-25.69576087	89	NA	0.142804989

*Voice-to-spreadsheet*

Below is a sample spreadsheet produced by the text-to-spreadsheet software (Fig. 28).

Figure 28: Text-to-spreadsheet Output

A snapshot of a spreadsheet produced by the Text-to-spreadsheet script.

DOWNY_MILDEW	NOTES	RUNT	SEEDLING
1		runt	26
3			3
2		runt	7
4			1
	dead		15
1			19
1		runt	27
3			8
4			12
3			16
1		runt	28
	dead		24
3			30
2			34
4			38
3			42
4		runt	46
4		runt	50
4	defoliant is likely due to downy mildew no sporulation		51
4			47
4			43
2			39
1	accidentally uprooted	runt	35
1		runt	31
4			52
2	small leaves		49
2	extensive necrosis could be downy mildew		40

Below are portions of Second Test Block Format (Fig. 29) and Seedling Format (Fig. 30) that are required by the Cornell Grape Breeding program, and be produced by scripts provided in this thesis.

*Second Test Block Format:*

## Figure 29: Second Test Block Report Format

Shown here are sample lines from a second test block report. The vine name and cross are shown on the top lines for each entry. Notes and Source information are shown below. The Vineyard Row is shown at the top of the page. The Plot Numbers for each vine are shown descending on the left. Traits of interest are shown left-aligned for each vine.

VINE	VINEYARD	35	ROW	1	YEAR	2013
------	----------	----	-----	---	------	------

1	95.0300.02		-			
2	NOTES:-/-/+/?/?; also at 35-14-052-54;					
	Source: 95.0300.02					
	BUD BRK	DM-FRT	CROP	BERRY	FLVR	R1-R5
	WI TRUNK	LVS	VIGOR	SIZE	TEXT	
	WI BUD	PM-FRT	CLU-SZ	COLOR	SHAPE	
	BLOOM	LVS	COMPAC	SEED		
	FLOWER	PDERM				

---

3	95.0302.01		-			
4	NOTES:-/-/-/?/?; also at 35-01-003;4;					
	Source: 95.0302.01					
	BUD BRK	DM-FRT	CROP	BERRY	FLVR	R1-R5
	WI TRUNK	LVS	VIGOR	SIZE	TEXT	
	WI BUD	PM-FRT	CLU-SZ	COLOR	SHAPE	
	BLOOM	LVS	COMPAC	SEED		
	FLOWER	PDERM				

---

5	Chancellor		-			
	NOTES:Seibel 7053;					
	Source: Chancellor					
	BUD BRK	DM-FRT	CROP	BERRY	FLVR	R1-R5
	WI TRUNK	LVS	VIGOR	SIZE	TEXT	
	WI BUD	PM-FRT	CLU-SZ	COLOR	SHAPE	
	BLOOM	LVS	COMPAC	SEED		
	FLOWER	PDERM				

---

*Seedling Format:*

## Figure 30: Seedling Format

Shown here is an example of the seedling report format. The Row Number is shown at the top of each page. The Plot Number, Vine Name, and Cross are shown on the top row for each entry. Notes are shown on the second line. The third line contains traits of interest for each vine.

VINE      VINEYARD 35              ROW 1              YEAR              2015

30      95.0310.03                      -

NOTES: -/-/+/?/?; also at 35-17-040-43;

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

-----  
31      97.0501.01                      -

NOTES: male; -/-/-/?/?; also at 35-17-054;55;

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

-----  
32      97.0501.01                      -

NOTES: male; -/-/-/?/?; also at 35-17-054;55;

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

-----  
33      97.0512.01                      -

NOTES: -/-/-/?/?; also at 35-18-003;4;

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

-----  
34      97.0512.01                      -

NOTES: -/-/-/?/?; also at 35-18-003;4;

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

## *Discussion*

### *Graphical Search Tool*

The original command-line interface was built using Perl due to my familiarity with the language and with text processing built-ins that were useful in organizing the search results. At the time, it made sense to build a simple GUI on top of the command-line interface. As the GUI has become more complex, creating well-designed screens has been difficult in Perl. I did not discover the difficulty until after some of the main screens had seen their first iteration and needed to be adjusted. If able to go back and change one thing in my thesis, it would be the choice of language in which the GUI was built.

The intent of the search tool is to be as simple and intuitive as possible. Ideally, users could determine the basic functionality of all parts from the interfaces alone. The Text Search screen is modeled to be a simple search bar that is present in all web browsers. The labels for the radio buttons should be informative enough for a user to understand what is being searched for. The Vineyard search type may be the least intuitive by name, but should become apparent as the drop-down list populates options based on what the user entered.

The Trait search mode is more difficult to understand at first glance due to the large amount of options on the screen. There are many message windows that can be triggered from here to help guide the user through the process of creating filters. One of the possible pitfalls may be that a user specifies a series of filters that no vines passes, and thus no results are returned. There is not much helpful feedback for the user in this case; they will not know where they went wrong in their search to return no results, or if the program is in fact to blame for the lack of results.

Ideally, the user will be able to understand the trait filter screen after reading the help message that can be displayed from the help button.

The Information Management Window is the easiest window to understand at a glance. The actions that a user can take are simple and can be described easily on buttons or labels on the screen. The windows specify traits and studies should be simple to understand as well. The feature most likely to confuse a new user will be the formatting options for the reports. A new user will not understand the results of selecting the formatting options without seeing examples of the reports each options will produce. Nonetheless, the fact that checking the boxes will alter the format of the report should be obvious from the labels around the boxes.

The structure of having a separate search tool and report making function was designed to allow for the replacement or modification of either component without affecting the other. I anticipate that the report making function may need modification or replacement to satisfy the needs of the Breeders who may want different formats for reports in the future. A new report making function can be created without modifying the search tool. Similarly, if the search tool needs modification or replacement, this can be accomplished without affecting the report making function. The report making function can also be used without the search tool; users would have to run their own MySQL queries, save the output to a temporary file (along with a list of studies and traits to include in the report), and call the report making script. Whichever component becomes outdated first can be replaced or modified without affecting the other.

There are two major assumptions made by the search tool and reporting script that need to be discussed here. First, all variables which the user would like to search need to be of the type “Observation Variate” when created. The type of variable (other examples include Germplasm Descriptor, Environmental Descriptor) tells the BMS where in the MySQL database to store the

information. Retrieving information from varying locations in the database proved difficult, and is not implemented in the search tool. The type specified for a variable impacts where it can be specified in a study or nursery. The Observation Variate type allows the variable to be added as a measurement to be taken in an experiment. Germplasm Descriptor indicates that the variable must be added in the trial descriptions tab, though there will still be an observation for every individual in the study. Environmental Descriptor tells the BMS that one observation needs to be taken to describe the environment of the nursery or trial (or one observation for every replication in a trial). In most cases, specifying a variable as Observation Variate will not impact its usefulness in the BMS, and allows it to be searched and retrieved by the search tool.

The other major assumption is that the year of the study is present in the name of the study (e.g. 2016 Vineyard 33, or 2016 Juice Quality Analysis Batch 1). The date of the study can be specified in the BMS; however, retrieving the date proved to be difficult. The date is stored in different location in the database depending on whether the dataset is a trial or nursery.

Furthermore, the default for a dataset is the date the dataset was created. If a user does not explicitly change the date, the date will be inaccurate. Adding the date to the name of the study is a good naming practice regardless, as duplicate trial names are not allowed. Adding the year to the beginning of the dataset name creates a uniform naming structure, naturally allows for unique dataset names indefinitely, and is easy to extract and manipulate without a more complicated and time consuming query for information that may not be reliable.

An update to the BMS could change the schema of the MySQL database. This may render the queries I have written obsolete. The BMS team indicated that an API will be released at some point in the future. The queries can be replaced with analogous API calls. Alterations to the MySQL queries to reflect the schema changes would also work, though using the API interface



would be a better long term solution as the API will not change as the schema changes. In (APPENDIX HERE), the MySQL queries used in the program are documented, as is the information expected to be returned for each query. As long as the expected information can be duplicated by API calls or updated queries, the search tool will continue to be functional.

The development of the Search and Report tool helps to solve one of the major tasks of the development of genetic markers, the extraction of useful data and the ability to find trends in the data. Extracting relevant information for developing disease resistance, that being disease ratings for vines, as well as the ability to extract well performing vines for the trait, allows researchers to quickly move into the analytics stage of marker development, and not have to spend time gathering data to begin. The Breeding Management System solves the problem of organizing data in a structured and consistent manner. Storing all records in a standard format, as well as the standardization of variables, allows the Search and Report tool to exist with relative simplicity.

### *Voice-to-spreadsheet*

Dragon is useful for its ability to manipulate the vocabulary. Dragon can be trained to identify complicated traits, and words similar to traits can be removed from the vocabulary. Other voice-to-text programs may not have the functionality to add new words to the vocabulary, or remove words that are similar to desired traits. The configuration file for the text-to-spreadsheet software is designed to handle short-comings of voice-to-text programs. Users can map many words that may appear in the text block to a certain trait. For example, very, vary, Barry, and bury all have been translated in the place of “berry”. With Dragon, these similar words can be removed from the vocabulary so that they will not appear again. Alternatively, all of the similar words can be mapped to the trait “berry” in the configuration file, and thus be recognized by the program.

The voice-to-spreadsheet method has been used to create data sheets for multiple recordings of disease resistance and plant status in vineyards on a cell phone, as well as multiple trial recordings in office settings on a microphone. The primary limitation of the process is the quality of the audio recording. There have been several times where a word appeared in the text block for which a close sounding variable could not be determined, or an expected value or trait was not found in the text block. In these situations, it was not possible to tell from the audio file what was spoken at the time. The common causes were wind noise disrupting the audio, and notification sounds on the phone being recorded while data was being spoken.

### *Ontology*

The variables chosen for creation in the ontology were selected out of immediate or predicted future need. As data sheets were delivered for upload, part of the process was to consider the variables in the data and determine if they matched an existing trait in the ontology, or if a new trait needed to be created. Breeding fieldbooks were also gathered and used to create variables for the ontology. Many of the unused variables are for data collected by other breeding programs. They have been defined if the need arrives to store data delivered from them.

The ontology (Fig. 31) comes with an existing basic structure. Some variables fit easily into the basic structure, others required more branches to be created to accommodate them. Many of the chemical assay traits needed their own unique branches, which are based off of the classification of molecule being assayed. Some of the higher branches group the traits into categories such as sugar and proteins, while lower branches further separate the molecules based on their properties. In cases where the variable was not easily understood, or not enough information

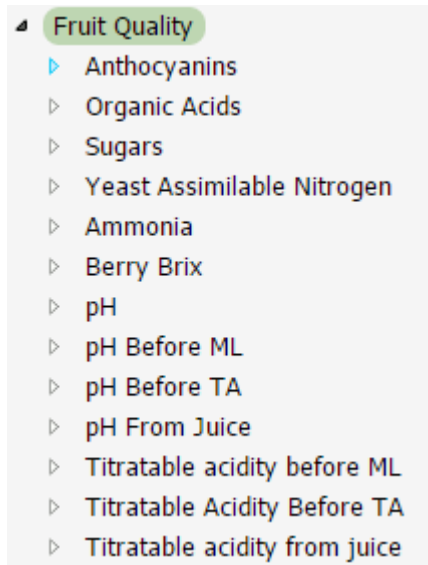
about the variable was immediately available to determine its place in the ontology, outside help was retrieved from those who understood the data best.

The creation of new branches of the ontology was primarily motivated by having enough similar variables to warrant the creation of a branch, or the existence of a variable that was vastly different from existing branches. The process that was used was to following the variable down the most logical branches until the most specific existing branch was found. The variable would then be compared to other variables on the same branch. If there were groups of variables that stood out as different, a new branch would be created for them. If the new variable was sufficiently different from others existing on that branch, a new branch would be created for the new variable.

Creation of new branches was avoided if possible as this may complicate the ontology in unnecessary ways. Since the ontology must be navigated by clicking to open new branches to reveal variables and branches.

## Figure 31: Ontology Browsing Screen

Shown here is an image of the ontology browsing screen. Note that each branch must be clicked to reveal lower branches and variables.



There is a balance to maintain between creating informative branches for grouping, and creating too many branches and making ontology navigation difficult.

The variable management of the BMS will play a large role in the standardization of data collection at Cornell and possibly in the VitisGen project group. Specific definition of how measurements are taken and standardization of the measurements across programs was well received at the *VitisGen* annual meeting. I was not the only researcher who had encountered problems with data standardization, and there was a consensus that more effort needed to be taken to regulate how common traits were being assessed. The specific variable definition allowed by the BMS helped me to work with researchers in the breeding program and other labs to specifically define how traits and other measurements are taken. Currently there is an excel spreadsheet with all of the defined variables, over one hundred have been currently defined. By

defining variables in this way, it is possible to achieve high regularity in how data is collected in the Cornell program. By sharing our variable definitions with other breeding programs, we may also be able to achieve consensus on how regular traits should be evaluated.

The structure of the underlying MySQL database was a key factor in deciding how to select the role of each variable, as can be done when creating a variable. The role of the variable dictates where in the database the BMS will store data collected for the variable. In order to simplify the data extraction process, the decision was made to select “Observation Variate” as the role of each variable even though the role of “Germplasm Descriptor” may better suit a variable called “Accession Number”. This is contradictory to the best aspects of the ontology, which is the ability to create detailed and accurate variable descriptions, but in this case a compromise was made.

### *Data Entry*

Data entry for the 181 nurseries and trials was done by hand. The two reasons that this process was not automated were the complexity of entering data into the MySQL database, and the lack of a long term need for such automation. Initial assessment of the automation of data upload proved that while pulling data out of the BMS was simple, properly automating the process was likely to take longer than manually uploading the data, and would be likely to introduce flaws in the data. Additionally, there is no expected need for bulk uploads of data in the future; there is data for about three years that needs to be entered, but this will constitute a vastly smaller amount of data sets to upload. Ultimately the process of manually entering data took about three

working days (about 24 hours total), and was more time efficient than developing a single-use routine for bulk data upload.

Additionally, the permissions set for the MySQL database for the root user do not allow the altering or entering of data. Working around this issue would only add to the time needed to develop a working method to enter data directly into the database. Manual entry was the most effective, and boring, way to solve the problem.

The Breeding Management System provides much of the basic functionality for storing, organizing, and standardizing data for the Cornell Grape Breeding program. There is, however, functionality required by the breeding program that is not provided by the BMS. Software to search, generate custom reports, produce fieldbooks, and have defined traits assessed by breeders, technicians, etc. have been built to accommodate these needs. The system developed here received highly positive feedback at the annual *VitisGen* project meeting. Interest was expressed by other breeding programs about using the system. To this point, none of the programs that expressed interest have attempted to use the BMS and the custom tools here. Transitioning to a new data management system takes a large amount of internal momentum and commitment from the breeding program; perhaps the success of the system in the Cornell program will convince other breeders to adopt the system as well.

### *Breeding Management System Additional Functionality*

The Breeding Management System has tools that can be used to help breeders make decisions on which vines to propagate and which to discard, and to create new nurseries based on certain crosses. For the Cornell program, we found these tools to be rigid, and that it was easier for the breeder to handle these decisions outside of the BMS. Fig.

## Citations

- Akkurt, Murat, Leocir Welter, Erika Maul, Reinhard Töpfer, and Eva Zyprian. Development of SCAR Markers Linked to Powdery Mildew (*Uncinula Necator*) Resistance in Grapevine (*Vitis Vinifera* L. and *Vitis* Sp.). *Molecular Breeding* 19.2 (2007): 103-11.
- Alleweldt, G., and J.v. Possingham. Progress in Grapevine Breeding. *Theoretical and Applied Genetics* 75.5 (1988): n. pag.
- Bowen, Pat, Jim Menzies, David Ehret, Lacey Samuels, and Anthony D.M. Glass. Soluble Silicon Sprays Inhibit Powdery Mildew Development on Grape Leaves. *Journal of the American Society for Horticultural Science*. N.p., Nov. 1992. Web. 30 Oct. 2014.
- Delannay, Xavier, Graham McLaren, and Jean-Marcel Ribaut. Fostering Molecular Breeding in Developing Countries. *Molecular Breeding* 29.4 (2012): 857-73.
- Barba, Paola, Lance Cadle-Davidson, James Harriman, Jeffrey C. Glaubitz, Siraprapa Brooks, Katie Hyma, and Bruce Reisch. Grapevine Powdery Mildew Resistance and Susceptibility Loci Identified on a High-resolution SNP Map. *Theoretical and Applied Genetics* 127.1 (2014): 73-84.
- The IBP Breeding Management System Version 3.0.8 (2015) The Integrated Breeding Platform. <https://www.integratedbreeding.net/breeding-management-system>.
- Dalbo, M. A., Ye, G. N., Weeden, N. F., Wilcox, W. F., and Reisch, B. I. 2001. Marker-assisted selection for powdery mildew resistance in grapes. *J. Am. Soc. Hortic. Sci.* 126:83-89.
- E-Brida: A Database for Breeders. Agri Information Partners.
- Fischer, B. M., Salakhutdinov, I., Akkurt, M., Eibach, R., Edwards, K. J., Topfer, R., and Zyprian, E. M. 2004. Quantitative trait locus analysis of fungal disease resistance factors on a molecular map of grapevine. *Theor. Appl. Genet.* 108:501-515.

Gohre, V., and Robatzek, S. 2008. Breaking the barriers: Microbial effector molecules subvert plant immunity. *Annu. Rev. Phytopathol.* 46:189-215.

Fung, R. W.m., M. Gonzalo, C. Fekete, L. G. Kovacs, Y. He, E. Marsh, L. M. McIntyre, D. P. Schachtman, and W. Qiu. Powdery Mildew Induces Defense-Oriented Reprogramming of the Transcriptome in a Susceptible But Not in a Resistant Grapevine. *Plant Physiology* 146.1 (2008): 236-49.

MKF Research LLC. The Impact of Wine, Grapes and Grape Products on the American Economy 2007. Publication. Wine Institute, National Grape and Wine Initiative, Wine America, WineGrape Growers of America, n.d. Web.

Multize, D. K. AGROBASE/4: A Microcomputer Database Management and Analysis System for Plant Breeding and Agronomy. *Agronomy Journal* 82.5 (1990): 1016. Web.

Ramming, David W., Franka Gabler, Joe Smilanick, Molly Cadle-Davidson, Paola Barba, Siraprapa Mahanil, and Lance Cadle-Davidson. "A Single Dominant Locus, Ren4 , Confers Rapid Non-Race-Specific Resistance to Grapevine Powdery Mildew." *Phytopathology* 101.4 (2011): 502-08 Production of Grape by countries. UN Food & Agriculture Organization. 2011.

Unity Project - Phenome Networks. Phenome Networks. Phenome Networks, 2014.

Tonietto, Jorge, and Alain Carbonneau. A Multicriteria Climatic Classification System for Grape-growing Regions Worldwide. *Agricultural and Forest Meteorology* 124.1-2 (2004): 81-97.

USDA-NASS. 2006. Agricultural Chemical Usage Fruit Summary. Published online by USDA-NASS at <http://www.nass.usda.gov/>.



Welter, Leocir J., Nilgün Göktürk-Baydar, Murat Akkurt, Erika Maul, Rudolf Eibach, Reinhard Töpfer, and Eva M. Zyprian. Genetic Mapping and Localization of Quantitative Trait Loci Affecting Fungal Disease Resistance and Leaf Morphology in Grapevine (*Vitis Vinifera* L). *Molecular Breeding* 20.4 (2007): 359-74.

## **Appendix I: Search Tool User Documentation**

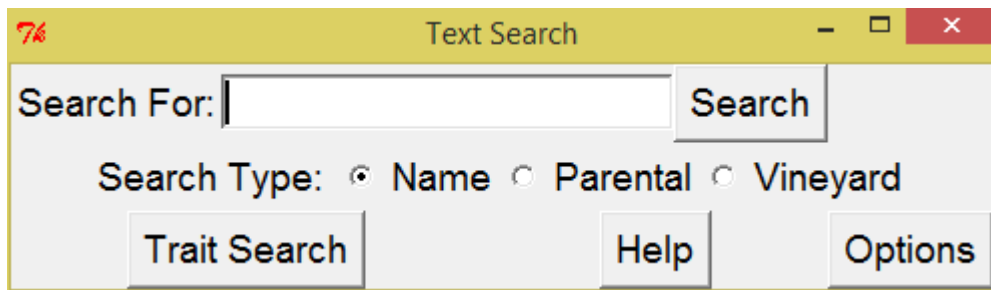
### **Search Tool Usage:**

#### **To Open:**

Click the desktop icon called vine search to open the tool.

#### **Search for vines by name**

Make sure the radio button on the main screen is selected for the search type “Name”.



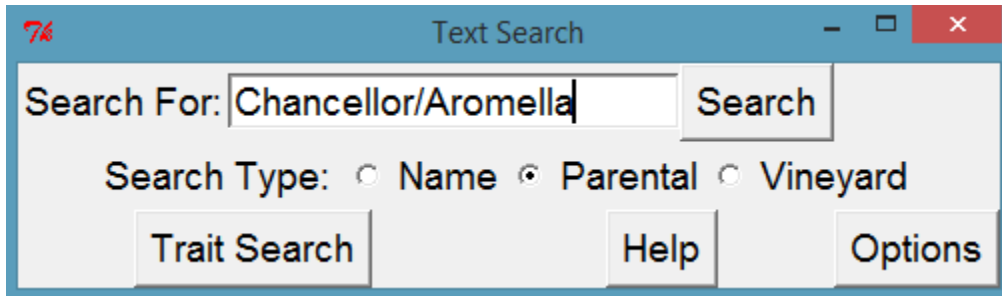
Enter the full or partial name of a vine. A search conducted using a partial vine name will return all vines that start with the name fragment entered.

Please note that all vines that contain an underscore version (ex: 04\_0406\_03) and period version (04.0406.03) will be returned

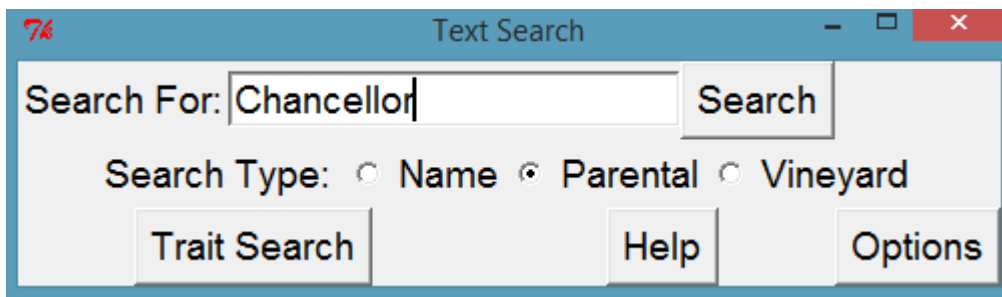
#### **Search for vines by parents**

Check the radio button next to “Parental” search type.

Enter the full or partial parentage of a vine into the search bar. An example of a full name is “Chancellor/Aromella”. This will return all vines with Chancellor as the female parent and Aromella as the male parent. Notice that the parents are entered together, separated by a forward-slash.



Searching for a single parent like “Chancellor” will return all vines that have “Chancellor” in their parentage.

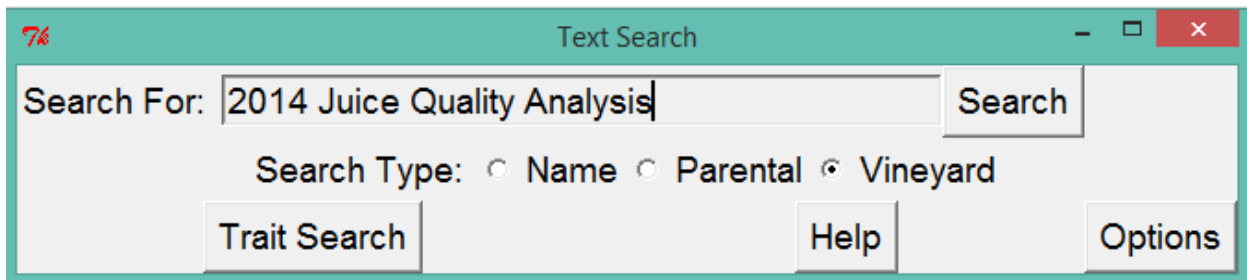


All underscored and period versions of parent names will be checked.

### **Search for vines by vineyard**

Click the radio button next to “Vineyard”.

This will change the search box into a drop-down menu. Here users can select a vineyard or study to search by. All vines that appear in the study the user search for will be returned. If the user selects a single year from the drop-down menu instead, all vines that appear in that year will be returned.



Users will not be allowed to enter the name of a vineyard or study that is not recognized by the software.

### Search for vines by traits

Click the “Trait Search” button at the bottom of the window the “Trait Search” window. Users can always return to the previous screen by clicking “Text Search” in the upper left corner of this new screen.

To start, users must enter the name of a trait they wish to search by in the box labeled “Trait”.

This box is a drop-down menu, and you can select any trait in the menu. Clicking on a trait, or pressing the return key when a trait is highlighted will bring up information about the trait in the “Trait Values” box. This information will allow users to make more informative searches.

The screenshot shows the 'Create Trait Filter' window. It has a header bar with the title 'Create Trait Filter'. Below the header, there are four main sections: 'Trait', 'Type', 'Operator', and 'Value'. The 'Trait' section contains a dropdown menu with 'DTA\_LT' selected. The 'Type' section has a dropdown menu with 'Single\_Value' selected. The 'Operator' section has a dropdown menu with '> (greater than)' selected. The 'Value' section is an empty text input field. To the right of the 'Value' field is an 'Add' button. Below these sections is a 'Year Range' section with an empty text input field. At the bottom of the window, there are two panes: 'Current Filters' on the left, which is currently empty, and 'Trait Values' on the right, which displays the following statistics for 'DTA\_LT':  
DTA\_LT  
Avg: -24.7108418367347  
StdDev: 1.97570186956945  
Min: -29.1  
Max: -18.1  
No. Obs.: 196  
No. Vines: 49

Next select a search type; the options are “Single\_Value” or “Average”. If single value is selected, any observation that meets the filter requirements will allow a vine to pass the filter. If “Average” is selected, the overall average value for the trait will be taken for each vine, and vines whose average value for the trait meet the filter requirements will pass. The “Average” type should not be selected for a non-numeric trait.

The screenshot shows the 'Create Trait Filter' window with the 'Type' dropdown menu set to 'Average'. All other elements, including the 'Trait' dropdown (DTA\_LT), the 'Operator' dropdown (> (greater than)), the empty 'Value' field, and the 'Year Range' field, remain the same as in the previous screenshot.

Next select an operator for this trait from the dropdown menu. For a numeric trait, you will not want to use the “like” and “not like” operators. These are used for text only.

Create Trait Filter			
Trait	Type	Operator	Value
DTA_LT	Average	< (less than)	
Year Range			
Add			

Optionally, you can enter a year range in the format yyyy-yyyy in the appropriate box. Only values within the year range supplied (ex 2003-2005 will include years: 2003, 2004, and 2005) will be considered for the filter requirements. If this box is left empty, all observations across all years will be considered.

Create Trait Filter			
Trait	Type	Operator	Value
DTA_LT	Average	< (less than)	
Year Range			
2012-2016			
Add			

Finally select a value. You are creating a filter, so only vines that meet the criteria you are creating will be returned. You can use the “trait values” box to better select a cutoff value for your filter.

Create Trait Filter			
Trait	Type	Operator	Value
DTA_LT	Average	< (less than)	-28
Year Range			
2012-2016			
Add			

Finally, click the “Add” button to create the filter. You will see it appear in the box labeled “Current Filters”.

Due to the way in which filters are processed, it is recommended to create equivalent Single\_Value filters for each Average filter (That is, filters that are the same in trait, operator, and value to the average filter but are of type Single\_Value. Year range does not affect this). If you create an Average filter without an equivalent Single\_Value filter, the program will ask if you would like the Single\_Value filter created for you. In most cases, you will select yes.

Trait	Type	Operator	Value	
DTA_LT	Average	< (less than)	-28	Add
Year Range				
2012-2016				

Current Filters					Trait Values
1)	DTA_LT <	-28	Average Value	2012-2016	DTA_LT
					Avg: -24.7108418367347
					StdDev: 1.97570186956945
					Min: -29.1
					Max: -18.1
					No. Obs.: 196
					No. Vines: 49

**NOTICE**

You have created an Average type filter with no analogous Single\_Value filter. It is recommended that one be created. Would you like one created for you?

Yes  No

Selecting yes will create the new filter for you.

Current Filters				
1)	DTA_LT <	-28	Average Value	2012-2016
2)	DTA_LT <	-28	Single Value	

You can repeat this process to add as many filters as you choose. More filters will result in a more selective search, but it will also take more time to look up the traits in the database.

You will notice that each filter is assigned a number in the “Current Filters” box. You can remove a filter by typing this number into the box labeled “Filter number to remove”. Click “Remove Filter” to remove the filter of the entered number. Pressing “Clear All Filters” will remove all filters in the “Current Filters” box.

Current Filters					Trait Values
1)	DTA_LT <	-28	Average Value	2012-2016	DTA_LT
					Avg: -24.7108418367347
					StdDev: 1.97570186956945
					Min: -29.1
					Max: -18.1
					No. Obs.: 196
					No. Vines: 49

Remove Trait Filter

Clear All Filters | Filter number to remove:  | Remove

Search

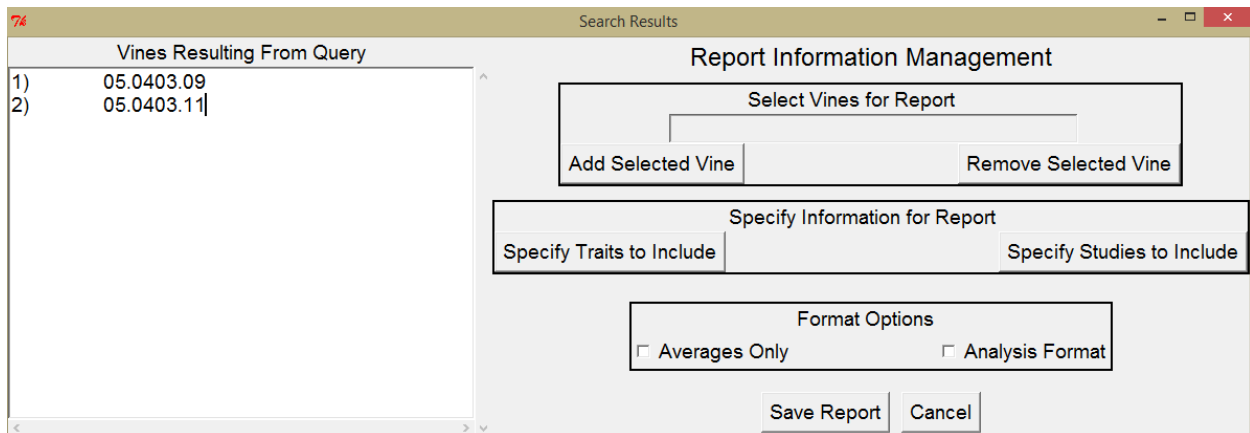
Click “Search” to search for vines that meet the filter criteria specified. Only vines that meet all filters will be returned.

**A Tip for trait year ranges:**

It is a good idea to look at what studies are available when opting to include a year range in a trait filter. If you select a year range for which no observations of a given trait are recorded (for example there are virtually no DTA trials in the early 2000’s), you will not get any vines from your trait search. You can check the studies available in the Text Search window, or by browsing through the studies in the Report Management screen, by clicking on “Select Studies to Include”.

**Search Results Screen and Information Management**

All searches will result in a screen which shows the vines that were returned from the search in a box on the left, as well as other options under “Report Information Management”. The vines in the box on the left are in alphabetic order.



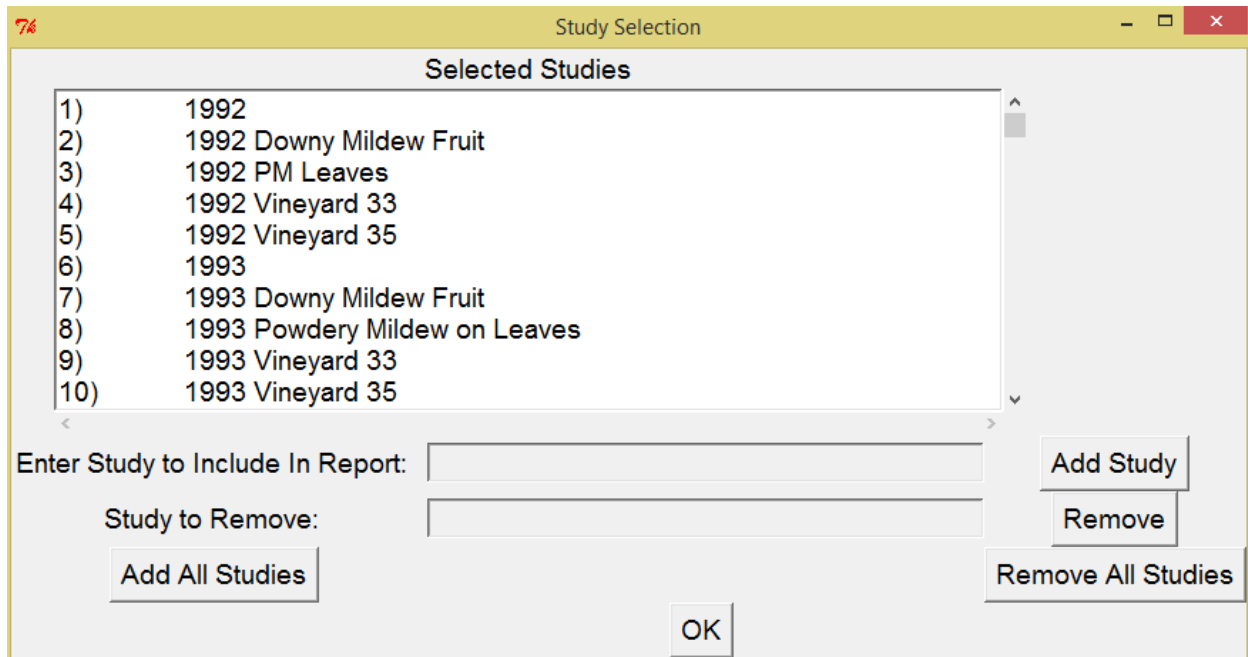
The vines that are displayed in the window on the left are the vines that will be included in the resulting report. To add a vine to this list, select a vine from the dropdown menu in the “Select

Vines for Report” section. Click “Add Vine” and the vine entered will appear in the list on the left. Adding a vine already in the list will have no effect.

To remove vines from the list, you can enter the name of the vine you wish to remove, and click “Remove Vine”. Removing a vine not on the list will have no effect.

You can specify which studies the report will draw information from. To do this, click “Specify Studies to Include” in the “Specify Information for Report” subsection. This will open a new window.

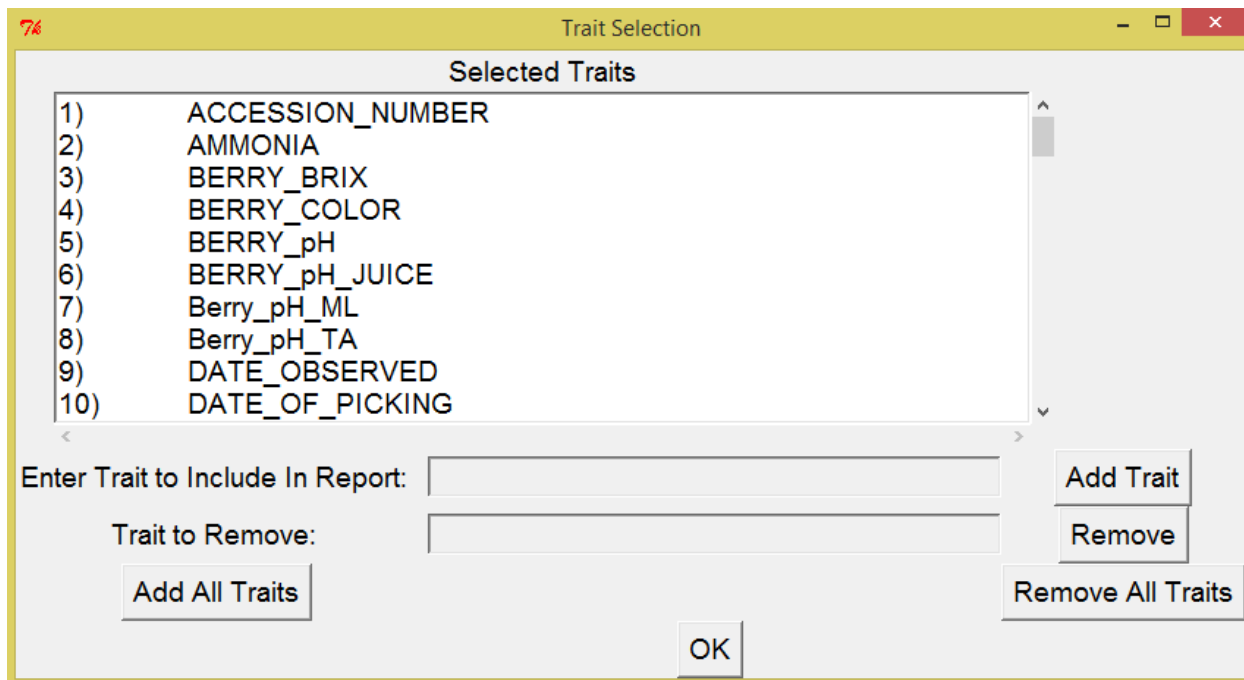




The new window lists all of the studies the report will draw information from. By default, every study is included. You can remove specific studies by selecting a study from the “Study to Remove” box, and pressing the “Remove” button. You can remove all studies by pressing the “Remove all studies” button.

You can add specific studies by selecting a study from the “Enter Study to Include in Report” box, and pressing “Add Study”. You can add all studies to the report by pressing the “Add All Studies” button.

You can specify which traits to include in the report by clicking the “Specify Traits to Include” in the “Specify Information for Report” subsection. Adding and removing traits from the report works in the exact same way that the study selection does.



You can specify the format of the report by toggling the check mark boxes in the “Format Options” section.

Checking neither box will result in the default report style showing all observations for all vines and traits selected for the report.

“Averages Only” will report the average value of every trait for each vine and trait in the report.

“Analysis Format” will format the report so that vine names are on the left column in the resulting spreadsheet, and variable names run across the top. It is recommended to use this format with “Averages Only” checked as well.

If you include only one study in the report, and check only “Analysis Format” you will produce the original spreadsheet used to make the study (with modified headers).

Click “Save” to generate the report. It may take some time to save the report.

### **Creating a report for a vineyard**

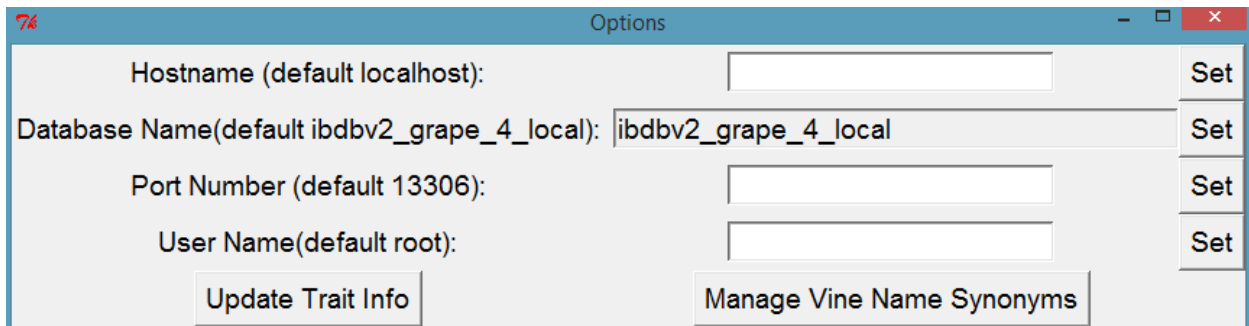
From the Text Search screen, select “Vineyard” search mode. Select the name of a study or vineyard you would like a report for a press “Search”. The result of the search will be all of the

vines in that vineyard or study. Next, click the “Specify Studies for Report” button. Click “Remove All Studies”, and then add only the study you want the report for. This will include only data from that study. Click “OK”, then save the report.

## Options Menu

Click the “Options” button to access the options menu. This button is located on the “Text Search” or “Trait Search” screens. The first four rows of information allow you to set various connection parameters to the database. These should be set up by default. If you want to change them, enter a new value into the boxes, and click the corresponding “Set” button to save the information for the current session. A dialogue box will appear asking you if you want to save the new setting. Clicking “Save” will save the new setting for future sessions.

You can click the “Update Trait Info” button to update stored information about traits.

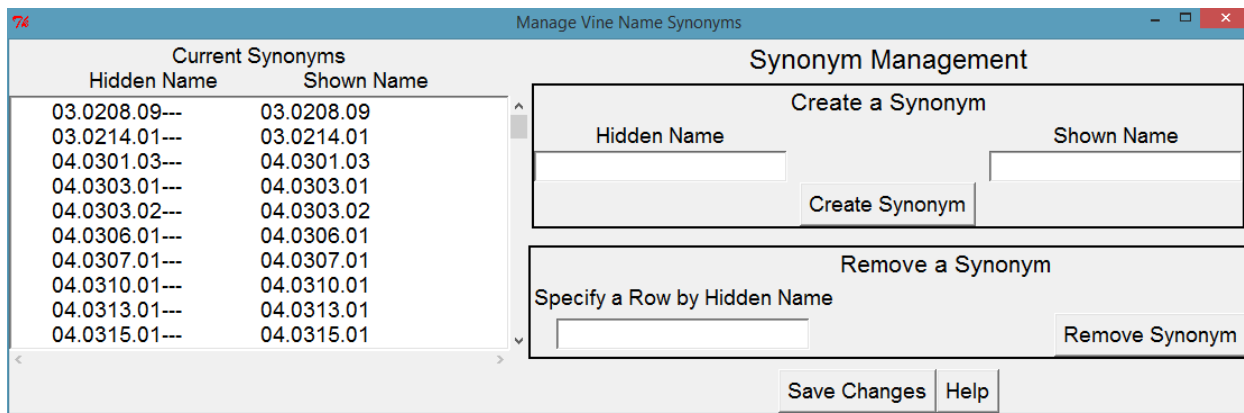


The screenshot shows a window titled "Options" with a blue header bar. Inside the window, there are four rows of configuration options, each with a text input field and a "Set" button to its right:

- Hostname (default localhost): [input field] Set
- Database Name(default ibdbv2\_grape\_4\_local): [input field containing "ibdbv2\_grape\_4\_local"] Set
- Port Number (default 13306): [input field] Set
- User Name(default root): [input field] Set

At the bottom of the window, there are two buttons: "Update Trait Info" and "Manage Vine Name Synonyms".

## Vine Name Synonyms



You can click the “Manage Vine Name Synonyms” button to open a window to manage vine name synonyms.

On the left is a list of current synonyms. Each synonym has two parts, a “Hidden Name” and a “Shown Name”. The Hidden Name is a bad, or old name that you do not want to see again. The Shown Name is the new or corrected name.

To create a synonym, enter a Hidden Name and Shown Name into the boxes in the “Create a Synonym” section. You will notice that most Hidden Names have a “---“ at the end. This indicates to the software to ignore everything after the “---“. Some names have extra information at the end of the name that should be ignored. It is usually best to add a “---“ to the end of the Hidden Name.

Click “Create Synonym” to create the synonym.

To remove a synonym, enter the “Hidden Name” of an existing synonym into the “Specify a Row by Hidden Name” box. Click “Remove Synonym” to remove the synonym pair.

Click “Save Changes” to save the changes you have made here.

## **Appendix II: Search Tool Technical Documentation**

### **Vine search technical documentation**

This document is intended for someone working with the codebase to understand the different paths that the data follows in Vine Search for either debugging, or perhaps adding to the tool.

This document refers to the code inside search.pl.

The document is broken up into the different search modes that the search tool offers, as they are mostly isolated from each other.

### **Section 1: Global Variables:**

The top of the document lists the imports necessary for the tool to run. The directory of the search tool source code is accessed here. The configuration file “search.conf” is read in at the beginning which gives the database connection settings. There is a line which indicates whether this is the first time the search tool has been run on the system. If this is read as “TRUE”, the options menu will open to allow the user to enter appropriate database connection settings.

Global variables that are not graphics components:

my @traitFilters; #An Array that holds all of the trait filters currently active

my %vines; #Holds current set of vines to search for. key is vine name, value is vine GID (from BMS). This is populated by all search types, and used by query functions to know which vines to pull information for.

my @traitsToInclude = getVariableList(); #List of traits to include in report. printed to file later.

The default initialization here is that all traits are included

my @studiesToInclude = getStudyList(); #List of studies to include in report. printed to file later.

The default initialization here is that all studies are included

my @allVinesList = @ {getAllVines()}; #List of all vines in database

my @allVineIDs = @ {getAllIDs()}; #List of all vine GIDs in the database

my @studies = getStudyList(); #List of all studies in database. Separate from @studiesToInclude. This is used by Vineyard Search Mode to populate the dropdown list

my \$currentVineSelected; #Holds the current vine selected from drop down list in the information management window.

my %VineSynonyms; #Hash map which maps {hidden name}->shown name

my %AltLookups; #Hash map which holds {shown name}-> Arraylist of hidden names

Other globals exist, and these mostly track which windows are open, or the values of certain GUI entry components. They are less important to understanding the flow of information in the program.

Finally, the graphics components, and other tracking variables for those components, are initialized.

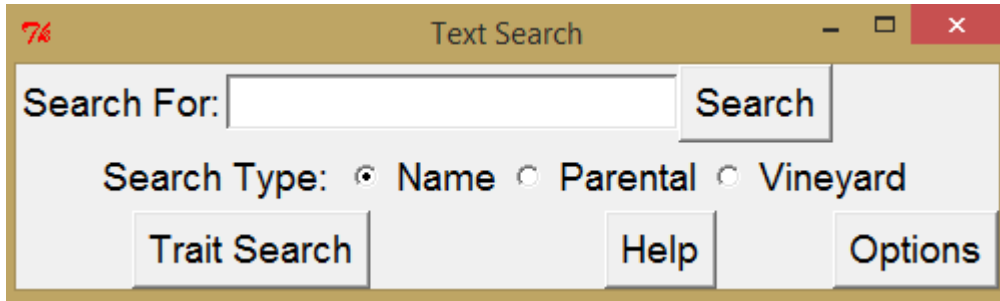
The Method ReadVines() reads in the file vineNames.conf and populates %VineSynonyms and %AltLookups with values from the file.

The Main() method is then called which arranges the Text Search window on the screen. If this is the first time using the tool, a Warning-Window will appear prompting the user to specify database connection settings.

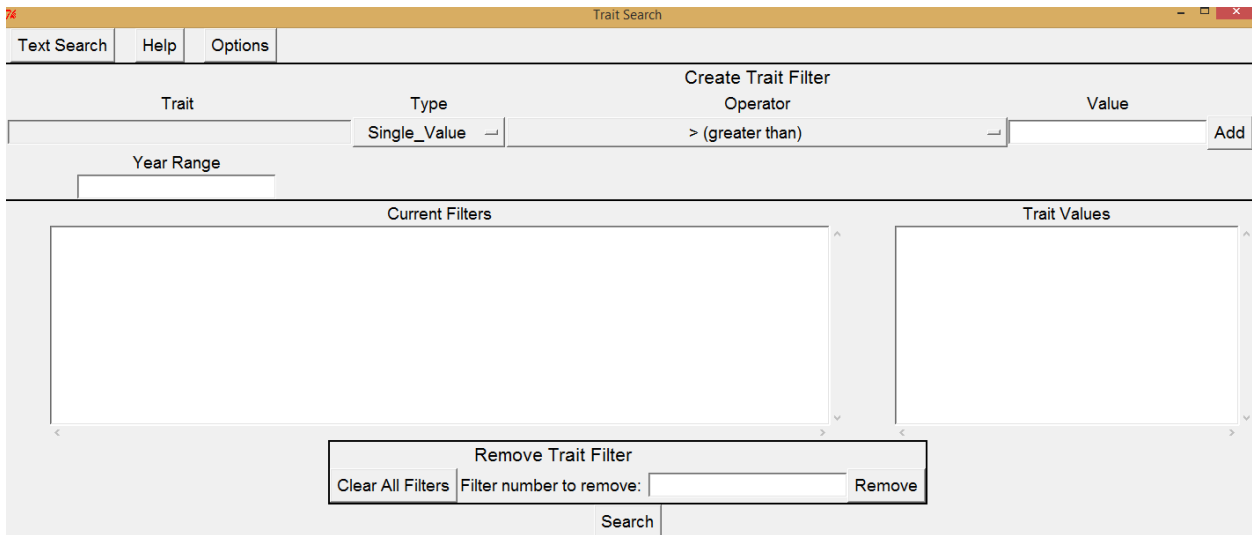
## **Part 2: Main Window Component Management**

The Main Window is either the Text Search box or Trait Search box. Only one can be open at a time.

Text Search:



Trait Search:



The buttons that say “Trait Search” or “Text Search” in each window are used to switch between the two windows.

The button called “my \$trait” on line 121 (as of writing this) handles the transition between the two. The method described for this button checks a Boolean to see which window is open. It then reverses the Boolean, calls a method to destroy either the string frame or trait frame (whichever is currently open), and then calls a method to open the new window. Notice that the Main() method is called to re-open the Text Search window. This is all that the Main() method does.

Note that the radio buttons call the searchBarManager(number) function. Name and parental queries supply a 0, and vineyard search supplies a 1. This function handles the type of search bar

that is present based on what radio button is selected. See line 95 for the radio buttons, and line 305 for the searchBarManager function.

### **Name Search:**

One of the three types of searching available from the Text Search window is to search by the name of the vine. The variable that tracks which radio button is pressed is called \$type on line 76, and is initialized to “name”. The search button on line 119 (\$search) is the search button you can see in the Text Search window. When the name search radio button is selected, the search button when pressed will get the contents of the search bar, and call nameQuery(\$value,1). The type (1) is never used by nameQuery, but it is still there.

nameQuery takes the name of the vine and then generates two strings, one with all periods (04.0506.98), and one with all underscores (04\_0506\_98). These two names are then passed to escapeMySQL() which returns a string in which any MySQL wildcards are removed. A query string is created (see MySQL documentation for more on this). This string is then passed to the function query(). Query handles most of the querying done by the search tool. The official function call here is query(\$query, 2, “”). \$query is the MySQL search string, 2 is the type of search, a search by vine name, and “” is the year specifications if there are any. In this case, none are allowed.

The query() function takes the MySQL search string, and runs the MySQL query using the appropriate database connection settings. The results of the query are returned and arranged into array of array pointers (2d array). The arrays being pointed to contain the values of the columns



returned for each line in the report. Further processing then happens based on what the type supplied to the function is.

Type 2 search indicates a name query. The results of the query are expected to be the name of the vine and its GID in in each row. Here, the name of each vine returned is checked against whether the name of the vine is a hidden name (%VineSynonyms and %altLookups global variables). Names are altered if need be. The names are then sorted alphanumerically, and a string is created with their names. This string is used to make the Information Management Window (see section for this). The vines returned from the query, before processing, are used to populate the %vines (see global variables) hash.

This is where the name query data pipeline ends and is taken up by the Information Management Window.

### **Parental Query**

One of the three types of searching available from the Text Search window is to search by the parents of the vine. The variable that tracks which radio button is pressed is called \$type on line 76, and is initialized to “name”. The search button on line 119 (\$search) is the search button you can see in the Text Search window. When the parental search radio button is selected, the search button when pressed will get the contents of the search bar, and call parentalQuery(\$value).

Parental names are stored in the format parent1/parent2. A user can search by entering the name of either parent (Traminette or 04.0506.07), or the partial name of either parent (04.0506).

Additionally, they could enter the full name of the first parent a “/”and the partial name of the second parent (Traminette/04.0506). They can also enter the names of both parents separated by a “/”(Traminette/04.0506.07).

The `parentalQuery()` function takes the string entered and creates two version of it, one with all underscores, and one with all periods. These names are then passed through a function remove all MySQL wildcard characters. A MySQL query string is then constructed from these name variations (see MySQL documentation for more info on this). The MySQL query string is then passed to the `query()` function. The exact function call is `query ($query, 2, "")`. `$query` is the MySQL search string, 2 is the type of search to be conducted, and `""` is the year specification. In this case, none is allowed.

The `query()` function takes the MySQL search string, and runs the MySQL query using the appropriate database connection settings. The results of the query are returned and arranged into array of array pointers (2d array). The arrays being pointed to contain the values of the columns returned for each line in the report. Further processing then happens based on what the type supplied to the function is.

Type 2 search indicates a parental query. The results of the query are expected to be the name of the vine and its GID in in each row.

The vines returned from the query, before processing, are used to populate the `%vines` (see global variables) hash.

Here, the name of each vine returned is checked against whether the name of the vine is a hidden name (`%VineSynonyms` and `%altLookups` global variables). Names are altered if need be. The names are then sorted alphanumerically, and a string is created with their names. This string is used to make the Information Management Window (see section for this). This is where the parental query data pipeline ends and is taken up by the Information Management Window.

## **Vineyard Search**

Vineyard search is conducted from the Text Search window and allows users to search based on the Trial or Nursery name that the vine appears in in the BMS. The variable \$studyLister on line 107 is the drop-down menu that is created when the vineyard search radio button is selected. The choices are returned from the getStudyList() function, and the current value of the dropdown list is stored in the \$studySearchString variable.

When the search button (line 118, \$search) is pressed, the value of the radio button is checked.

When the vineyard search radio button is selected, the search button calls vineyardQuery(\$studySearchString). This sets into motion the vineyard search pipeline.

The vineyardQuery() function takes the name of the supplied study, and passes it through the MySQLEscape() function to remove any MySQL wildcard characters. A MySQL search string is then constructed from the name of the study (see MySQL documentation for more information).

The method query() is then called to conduct the query. The exact function call is query(\$query, 2, ""). \$query is the MySQL search string, 2 is the search mode, and "" is the year constraints.

Not year constraint is allowed here.

The query() function takes the MySQL search string, and runs the MySQL query using the appropriate database connection settings. The results of the query are returned and arranged into array of array pointers (2d array). The arrays being pointed to contain the values of the columns returned for each line in the report. Further processing then happens based on what the type supplied to the function is.

Type 2 search indicates a vineyard query. The results of the query are expected to be the name of the vine and its GID in in each row.

The vines returned from the query, before processing, are used to populate the %vines (see global variables) hash.

Here, the name of each vine returned is checked against whether the name of the vine is a hidden name (%VineSynonyms and %altLookups global variables). Names are altered if need be. The names are then sorted alphanumerically, and a string is created with their names (one line per name, and the number of the vine in order before each name). This string is used to populate the Information Management Window vine list (see section for this). This is where the vineyard query data pipeline ends and is taken up by the Information Management Window.

## Trait Search

Trait searches are conducted from the Trait Search window, shown again below:

The 5 fields to focus on first are the trait, tpe, operator, value, and year range fields. Each component has a variable assigned to tracking its value, or natively has a method for retrieving the value inside of it.

The frist concept to understand is that of a filter. A filter has 5 parts. The specifies what trait the filter will be focused on. The operator and value determine what criteria are required to pass the filter. The year range specifies what years to draw data from. The type specifies how vines can pass the filter. A single\_value type filter requires that a vine only have one observation of the trait, and within the constraints of the operator and value to pass through the filter. An Average

filter requires that the average value of observation for the specified trait for an individual meet the operator and value constraints to pass.

When a trait is selected from the Trait dropdown box, a method is called (`updateTraitDescription()`) to populate the Trait Values window with information on the trait that was selected.

The user then selects the filter type, the operator and value constraints, and enters a year range if one is desired. A Blank year range results in all data being considered for the filter. Once the filter has been specified, “Add” (`$addButton` on line 172) is clicked, which calls the `addFilter` method. The exact function call is `addFilter($traitString,$operator,$sentValue->get(),$traitType,$sentYear->get())`. `$traitString` is the name of the selected trait, `$operator` is the operator elected from the dropdown list, `$sentValue->get()` gets the value of the Value entry box, `$traitType` is the value of the filter type selected from the dropdown box, and `$sentYear->get()` gets the value of the year range box.

`addFilter()` makes several checks to ensure that the filter is acceptable. First it checks that `$traitString`, `$operator`, and `$sentyear->get()` are all defined.

A check is made to ensure that the operator is an accepted symbol. This check is a holdover from when this was a command-line based search tool. I do not think it is possible to fail at this check due to the dropdown list implementation.

A check is made by calling the `checkVariableName()` function. The exact function call is `checkVariableName($trait)`. The function returns a 1 or a 0 indicating whether the trait name exists in the database or not.

Several checks are then made to ensure that \$entYear-get() is in the proper format of yyyy-yyyy and that the year on the left is <= the year on the right. A proper year format is 2001-2005 or 2002-2002. The year range includes the years specified at the ends of the range.

At any point during the checks if the filter fails, a failure boolean is set to 1, and the function will return at the end without adding the filter to the @traitFilters (See global variables).

If the filter passed all checks, an array of the filter is constructed and added to the @traitFilters array. A string representatio of the filter is also constructed, and the text is set in the text box labeled “Current Filters”.

To remove a filter, the user enters the index number of a filter in the appropriate box and clicks “remove” (\$removeButton, line 179). The function removeFilter(\$removeEnt->get()) is called, at line 379. The filter index is spliced out of the array so long as it looks like a number. A string representation of the filters are then constructed and the terxt box is updated with the new filter string.

A user can also click “Clear Filters” (\$clearButton, line 176) which calls the clearFilters() method on line 418. This empties the @traitFilters array, and sets the text box to be empty. Finally, the user can click “Search” which will conduct the search based on all of the filters (\$searchButton, line 174). The calls the traitQuery() method.

The logic of how the filters are grouped and processed is described here. First, all filters are chekced for the type and year range. If the type is “Average” or a year range has been specified, the filter will be added to a secondary filter group to be processed later. Only filters of type “Single\_Value” and with no year range specified will be initially processed.

The first filters to be processed are then further grouped based on the traits they are filtering for. Each group of initial filters can only have 1 filter for a specific trait. The reason for this is that it

is not possible to determine which of the filters a value passed from the MySQL report; it is not able to be determined if a value passed one or both filters. To begin this grouping process, a first group of filters is initialized and is empty. A hashmap keeps track of the traits that have been added to each group. A filter cannot be added to a group if a filter exists in the same group with the same trait. If this collision occurs, a second group is created to accommodate the duplicate trait filter. Filters are added to the first group as long as no collisions occur, and will be added to the second group if a collision occurs in the first so long as no collision occurs in the second (unlikely). If a collision does occur in the second filter, a third group will be made.

It does not matter which group the initial filters fall into. What is returned from each group is the number of filters each vine passed. A vine must pass all filters in all groups to be included in the final report.

Several check steps occur and lines 1344 and 1354 to ensure that some filters have been entered, and that the groups of filters do not violate the no duplicate trait rule (which should not be possible).

A MySQL search string is constructed for each group of filters. This search string is passed to the query() function. The number of filters passed from each group is recorded. Again, vines must pass all filters in all groups to be in the final report. The official function call is query(\$queryString, 3, ""). \$queryString is the MySQL search string that was constructed for the filter group, 3 is the type of processing, and "" is the year range, which is not allowed here.

The query() function takes the MySQL search string, and runs the MySQL query using the appropriate database connection settings. The results of the query are returned and arranged into array of array pointers (2d array). The arrays being pointed to contain the values of the columns

returned for each line in the report. Further processing then happens based on what the type supplied to the function is.

Type 3 processing indicates a trait query. The year range is not specified so all years are accounted for (actually years 0 through the current year). The trait query returns the name of the vine, a placeholder column to identify which filter was passed, and the trial or nursery name. The year is derived from the beginning of the trial or nursery name. It is expected that the year of the study is found at the beginning of the study name like “2015 DTA Trial”. The year from the study is extracted, and if it falls within the range of the filters supplied (in this case it must), then the count of filters passed for that vine is incremented. This processing occurs for all rows in the MySQL report.

Finally, a hash is returned to the traitQuery() function. The hash maps the name of the vine (key) to the number of filters that the vine passed from that filter group (value). traitQuery takes the returned hash and extracts all vines that passed the total number of filters in that group. Those vines are added to an array to keep track of later, @returnedVines. .

The above process repeats for all filter groups. After this has occurred, the array @returnedVines is processed. The appearances of each vine are counted in that array, and the total must be equal to the number of filter groups that were processed. Vines that pass this check pass the initial level of filter processing, and are now filtered through the secondary filters (with Average or year ranges).

The reason that secondary filters are processed later is that they take significantly longer to run the MySQL queries. I therefore choose to only run them on the vines that pass the initial filtering, rather than the thousands of vines that exist in the database.



To begin, the vines that passed the initial filtering are added to the @vinesThatPassed array. If there were no initial filters, the vines that passed array is populated with all of the vines in the database.

The following processing occurs for every secondary filter, and every vine in @vinesThatPassed.

A query is constructed from the secondary filter. This query is passed to the query() method.

The exact function call is query(\$secondaryQueryString,4,\$secondaryFilter[4]);

\$secondaryQueryString is the MySQL query string, 4 is the search type, and \$secondaryFilter[4] is the year range to process the vines in.

Query processing type 4 is processed the same as type 3, except that the raw values for each vine are returned in an array to traitQuery(). If the filter requires an average value to pass the constraints, then the average of all values is calculated for the vine, and a check is made based on the operator in the filter. If the vine passes this check, a counter is incremented in a hashmap which maps the vine name (key) to the number of secondary filters it passed (value).

If the type of the secondary filter is single value, any value in the array that passes the operator and value constraint will cause the incrementation of the secondary filter (a boolean is set in this case to avoid multiple passing values causing multiple counter increments for the same filter).

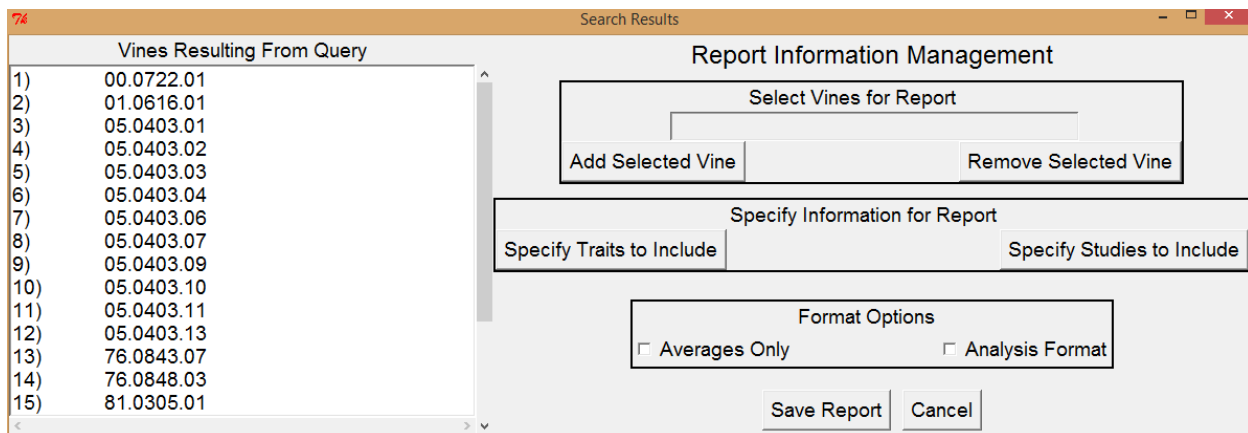
Finally, regular expression checks are used if the values are judged to be text based.

If the counter in the hashmap for each vine is the same as the number of secondary filters, it has passed all filters the user applied.

A (now redundant) check is made to ensure that a hidden name is not present in the list of passed vines. The list of vines is converted into a string for the information management window. The passing vines are added to the %vines hash. Trait filter processing ends here, and further control is given to the information management window.

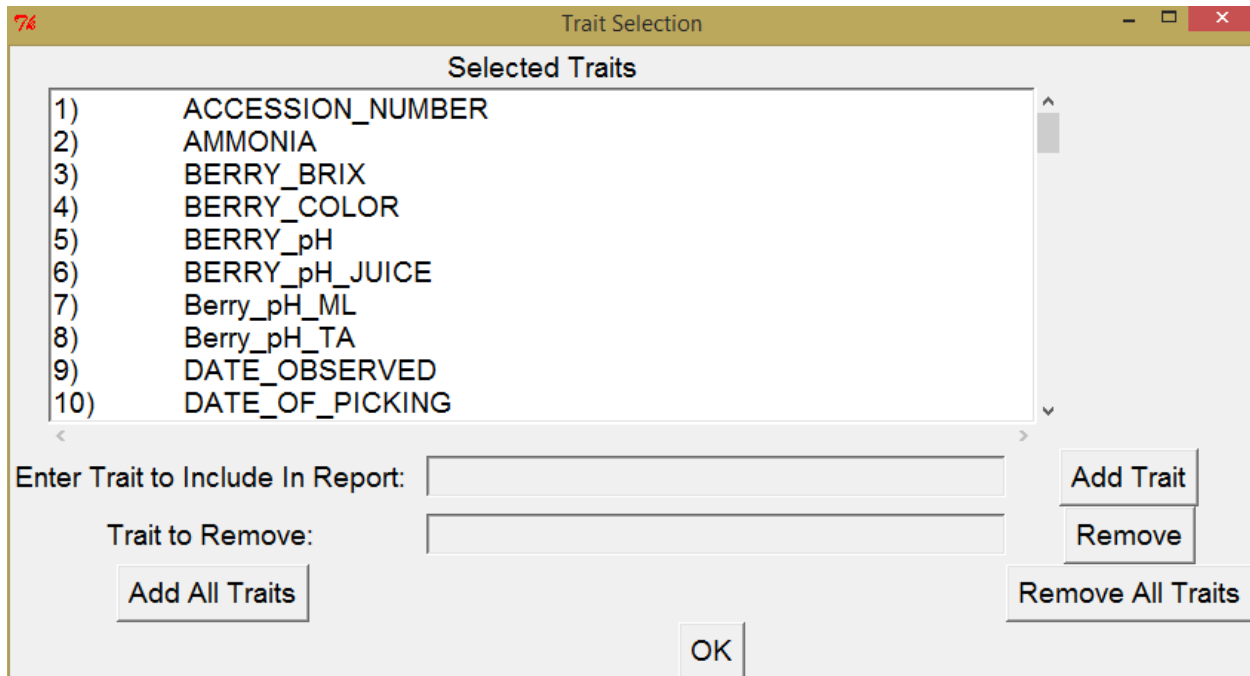
## Information Management Window

This is the image of the Information Management Window:



The vines that passed the resulting query (name, parental, vineyard, or trait) are displayed in the box on the left. The information management window is called at the end of all query functions, and is supplied the list of vines resulting from the query. The definition starts on line 1089. The box labeled “Select Vines for Report” allows a user to enter a vine name to a fill-in box. They can then add or remove the selected vine using the \$addVineButton or \$removeVineButton on lines 1131 and 1132 respectively. Adding a vine simply adds the currently selected vine to the %vine hash, and calls a method to update the window on the left. The generateVineString() (line 2573) method takes the list of vines in %vines and generates the string you see on the left text box. The most important processing is removing vines from the list that are hidden names. The method then sorts the vines alphanumerically, and returns the string which it generated.

Next a user can click the “Specify Traits to Include” (\$specifyTraitButton, line 1126) . This opens a window managed by the makeTraitSelectWindow() function (line 555). An image of the trait select window is shown below:



By default all traits in the database are included. Traits that are selected for inclusion will be included in the final report. This feature allows users to include or exclude specific traits to generate reports of their choosing.

A user can enter a trait into either of the two drop-down list boxes. If a user enters a trait into the Add trait box and clicks “Add” (\$addButton, line 594), the addATrait() method is called (line 594). The specific function call is addATrait(\$currentTrait). \$currentTrait keeps track of the currently selected trait in the Add Trait box (\$traitEnt, line 576). addATrait() checks to make sure \$currentTrait is a valid trait in the database, and adds it to @traitsToInclude (see global variables). It then generates the string shown in the upper box. \$addButton then adjusts the options in the Remove Trait dropdown box to reflect the traits that are in the upper box.

If a user enters a trait into the Remove Trait box and clicks “Remove” (\$removeIndexButton, line 602), the removeATrait() method is called (line 628). The specific function call is removeATrait(\$removeTrait). \$removeTrait keeps track of the currently selected trait in the Remove Trait box (\$removeEnt, line 567). removeATrait() removes all instances of

\$removeTrait from @traitsToInclude (see global variables). It then generates the string shown in the upper box. \$removeButton then adjusts the options in the Remove Trait dropdown box to reflect the traits that are in the upper box.

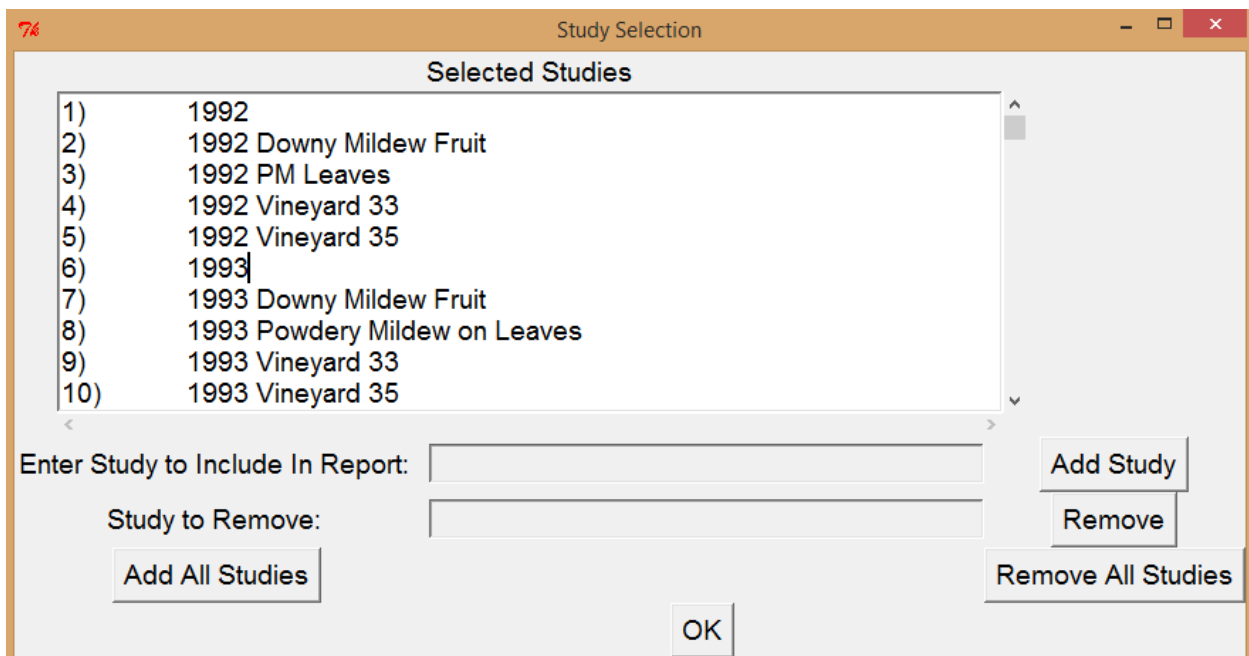
A user can then click the “Add All Traits” button (\$addAllButton, line 596). This calls the addAllTraits() method on line 667. This adds all of the traits in the database to @traitsToInclude.

A string is generated for the upper box, and \$addAllButton updates the upper box, and updates \$removeEnt to reflect the traits that are in the upper box.

A user can also click “Remove All Traits” (\$removeAllButton, line 598). This will set @traitsToInclude to an empty array, set the text box to be empty, and remove all choices from \$removeEnt.

Clicking “ok” will close the window.

A user can click the “Specify Studies to Include” (\$specifyStudytButton, line 1127) . This opens a window managed by the makeStudySelectWindow() function (line 694). An image of the study select window is shown below:



By default all studies in the database are included. Studies that are selected for inclusion will be included in the final report. This feature allows users to include or exclude specific studies to generate reports of their choosing. Items in the list that include only a year are actually folder names in the BMS. Selecting these folders for inclusion will result in all studies which begin with that year to be selected.

A user can enter a study into either of the two drop-down list boxes. If a user enters a study into the Add Study box and clicks “Add” (`$addButton`, line 731), the `addAStudy()` method is called (line 775). The specific function call is `addAStudy($currentStudy)`. `$currentStudy` keeps track of the currently selected study in the Add Study box (`$studyEnt`, line 714). `addAStudy` checks to make sure `$currentStudy` is a valid study in the database, and adds it to `@studiesToInclude` (see global variables). It then generates the string shown the upper box. `$addButton` then adjusts the options in the Remove Study dropdown box to reflect the studies that are in the upper box.

If a user enters a study into the Remove Study box and clicks “Remove” (`$removeIndexButton`, line 736), the `removeAStudy()` method is called (line 760). The specific function call is `removeAStudy($removeStudy)`. `$removeStudy` keeps track of the currently selected study in the Remove Study box (`$removeEnt`, line 704). `removeAStudy()` removes all instances of `$removeStudy` from `@studiesToInclude` (see global variables). It then generates the string shown the upper box. `$removeButton` then adjusts the options in the Remove Study dropdown box to reflect the studies that are in the upper box.

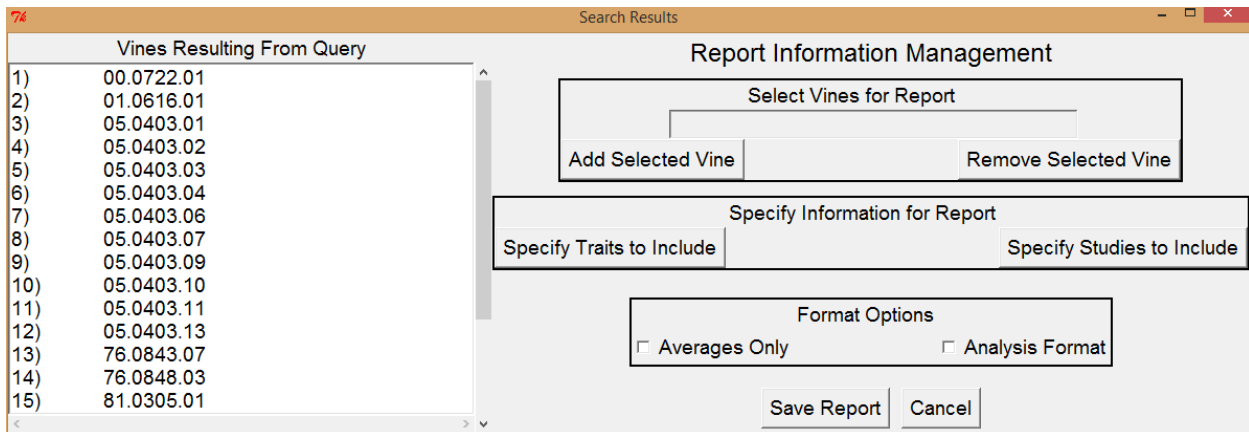
A user can then click the “Add All Studies” button (`$addAllButton`, line 732). This calls the `addAllStudies()` method on line 792. This adds all of the traits in the database to

@studiesToInclude. A string is generated for the upper box, and \$addAllButton updates the upper box, and updates \$removeEnt to reflect the studies that are in the upper box.

A user can also click “Remove All Studies” (\$removeAllButton, line 733). This will set @studiesToInclude to an empty array, set the text box to be empty, and remove all choices from \$removeEnt.

Clicking “ok” will close the window.

Looking back at the original window now:



The format options are checkboxes that allow users to specify the format of the resulting report (\$cb and \$cbAnalysis at lines 1129 and 1130 respectively). These buttons set booleans which track whether the button is on or off. The Averages Only button will take the average value of each trait for each vine in the report. The default is to report every value for each trait. This can

condense the report, and make it more readable. Analysis format will create a simple spreadsheet layout with trait names across the top row, and vine names in the far left column. There will be multiple entries for each vine if this format is used, one for each replication in each study. It is not recommended to use Analysis Format without Averages only unless the user is only including one study in the report.

Save report calls the `traitReport()` function and supplies the name of a temporary file for output. This is where the control of the information management window ends, and the `traitReport()` method begins.

### **Trait Report Method**

This method is called when a user clicks save report in the Information Management Window. The method first iterates through the names of all of the vines in the `%vines` hash (see global variables) and determines if the vine is a hidden name or a shown name. If the vine is a hidden name, I need to add the shown name a list called `@altLookups`, as well as all other hidden names associated with that shown name. If name is a shown name, all hidden names associated with it must be added to `@altLookups`. The logic you see between lines 1823 and 1879 is designed to handle this. The function then calls `vineQuerier()` for each vine in `%vines` and each vine in `@altLookups`. The function call will always be `vineQuerier([name], 1)` or `vineQuerier([name], 2)`.

A 1 designates a search by GID, and a 2 designates a search by vine name.

`vineQuerier()` (line 1962) constructs a MySQL query from the supplied name of the vine (see MySQL documentation for more information) and conducts a MySQL query based on the vine name. Each row returned from the query should contain the vine name, parents, trait, value of observation, trait description, study name, and a unique identifier for the study and observation.

The resulting MySQL report is then written to the file tempoutfileforreport.txt, which is used by a later function.

Once all vines have been queried for, traitReport() prints two alternate files traitfileforreport.txt and studyfileforreport.txt. The files contain the list of traits and studies contained in @studiesToInclude and @traitsToInclude (see global variables). This information is used by later report making functions.

An options string is generated from the Average and Analysis Boolean trackers (see Information Management Window) to specify how the report should be formatted).

The function finally calls reportMaker2.pl, the report formatting function. Below is the function call:

```
$dir/reportMaker2.pl" \"$dir/tempoutfileforreport.txt\" \"$outFile.csv\"  
\"$dir\\traitfileforreport.txt\" \"$dir\\studyfileforreport.txt\" $yearsSince $optionsString.  
$yearsSince is no longer supplied. Instead the studies included serve as a year cutoff.
```

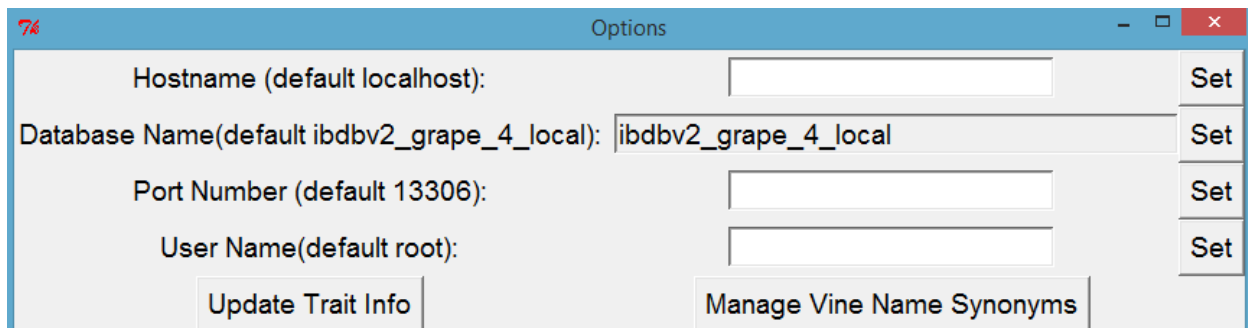
This is the end of control for the Vine Search Window. See the section below for the options menu documentation.

### **Options Menu**

The options menu appears when the user clicks the options button in either the Text Search or Trait Search Windows (\$options, line 106). Clicking this button calls the options\_menu() function on line 864.

The options menu is shown here:





The four fields (hostname, DB name, port number, User Name) allow the user to set specific database connection settings. The default settings for everything except the database name should be sufficient. The Database name can be selected from a drop-down list (\$entDB, line 876) which is populated with a list of databases from the BMS.

Clicking Update Trait Infor (\$refreshTraitDescriptionsButton, line 893) will call updateTraitValueInfo() which calculates all of the trait statistics seen in the Trait Search Window. This takes a considerable amount of time to do. Then getTraitValueInfo() is called by the button which updates this info for the current session.

Setting the values for the above fields sets their value for the current session. A box will then appear asking if the new setting should be saved. Clicking yes will rewrite the vineSearch.conf file (see globals section) with the new database connection settings.

Clicking Manage Vine Name Synonyms (\$openVineNameButton, line 892) will call the makeVineNameWindow() function on line 930 (see Manage Vine Name Synonyms Section).

### **Manage Vine Name Synonyms**

This window is used to manage vine name synonyms. Vine name synonyms specify that two entries have different names, but are the same genotype. This can arise when a variety is released and given a new name, or when mistakes are made in data entry, and the names do not match.

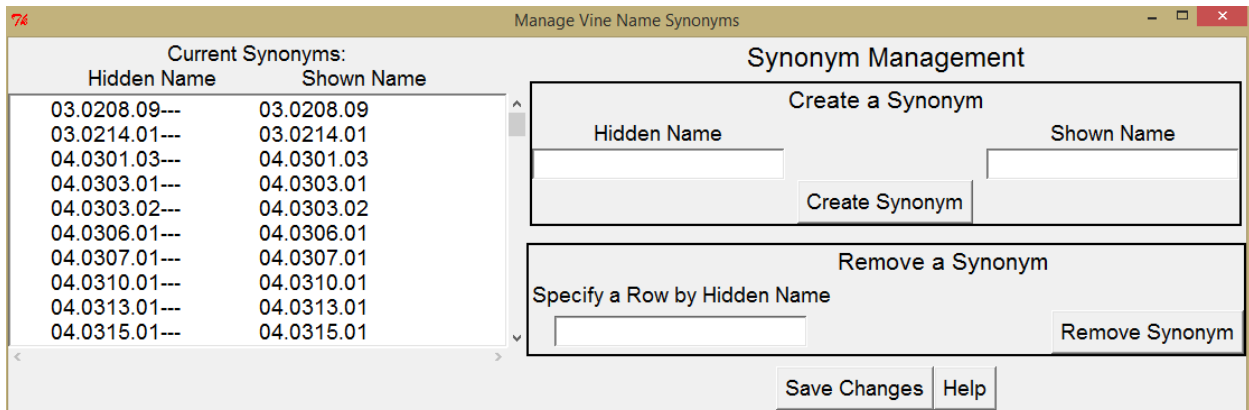
This window allows users to specify these relationships. The hidden name is considered the old

or bad name, it will not be shown in any report. The shown name is the corrected or new name. The hidden name will be replaced by the shown name in all cases.

Things to note about these relationships: the hidden name is unique. It does not make sense to map one hidden name to multiple shown names. Many different hidden names can map to the same shown name though. This is a many to one relationship type.

Secondly, the “---“ following most hidden names is a flag for the program to ignore all other characters. Sometimes extra information is added onto the end of the name of the vine, and this is useful to correct for that. I do not add them by default as there are some cases where this functionality is not desired.

Here is a picture of the Manage Vine Name Synonyms Window:



The box on the left shows current synonyms. Users can create a new synonym by filling in the hidden name field and shown name field and clicking create synonym (\$addButton, line 959). \$addButton calls the addSynonym() function on line 1037. The specific function call is addSynonym(\$hnameEnt->get(),\$snameEnt->get()). \$hnameEnt->get() is the value in the hidden

name field, and `snameEnt->get()` is the value in the shown name field. `addSynonym()` checks to make sure that both fields are defined and that the hidden name has not already been taken. After satisfying these requirements, the relationship will be added to the `%VineSynonyms` hash (see global variables), but not the `$altLookups` hash. A string is generated from the new `%VineSynonyms` hash which is returned to the button. The button then enters this text into the Current Synonyms text box.

Users can remove a synonym relationship in the “Remove A Synonym” area. Users must enter a hidden name into the Hidden Name box, and click “Remove Synonym” (`$removeButton`, line 952). `$removeButton` calls the `removeASynonym` function. The specific function call is `removeSynonym($removeEnt->get())`, where `$removeEnt->get()` is the value in the Hidden Name field. `removeSynonym()` checks to make sure the hidden name is defined, and that it exists in the `%VineSynonyms` hash. Passing these tests, the synonym is removed from the hash. `removeSynonym()` creates a text representation of the new list of synonyms, returns this to the calling button, which then writes the text to the box on the left.

Clicking “Save Changes” (`$quitButton`, line 948) will call the `writeVineConfigFile()` and close the window.

`writeVineConfigFile()` method writes the vine synonym relationships to the `vineNames.conf` file.

This is read in when a new session is opened.

### **Final Output of Vine Search**

The final output of all of the search paths are `dir/tempoutputfileforreport.txt`, `$dir\traitfileforreport.txt`, and `$dir\studyfileforreport.txt`. `tempoutputfileforreport.txt` is just the raw output of the MySQL queries that are generated by Vine Search. All data for all vines

included in the report is printed to this file. Filtering of data that is not in included studies or for observations of traits that are not included is performed by the report maker.

The columns expected in tempoutputfileforreport.txt are VINE\_NAME CROSS

VALUE TRAIT TRAIT\_DESCRIPTION STUDY  
RELATED\_IDENTIFIER.

The names should be self-explanatory.

The reportMaker program takes the raw output and formats the report according to specifications given to it. Please see documentation for the report maker (report maker technical documentation).

### **Adding a Search Method**

In order to add a search method, you will have to perform a number of steps:

- 1) Create a GUI interface for the search. One way to approach this may be to add a button to open a new window to your search. From there, you can have buttons, entry fields, or whatever else you need to specify the options to the search. Some GUI components will have built in get() methods which return the current value of the component. Others will require that you specify a variable to store the value of the field in. You can google what is expected for each component.
  - a. Consider the design of the GUI closely. A new user should be able to figure out the basics of how to use your interface without needing to read instructions. If a user has to read instructions to figure out how to use your interface, it is not designed well.
  - b. Take a look at the code for some of the current screens to know how to place the GUI elements where you want them.

- 2) You will have to develop your own MySQL query for the search. You may be able to easily adapt one or more queries that are currently written to your needs. In any case, you'll want to use some parts of the query that extract the names of the vines.
- 3) The query should return a list of vines. Hidden names should be removed from the list, and then the function `makeVineListingWindow()` should be called, supplying the windows with the list of vines as a parameter. This will bring up the screen to add/remove vines, studies, and traits, as well as format and save the report.

### **Expected File Locations**

This section describes where expected files are for the search tool.

`reportMaker2.pl` and `search.pl` are both stored in the main folder. This is where the temporary study, trait, and raw MySQL text output files are stored. `search.conf` (database connection details), `traitDescriptions.conf` (information about traits displayed in trait search window), and `vineNames.conf` (hidden/shown name synonyms) are found in a folder called `Configuration`. `Configuration` is in the same directory as `search.pl`.

### **Appendix III: Report Making Documentation**

This document is intended for someone working with `reportMaker2.pl` to understand how to use the report making function.

`ReportMaker2.pl` is designed to be run by `vineSearch.pl`, but it can be run by hand. Please note that `ReportMaker2.pl` needs input of only the raw MySQL files, the list of studies and traits to include in the resulting reports, and some flags to specify what format to print in. These files are generated by `vineSearch.pl`. If you want to create a new format type, it is recommended to create your own formatting script. This one will be very difficult to add a new feature to, but you may

want to copy some of the logic in the beginning to read the necessary files in, and correct vine name mistakes. These details are provided here.

The inputs for this script can also be generated without vineSearch.pl (please see documentation for MySQL queries). You will have to run MySQL queries by hand, and copy and paste the results into files and feed them into the script. I would only recommend this if vineSearch.pl is unusable for some reason.

### **Preparation Steps**

The preparation steps include reading in the studies file, trait file, and vine name file. The vine name file specifies hidden and shown name relationships (please see vine search technical documentation for more information). Essentially, bad names are corrected to good names in this file, and the information is read in by the script.

Reading in the studies and traits is simple; each line in the files contains one study or trait, and the names of the studies and traits are stored in a hash for easy lookup later. Vine names are stored in a hash where bad name (key) is mapped to good name (value).

Next the script reads in tempoutputfileforreport.txt which is the raw data from the MySQL query.

The expected columns in the file are: VINE\_NAMECROSS          VALUE          TRAIT  
                         TRAIT\_DESCRIPTION          STUDY          RELATED\_IDENTIFIER.

The fields should be tab separated, and each row should contain the information for 1 observation. Rows can unfortunately be split among several lines in the file for whatever reason (I could never figure this out) so the script will read lines until all 7 columns have been populated with information (see lines 104-111). Lines 113 to 126 replace bad names with good names based on hash lookups. Finally lines 128-135 will only accept the row if the study name and trait name is valid for the report.

The representation of the report in the script is a 2d array. The outer array contains pointers.

Each pointer points to an array which represents a row with 7 entries. The array index mapping is as follows:

0: VINE\_NAME

1: CROSS

2: VALUE

3: TRAIT

4: TRAIT\_DESCRIPTION

5: STUDY\_NAME

6: RELATED\_IDENTIFIER

Finally, please experiment with the different formats of reports to see what the report making script will give you.

#### **Appendix IV: Query Documentation**

The purpose of this document is mostly to help another person in my position to understand the MySQL queries I have written.

Outlined here are the 11 queries that my search tool uses.

Each query has a “Use” section, which briefly describes what the query is used for, and a “How to” section. The “How to” section describes which place holders (such as \$vineName) need to be replaced with an actual value to use the query. The query is listed below the How To section.

Some queries are easier to use than others. The queries are organized so that the simpler ones to execute are first, and the most complicated queries are at the end.

Another document (Name will be supplied when I have finished it) that I am preparing describes where to use these queries and how to run them, and refers to this document as a reference for writing queries to the MySQL underlying the BMS.

Finally, please be mindful of retaining the quotation marks and other characters that are in the queries. Only replace the characters that represent the place holders. The dollar signs (\$) mark the place holders, and should be replaced as well.

Query1:

Use: Return list of all variables in the database

How to: Simply use query as is

```
select * from projectprop where type_id = 1043 or type_id = 1048;
```

Query2:

Use: Get list of all studies in the database

How to: Simply use query as is

```
select name from project;
```



Query 3:

Use: Get list of all vines in database

How to: Simply use query as is

```
select desig from LISTDATA group by desig;
```

Query 4:

Use: Get the GID of a vine from its name

How to: Replace \$name with the name of a vine you want to know the name of

```
select dbxref_id from stock where name = '$name';
```

Query 5:

Use: Get the name of a vine from its GID

How to: Replace \$GID with the GID of a vine

```
select name from stock where dbxref_id = $GID;
```

Query 6:

Use: Return information about a study

How to: Replace \$name with the name of a study like 2015 Downy Mildew Leaves

```
select * from project where name = '$name';
```

Query 7:

Use: Get the scale of a variable based on the variable name

How to: Replace \$trait with the variable you want to know the scale of

```
select scale from standard_variable_details where stdvar_name = '$trait';
```

Query 8:

Use: Search for vines based on studies/vineyards they appear in

How to: replace \$studyname with the name or partial name of a study.

"2015 Vineyard 44" will return all vines in that vineyard.

"2015" will return all vines appearing in any study in 2015.

```
select e.name as 'VINE_NAME', e.dbxref_id as 'ID'
      from nd_experiment c
      JOIN nd_experiment_stock d on c.nd_experiment_id = d.nd_experiment_id
      JOIN stock e on d.stock_id = e.stock_id
      JOIN nd_experiment_project f on c.nd_experiment_id = f.nd_experiment_id
      JOIN project g on f.project_id = g.project_id
      where substring(g.name,1,$length) = '$studyname'
      GROUP BY 1
```

;

#### Query 9:

Use: Find all vines with different variations of a name. Used in search tool for finding vines name variations with underscores (\_) or periods (.)

How to: Replace \$vineName and \$vineName2 with different variations of a name. Example:

\$vineName replaced with 07.0504.07 and \$vineName2 replaced with 07\_0504\_07

If there is only one variation or you only desire to search for one name, use the same name for both fields.

```
select e.name as 'VINE_NAME', e.dbxref_id as 'ID'
        FROM STOCK e
        JOIN LISTDATA h ON e.name = h.desig
        JOIN ND_EXPERIMENT_STOCK d ON e.stock_id =
d.stock_id
        JOIN ND_EXPERIMENT_PROJECT f ON d.nd_experiment_id =
f.nd_experiment_id
        JOIN ND_EXPERIMENT_PHENOTYPE b ON d.nd_experiment_id =
b.nd_experiment_id

        WHERE (e.name = '$vineName'
        AND h.desig = '$vineName') or (e.name = '$vineName2'
        AND h.desig = '$vineName2')
```

GROUP BY 1

;

Query 10:

Use: Return all observations for a given vine name

How to: Replace \$vineName with the name of a vine you want to search for.

Alternate Uses:

1) replace "e.name = '\$vineName' AND h.desig = '\$vineName'" with "i.TRAIT = '\$trait'".

\$trait can be any variable. This will return all observations of a given trait.

```
SELECT          e.name          as
'VINE_NAME'    ,
               h.grpname      as 'CROSS'
,
               a.value        as 'VALUE'
,
               i.TRAIT        ,
               i.TRAIT_DESCRIPTION ,
               REPLACE(g.name, '-PLOTDATA',
") as 'STUDY_NAME'    ,
               d.nd_experiment_id  as
'RELATED_IDENTIFIER'
FROM STOCK      e
```

```

                                JOIN LISTDATA          h ON e.name      =
h.desig

                                JOIN ND_EXPERIMENT_STOCK d ON
e.stock_id      = d.stock_id

                                JOIN ND_EXPERIMENT_PROJECT f ON
d.nd_experiment_id = f.nd_experiment_id

                                JOIN ND_EXPERIMENT_PHENOTYPE b ON
d.nd_experiment_id = b.nd_experiment_id

                                JOIN PROJECT          g ON f.project_id      =
g.project_id

                                JOIN PHENOTYPE          a ON b.phenotype_id
= a.phenotype_id

                                JOIN ( SELECT i1.value ,
                                           i2.value      as
'TRAIT'      ,
                                           REPLACE(i3.value,
'&quot;;', ") as 'TRAIT_DESCRIPTION'

                                FROM projectprop i1
                                JOIN projectprop i2 on
i1.project_id = i2.project_id and i1.rank = i2.rank and (i2.type_id = 1043 or i2.type_id = 1048)
                                JOIN projectprop i3 on
i1.project_id = i3.project_id and i1.rank = i3.rank and i3.type_id = 1060

```

```

) i ON a.observable_id =
i.value
WHERE e.name = '$vineName' AND h.desig =
'$vineName'
GROUP BY 1,3,4,5,6,7;

```

Query 11:

Use: Search for vines based on parental names

How to: Replace \$vineName and \$vineName2 with period and underscore versions of a name.

Use the same name for both fields if only one version of the name exists.

Replace \$length1 and \$length2 with the character lengths of \$vineName and \$vineName2 respectively.

The name format for parents is \$parent1/\$parent2. If a single name is entered, all parent groups which start or end with that parent will be selected.

```

select e.name as 'VINE_NAME', e.dbxref_id as 'ID'
FROM STOCK e
JOIN LISTDATA h ON e.name = h.desig
JOIN ND_EXPERIMENT_STOCK d ON e.stock_id = d.stock_id
JOIN ND_EXPERIMENT_PROJECT f ON d.nd_experiment_id =
f.nd_experiment_id

```

```

JOIN ND_EXPERIMENT_PHENOTYPE b ON d.nd_experiment_id =
b.nd_experiment_id

WHERE (substring(h.grpname,1,$length1) = '$vineName' or h.grpname like
'/$vineName')

or (substring(h.grpname,1,$length2) = '$vineName2' or
h.grpname like '/$vineName2')

GROUP BY 1

;

```

Query 12:

**WARNING: VERY DIFFICULT TO WRITE THIS QUERY WITHOUT SOFTWARE HELP**

Use: Return the vines that meet trait filter requirements

How to:

1) Create filter string by replacing \$trait with a trait, \$operator with an operator (examples: =,!=,<,<=,>,>=,like,not like) and \$value with a value for the phrase below.

(I.TRAIT = \$trait and a.value \$operator \$value)

An example is (I.TRAIT = DTA\_LT and a.value < -25)

2) String multiple filters together by separating them with "or" in the following format:

(I.TRAIT = \$trait and a.value \$operator \$value) or (I.TRAIT = \$trait and a.value \$operator \$value) or (I.TRAIT = \$trait and a.value \$operator \$value)

3) Replace \$filterString below with the string of filters you have created above.

4) Replace \$filterNumber with the number of filters statements you have strung together

```

SELECT VINE_NAME FROM
    (SELECT VINE_NAME, COUNT(*) as 'COUNT' FROM
        (SELECT VINE_NAME, CONCAT(VINE_NAME,TRAIT) as GLOB
FROM
    (SELECT
        e.dbxref_id          as 'VINE_NAME',
        h.grpname           as 'CROSS'      ,
        a.value             as 'VALUE'     ,
        i.TRAIT             ,
        i.TRAIT_DESCRIPTION ,
        REPLACE(g.name, '-PLOTDATA', '') as
'STUDY_NAME'      ,
        d.nd_experiment_id  as
'RELATED_IDENTIFIER'
FROM STOCK          e
JOIN LISTDATA       h ON e.name          = h.desig
JOIN ND_EXPERIMENT_STOCK  d ON e.stock_id  =
d.stock_id

```



```

        JOIN ND_EXPERIMENT_PROJECT f ON
d.nd_experiment_id = f.nd_experiment_id
        JOIN ND_EXPERIMENT_PHENOTYPE b ON
d.nd_experiment_id = b.nd_experiment_id
        JOIN PROJECT g ON f.project_id = g.project_id
        JOIN PHENOTYPE a ON b.phenotype_id =
a.phenotype_id
        JOIN ( SELECT i1.value ,
                i2.value as 'TRAIT'
,
                REPLACE(i3.value, '&quot;', ") as
'TRAIT_DESCRIPTION'
                FROM projectprop i1
                JOIN projectprop i2 on i1.project_id =
i2.project_id and i1.rank = i2.rank and (i2.type_id = 1043 or i2.type_id = 1048)
                JOIN projectprop i3 on i1.project_id =
i3.project_id and i1.rank = i3.rank and i3.type_id = 1060
                ) i ON a.observable_id = i.value
        WHERE $filterString
        GROUP BY 1,3,4,5,6,7)
as BIG GROUP BY GLOB)
as SMALL GROUP BY VINE_NAME)
as LAST where COUNT = $filterNumber;

```

## Appendix V: search.pl

```
#!/usr/bin/perl -w
#####IMPORTS#####
use strict;
use Tk;
use DBI;
use Scalar::Util qw(looks_like_number);
use Tk::MatchEntry;
use Cwd;
use File::Spec::Functions qw(rel2abs);
use File::Basename;
use List::Util qw(sum);
use List::Util qw(min);
use List::Util qw(max);
use feature "switch";

#####DATABASE CONNECTION INFORMATION#####
my $dir = dirname(rel2abs($0)); #Directory that this script is actually in
my $database= "ibdbv2_grape_4_local"; #Database name
my $port= "13306"; #Port number
my $host= "localhost"; #host name
my $username= "root"; #Username

#Open the database connection file. Read in the variables to connect to database
#This must be done to access the MySQL database
#There is no password as the MySQL database is not password secured
#Please see Configuration\search.conf for file structure if you are interested.
open(my $CONFIGURATION_FILE, "$dir\\Configuration\\search.conf") or die "Could not locate configuration file $dir\\Configuration\\search.conf: $!\n";
my $configurationCounter = 1; #Keep track of what line I am on
my $firstTimeFlag;

my @traitsToInclude; #List of traits to include in report. printed to file later
my @studiesToInclude; #List of studies to include in report. printed to file later
my @allVinesList ; #List of all vines in database
my @allVineIDs ;
my @traits; #Get the variable list used in the drop down menu when specifying traits

my @studies = getStudyList();
while (eof($CONFIGURATION_FILE)) {
    my $line = readline($CONFIGURATION_FILE);
    chomp $line;
    $line =~ s"/"/g;
    my @splitLine = split("\t", $line);
    if ($configurationCounter == 3) {
        $database = $splitLine[0];
    }
}
```

```

}elsif($configurationCounter == 4){
    $port = $splitLine[0];
}elsif($configurationCounter == 5){
    $host = $splitLine[0];
}elsif($configurationCounter == 6){
    $username = $splitLine[0];
}elsif($configurationCounter==7){
    if ($splitLine[1] eq "TRUE") {
        $firstTimeFlag = 1;
    }else{
        $firstTimeFlag = 0;
    }
}

}
$configurationCounter++;
}
close($CONFIGURATION_FILE);
my @traitFilters; #Holds trait filters
my %vines; #Holds current set of vines to search for. key is vine name, value is vine GID (from BMS)
if ($firstTimeFlag == 1) {
    #do nothing
}else{

#####GLOBS USED BY MANY FUNCTIONS#####
#####

@traitsToInclude = getVariableList(); #List of traits to include in report. printed to file later
@studiesToInclude = getStudyList(); #List of studies to include in report. printed to file later
@allVinesList = @ {getAllVines()}; #List of all vines in database

@allVineIDs = @ {getAlIIDs()};
@studies = getStudyList(); #List of all studies in database
@traits = getVariableList(); #Get the variable list used in the drop down menu when specifying traits

}

my $currentVineSelected; #Holds the current vine selected from drop down list in the information management window
my %VineSynonyms; #Hash map which maps {hidden name}->shown name
my %AltLookups; #Hash map which holds {shown name}-> Arraylist of hidden names
#####

#MAIN WINDOW COMPONENTS FOR TEXT SEARCH MODE

#####

#####GLOBAL VARIABLES FOR MAIN WINDOW HERE#####

```

```

my $type = "name"; #Search type. Radio button change this. Starts as name search
my $traitFrameOpen = 0; #Track if trait frame is open. 0 is no, 1 is yes
my $searchModel = 0; #Indicates search mode for search bar. Name and parental searches are mode 0 (indicates standard entry bar). Mode 1 is for vineyard searches, which will create a dropdown
bar instead
my $analysisBool = 0; #Used to track if "Analysis" checkbox is checked.
my $avgBool = 0; #Track if average box is ticked
my $yearsSince = 0; #Store the year cutoff. Default is zero to be all inclusive. Thus if year is never specified all studies are included
my $studySearchingString = "";
my $straitType = 1; #Search type where 1 = any value, 2 = average

    ##Main Window Components##
my $mw = new MainWindow;
$mw->optionAdd(*font, 'Helvetica 14');
my $frm_name = $mw -> Frame(); #Main frame
my $slab = $frm_name -> Label(-text=>"Search For:");
my $ent = $frm_name -> Entry(); #Entry point for search string in main window
my $frm_type = $mw -> Frame();
my $dummyLabel = $frm_type->Label(); #Label for formatting
my $lbl_type = $frm_type -> Label(-text=>"Search Type: "); #Label next to search bar

    ###Radio Buttons here###
#The following radio button changes the search type to Name search and calls search bar manager with the new bar type
my $rdb_n = $frm_type -> Radiobutton(-text=>"Name",
    -value=>"name", -variable=>$type, -command=>sub{searchBarManager(0)}); #Name search radio button
#The following radio button changes the search type to parental search and calls the search bar manager with the new bar type
my $rdb_p = $frm_type -> Radiobutton(-text=>"Parental",
    -value=>"parental",-variable=>$type, -command=>sub{searchBarManager(0)}); #Parental search radio button
#The following radio button changes the search type to vineyard and calls the search bar manager with the new bar type.
my $rdb_v = $frm_type -> Radiobutton(-text=>"Vineyard",
    -value=>"vineyard",-variable=>$type, -command=>sub{searchBarManager(1)}); #Vineyard search radio button

    ###Other Main Window Components###
my $options = $mw -> Button(-text=>"Options", -command => \&options_menu); # "Options" button to open options menu
my $studyLister = $frm_name->MatchEntry(
    -multimatch=>0,
    -autoshrink=>1,
    -maxheight=>10,
    -ignorecase=>1,
    -variable => \$studySearchingString,
    -state => 'normal',
    -choices => [getStudyList()],
    -width => 35); #create drop down menu for studies

#Call appropriate search function (based on $type which is updated by radio buttons) when search is pressed
my $search = $frm_name -> Button(-text=>"Search", -command => sub{ if($type eq "name"){my $value = $ent->get(); $value =~ s/"/#/g; nameQuery($value,1);} elsif($type eq "parental"){my
$value = $ent->get(); $value =~ s/"/#/g;parentalQuery($value);} elsif($type eq "vineyard"){vineyardQuery($studySearchingString);}});

```

```

#Trait frame button. Calls appropriate functions to manage transition between trait search and string search windows.
my $trait = $mw -> Button(-text=>"Trait Search", -command=>sub{if($traitFrameOpen == 0){makeTraitMenu();$traitFrameOpen = 1;destroyStringFrame();}else{destroyTraitFrame();
$traitFrameOpen = 0;Main();}}); #Open trait frame or main frame

#Opens help menu for main window
my $helpButton = $mw->Button(-text=>"Help", -command=>sub{showMainHelp();}); #Open the help window

#####END MAIN WINDOW COMPONENTS#####

#####TRAIT WINDOW COMPONENTS#####

#GLOBAL VARIABLES FOR TRAIT WINDOW HERE##

my $operator; #holds operator selected from dropdown list
my $traitString = ""; #Holds string of trait in the entry field
my $traitDescribersRef = getTraitValueInfo();
my %traitDescribersHash = %$traitDescribersRef; #Holds the average, std, min, max, etc for each trait in the form of a string. key is trait name, value is string
#my $vineTraitAveragesRef = getTraitAveragesInfo();
#my %vineTraitAveragesHash = %$vineTraitAveragesRef; #Holds the average value of every trait for every vine. key is vine name, value is pointer to hash. The hash being pointed to is
structured as key (trait) and value (average)

#Filter management here
my $filterFrame = $mw->Frame(-highlightbackground=>"black", -highlightthickness=>2);
my $removeFilterFrame = $mw->Frame(-highlightbackground=>"black", -highlightthickness=>2);
my $searchFrame = $mw->Frame();
my $slabTrait = $filterFrame ->Label(-text=>"Trait");
my $slabType = $filterFrame->Label(-text=>"Type");
my $slabOp = $filterFrame ->Label(-text=>"Operator"); #Operator label
my $slabValue = $filterFrame ->Label(-text=>"Value"); #Value label
my $sentValue = $filterFrame->Entry(); #Value entry field
my $sentYear = $filterFrame->Entry();
my $slabYear = $filterFrame->Label(-text=>"Year Range");
my $sentType = $filterFrame->Optionmenu(
    -options => [[Single_Value=>1], [Average=>2]],
    -variable => \$traitType
);
my $jb1 = $filterFrame->MatchEntry(
    -multimatch=>0,
    -autoshrink=>1,
    -maxheight=>10,
    -ignorecase=>1,
    -variable => \$traitString,
    -state => 'normal',
    -choices => [@traits],
    -width => 35,
    -entercmd => sub{updateTraitDescription($traitString);}, #create drop down menu for traits

```

```

        -command => sub{updateTraitDescription($traitString);});
my $filterCreateLabel = $filterFrame->Label(-text=>"Create Trait Filter", -font=> "Helvetica 16");
my $opt = $filterFrame->Optionmenu(
-operators => [['> (greater than)'=>>'], ['< (less than)'=><'], ['<= (less than or equal to)'=><='], ['>= (greater than or equal to)'=>>='], ['= (equal to)'=>='], ['!= (not equal to)'=>!='], ['like (contains
the following phrase)'=>'like'], ['not like (does not contain the following phrase)'=>'not like']],
-variable => \$operator,
-textvariable => \$operator
); #dropdown list for valid operators
my $addButton = $filterFrame->Button(-text=>"Add", -command=> sub{addFilter($traitString,$operator,$sentValue->get(),$traitType, $sentYear->get());}
); #Click button to add trait filter
my $searchButton = $searchFrame -> Button(-text=>"Search", -command=>sub{traitQuery();}); #Execute trait search when search button is clicked
my $removeFilterLabel = $removeFilterFrame ->Label(-text=>"Remove Trait Filter", -font=> "Helvetica 16");
my $clearButton = $removeFilterFrame -> Button(-text=>"Clear All Filters", -command=>sub{clearFilters();}); #Clear filters when this button is clicked
my $removeLab = $removeFilterFrame -> Label(-text=>"Filter number to remove: "); #Remove label
my $removeEnt = $removeFilterFrame ->Entry(); #Remove entry field
my $removeButton = $removeFilterFrame -> Button (-text=>"Remove", -command => sub{removeFilter($removeEnt->get());}); #remove filter with the index stored in the removeEnt field.

#Text box for current filters
my $textArea = $mw -> Frame();
my $txt = $textArea -> Text(-width=>80, -height=>10);
my $srl_y = $textArea -> Scrollbar(-orient=>'v',-command=>[yview => $txt]); #Y axis scrollbar
my $srl_x = $textArea -> Scrollbar(-orient=>'h',-command=>[xview => $txt]); #X axis scrollbar
my $filterLabel = $textArea-> Label(-text=>"Current Filters");
$txt -> configure(-yscrollcommand=>[set', $srl_y],
-xscrollcommand=>[set',$srl_x]); #

##make trait description window
my $describeArea = $mw ->Frame();
my $describe = $describeArea -> Text(-width=>35, -height=>10);
my $describe_y = $describeArea -> Scrollbar(-orient=>'v',-command=>[yview => $txt]); #Y axis scrollbar
my $describe_x = $describeArea -> Scrollbar(-orient=>'h',-command=>[xview => $txt]); #X axis scrollbar
my $describeLabel = $describeArea-> Label(-text=>"Trait Values");
$describe -> configure(-yscrollcommand=>[set', $describe_y],
-xscrollcommand=>[set',$describe_x]);

sub readVineNames{
#Takes: Nothing
#returns: Nothing
#Purpose: Populates %VineSynonyms and %AltLookups hashes with vine name synonym relationships
#Called directly below function. Please see Configuration\vineNames.conf for file information.
%VineSynonyms = ();

```

```

%AltLookups = ();
open(my $VINE_FILE, "$dir\\Configuration\\vineNames.conf") or makeWarningWindow("Could not open $dir\\Configuration\\vineNames.conf");
while (!eof($VINE_FILE)) {
    my $line = readline($VINE_FILE);
    chomp $line;
    my @splitLine = split("=", $line);
    if (scalar @splitLine != 2) { #If there are not two names separated by an "=", there is an error in the file which needs to be corrected
        makeWarningWindow("FORMATTING ERROR IN vineNames.conf");
    }

    $VineSynonyms{$splitLine[0]} = $splitLine[1];
    if (exists($AltLookups{$splitLine[1]})) {
        push(@{$AltLookups{$splitLine[1]}}, $splitLine[0]);
    } else {
        my @tempArray;
        push @tempArray, $splitLine[0];
        $AltLookups{$splitLine[1]} = \@tempArray;
    }
}

close ($VINE_FILE);
}

readVineNames(); #Read in vine names

#delete a temp file that can hang around if program is interrupted in earlier session
open(my $DELETE, ">>", "$dir\\tempoutfileforreport.txt") or die makeWarningWindow("Could not open $dir\\tempoutfileforreport.txt: $!"); #Open the file for appending
close $DELETE;

Main(); #Call method to open main window

#####

#####ALL WINDOWS CREATED WITH FUNCTIONS BELOW#####

sub Main { #Open main window
    $mw->optionAdd(*font, 'Helvetica 14');

    $mw->title("Text Search");
    #Geometry Management -- Place everything where it goes in the main window
    $slab -> grid(-row=>1, -column=>1);
    $sent -> grid(-row=>1, -column=>2);
    $helpButton->configure(-command=>sub{showMainHelp();}); #Not geometry management. Specifying which function to call when helpButton is pushed.
    $frm_name -> grid(-row=>1, -column=>1, -columnspan=>2);
    $lbl_type -> grid(-row=>3, -column=>1);
    $rdb_m -> grid(-row=>3, -column=>2);
    $rdb_f -> grid(-row=>3, -column=>3);
    $rdb_v -> grid(-row=>3, -column=>4);
}

```

```

$frm_type -> grid(-row=>3,-column=>1,-columnspan=>5);
$search -> grid(-row =>1,-column=>3);
$trait ->grid(-row=>6,-column=>1);
$helpButton->grid(-row=>6,-column=>2);
$options -> grid(-row=>6, -column=>3);
#$dataForYear-> grid(-row=>6,-column=>4);
$mw->update;
if ($firstTimeFlag == 1) {
    makeWarningWindow("It looks like this is your first time using this tool.\n Please fill out the MySQL connection settings in the options page to connect to your database." );
    options_menu();
}

MainLoop;
}

#####HELPER FUNCTIONS FOR TRAIT AND MAIN WINDOWS#####
sub destroyTraitFrame{
    #Returns nothing, takes nothing
    #Called when Text search button is pressed from trait search screen. Destroys the trait frame and reopens the Text search screen.

    $traitFrameOpen = 0; #let everyone know the trait frame is not open
    $textArea->gridRemove(); #remove text area from screen
    $describeArea->gridRemove();
    $filterFrame->gridRemove();
    $removeFilterFrame->gridRemove();
    $searchFrame->gridRemove();
    $trait->configure(-text=>"Trait Search" ); #Change button text
    $mw->update; #update main window
}

sub destroyStringFrame{
    #Returns nothing, takes nothing
    #Called when Trait Search button is pressed in TExt Search screen. Destroys the Text earch screen and opens the trait search screen.
    $frm_name->gridRemove(); #Remove name frame (search bar)
    $frm_type->gridRemove(); #Remove type frame (radio buttons)
    $trait->configure(-text=>"Text Search"); #Change name of "Trait Search" button to "Text Search"
    $trait->grid(-row=>1,-column=>1);
    $helpButton->grid(-row=>1,-column=>2);
    $options->grid(-row=>1,-column=>3);
    $traitFrameOpen =1; #Mark that the trait frame is open
    $mw->update;
}

sub searchBarManager{
    #Takes nothing and returns nothing
    #Called when the radio buttons are pressed to change the search bar into the appropriate type

```



```

my ($modeInto) = @_ ;
if ($modeInto == 1) { #Transitioning to vineyard search
    $ent->gridRemove();
    $studyLister -> grid(-row=>1,-column=>2);
    $searchMode = 1;
}else{
    $searchMode = 0;
    $studyLister -> gridRemove();
    $ent->grid(-row=>1, -column=>2);
}
}

#TRAIT FILTER SELECTION WINDOW##
sub makeTraitMenu{
    #Sub to call the trait menu when "Trait Search" is clicked.
    #Place window components in the appropriate locations
    $mw->optionAdd(*font, 'Helvetica 11');

    #Creates the trait search screen. Called when "trait search button is pressed from text search screen
    $mw->title("Trait Search");
    $helpButton->configure(-command=>sub{showTraitHelp();});
    $filterFrame->grid(-row=>3,-column=>1, -columnspan=>3);
    $textArea -> grid(-row=>7,-column=>1,-columnspan=>5, -rowspan=>5);
    $filterCreateLabel->grid(-row=>1, -column=>3);
    #Geometry management -- place everything where it needs to go in the trait search window
    $frm_trait ->grid(-row=>7,-column=>1,-columnspan=>3);
    $describeArea -> grid(-row=> 7, -column=>6, -columnspan=>5, -rowspan=>5);
    $removeFilterFrame->grid(-row=>12,-column=>1,-columnspan=>6);
    $searchFrame->grid(-row=>16,-column=>1, -columnspan=>6);
    $slabTrait->grid(-row=>2,-column=>1);
    $jb1->grid(-row=>3,-column=>1, -columnspan=>1);
    #$dummylabel ->grid(-row=>1,-column=>2);
    $slabOp->grid(-row=>2,-column=>3);
    $slabType->grid(-row=>2,-column=>2);
    $sentType->grid(-row=>3,-column=>2);
    $sopt->grid(-row=>3,-column=>3, -columnspan=>1);
    $slabValue->grid(-row=>2,-column=>4);
    $sentValue->grid(-row=>3,-column=>4);
    $slabYear->grid(-row=>4,-column=>1);
    $sentYear->grid(-row=>5,-column=>1);
    $saddButton->grid(-row=>3,-column=>6);

    $txt -> grid(-row=>1,-column=>1, -columnspan=>3, -rowspan=>2);
    $srl_y -> grid(-row=>1,-column=>4,-sticky=>"ns", -rowspan=>2);
    $srl_x -> grid(-row=>3,-column=>1,-sticky=>"ew", -columnspan=>3);
}

```

```

$describe -> grid(-row=>1, -column=>1, -columnspan=>3, -rowspan=>2);
$describe_y->grid(-row=>1, -column=>4, -sticky=>"ns", -rowspan=>2);
$describe_x->grid(-row=>3, -column=>1, -sticky=>"ew", -columnspan=>3);
$describeLabel->grid(-row=>0, -column => 2);
$filterLabel -> grid (-row => 0, -column=>2);
$searchButton -> grid(-row=>1,-column=>6);
$clearButton -> grid(-row=>2,-column=>1);
$removeFilterLabel->grid(-row=>1,-column=>4);
$removeLab -> grid(-row=>2,-column=>4);
$removeEnt->grid(-row=>2,-column=>5);
$removeButton->grid(-row=>2,-column=>6);
#$dataForYear->grid(-row=>10, -column=>4);
#$fakeLabel->grid(-row=>6,-column=>7);
$mw -> update;

}

```

#####FILTER MANAGEMENT METHODS CALLED BY TRAIT WINODW#####

```

sub removeFilter{
  #Takes: A filter index to remove
  #Retuns: Nothing
  #Use: Removes a filter based on the index provided. Splices out the supplied index (subtract 1 to convert to zero based array).
  my ($index) = @_ ;
  if (looks_like_number($index)){
    splice @traitFilters,$index-1,1; #Remove based on index minus 1
  }

  my $counter = 0;
  my $string = "";
  #Generate string representing current filters and update text field
  foreach my $filter (@traitFilters){#Update list of trait filters in textarea
    $counter++;
    $string = $string."$counter)t";

    my @filterArray = @$filter;
    for(my $counter = 0;$counter < scalar @filterArray; $counter++){
      if ($counter != 3) {
        $string = $string."$filterArray[$counter]t";
      }elseif($counter==3){
        if ($filterArray[$counter] == 1) {
          $string = $string."Single Value)t";
        }else{

```



```

$failBool = 1;
)elseif($yearString ne ""){
    my @years = split("-", $yearString);
    if ((scalar @years)!= 2) {
        makeWarningWindow("Invalid Years Entered. Format is yyyy-yyyy");
        $failBool = 1;
    }
    foreach my $year (@years){
        if ($year !~ /^[d]{4}/) {
            makeWarningWindow("Invalid Years Entered. Format is yyyy-yyyy");
            $failBool = 1;
        }
    }
    if ($years[0] > $years[1]) {
        makeWarningWindow("Invalid Years Entered. First year must be less than second year");
        $failBool = 1;
    }
}

)if($typeNo == 2){
    #if average filter, check if analagous single filter
    my $hasAnalogueBool = 0;
    foreach my $filterRef (@traitFilters){
        my @filter = @$filterRef;

        if ((($trait eq $filter[0]) && ($operator eq $filter[1]) && ($value == $filter[2]) && ($filter[3] == 1)) {
            $hasAnalogueBool = 1;
        }

    }

    if ($hasAnalogueBool != 1) {
        MakeFilterPromptWindow($trait, $operator,$value);
    }

}

)if($failBool== 0){

    #Success
    #create filter and add it to filter list
    my @filter;
    push(@filter,$trait);
    push(@filter,$operator);
    push(@filter,$value);
    push(@filter,$typeNo);
    push(@filter,$yearString);
    push(@traitFilters,\@filter);
    my $string= "";
    my $counter = 0;

```

```

foreach my $filter (@traitFilters){ #Update text area that shows trait filters

    $counter++;
    $string = $string."$counter)t";
    my @filterArray = @$filter;
    for(my $counter = 0;$counter < scalar @filterArray; $counter++){
        if ($counter < 3) {
            $string = $string."$filterArray[$counter]t";
        }elseif($counter == 3){
            if ($filterArray[$counter] == 1) {
                $string = $string."Single Value)t";
            }else{
                $string = $string."Average Value)t";
            }

        }elseif($counter == 4){
            if ($filterArray[$counter] eq "") {
                $string = $string."Average Value)t";
            }else{
                $string = $string.$filterArray[$counter];
            }

        }
    }
    $string = $string."n";
    $txt->delete('0,0','end');
    $txt -> insert('end',"$string");
    $mw->update;
}

}

sub checkVariableName{
    #Takes: Variable name
    #Returns: 1 if variable name exists in database
    #: 0 if variable name does not exist in database
    #Looks up variable name in database. Returns 0 if variable does not exist. Returns 1 if it does.
    my ($name) = @_;
    $name = escapeMySQL($name);
    if ($name eq "") {
        return(0);
    }

    my $query = "select * from projectprop where value = \"$name\"";
    my $dbh = DBI->connect("DBI:mysql:$database:$host:$port", $username, $password, {RaiseError=>0,PrintError=>1}); #Connect to database
    my $sqlQuery;

```

```

if (defined($dbh)) {
    $sqlQuery = $dbh->prepare($query) or die "Can't prepare $query: $dbh->errstr\n"; #Prepare the query
}else{
    makeWarningWindow("Bad Connection Settings. Change in Options Menu" );
    #options_menu();
    return ();
}
my $rv = $sqlQuery->execute or die "can't execute the query: $sqlQuery->errstr"; #Execute the query
my @report = []; #Initialize array for report
while (my @row= $sqlQuery->fetchrow_array()) { #get each row of results
    push (@report,\@row); #add pointer to row array to report array
}
if ((scalar @report)< 2) {
    return(0);
}
return(1);
}

```

####Sub called by trait filter search window

```

sub updateTraitDescription{
    #Takes: Trait name as a string. The same trait displayed in drop down menu of trait selection for filters
    #Returns: Nothing
    #Purpose: Update "Trait Information" text box with appropriate trait information
    my ($trait) = @_ ;
    if (exists($traitDescribersHash{$trait})) {
        my $string = $traitDescribersHash{$trait};
        $describe-> delete('0.0','end');
        $describe -> insert('end',"$trait\n$string");
        $mw ->update;
    }else{
        $describe-> delete('0.0','end');
        $describe -> insert('end',"Trait not recognized\n");
    }
}

```

#####More Windows#####

```

sub makeTraitSelectWindow(){
    #Takes nothing, returns nothing
    #Manages the addition and subtraction of traits to include in the report
    #This window opens when the Specify Studies for Report button is clicked.
    my $TraitSelectWindow = new MainWindow();
    $TraitSelectWindow->optionAdd('*font', 'Helvetica 14');
    $TraitSelectWindow->title("Trait Selection");
}

```

```

my $StraitFrame1 = $TraitSelectWindow->Frame();

my $StraitLab = $StraitFrame1->Label(-text=>"Enter Trait to Include In Report: "); #Label to enter a trait to include

my $ScurrentTrait = ""; #Holds the string of the current trait in the trait entry field

my $SremoveTrait = ""; #Holds current trait in drop down field

my $SremoveEnt = $StraitFrame1->MatchEntry(
    -multimatch=>0,
    -autoshrink=>1,
    -maxheight=>10,
    -ignorecase=>1,
    -variable => \$SremoveTrait,
    -state => 'normal',
    -choices => [@traitsToInclude], #The currently selected traits
    -width => 35);

my $StraitEnt = $StraitFrame1->MatchEntry( #Drop down menu to select traits
    -multimatch=>0,
    -autoshrink=>1,
    -maxheight=>10,
    -ignorecase=>1,
    -variable => \$ScurrentTrait,
    -state => 'normal',
    -choices => [@traits],
    -width => 35); #create drop down menu for traits

my $StxtArea = $TraitSelectWindow->Frame(); #Text area to show currently selected traits

my $StraitFrameLabel=$StxtArea->Label(-text=>"Selected Traits"); #Label above text area

my $Stxts = $StxtArea -> Text(-width=>60, -height=>10); #Text area

my $Srl_y1 = $StxtArea -> Scrollbar(-orient=>'v',-command=>[yview => $Stxts]); #Y axis scrollbar

my $Srl_x1 = $StxtArea -> Scrollbar(-orient=>'h',-command=>[xview => $Stxts]); #X axis scrollbar

my $SfilterLabel = $StxtArea-> Label(-text=>"Current Filters");

$Stxts -> configure(-yscrollcommand=>['set', $Srl_y1],
    -xscrollcommand=>['set',$Srl_x1]);

#Add button gets trait from drop-down list, calls Add Function, gets string from that function, and updates text field to show string

my $SaddButton = $StraitFrame1->Button(-text=>"Add Trait", -command=>sub{ my $NewString = addATrait($ScurrentTrait);$SremoveEnt->configure(-choices=>[@traitsToInclude]);$Stxts->delete(0.0,'end');$Stxts -> insert('end'," $NewString");$TraitSelectWindow->update;});

#Adds all traits in database. Calls addAll function, get string from that function and update text field to hold that text

my $SaddAllButton = $StraitFrame1->Button(-text=>"Add All Traits", -command=>sub{ my $NewString = addAllTraits();$SremoveEnt->configure(-choices=>[@traitsToInclude]);$Stxts->delete(0.0,'end');$Stxts -> insert('end'," $NewString");$TraitSelectWindow->update;});

#Remove all traits from resulting report. Update text field to be blank

my $SremoveAllButton = $StraitFrame1->Button(-text=>"Remove All Traits", -command=>sub{ @traitsToInclude=();$SremoveEnt->configure(-choices=>[@traitsToInclude]);$Stxts->delete(0.0,'end');$TraitSelectWindow->update;});

my $SremoveByIndexLab=$StraitFrame1->Label(-text=>"Trait to Remove: "); #Label to remove traits by name

#Button to remove the entered trait. Calls appropriate function, gets string from that function, and updates the text field to hold that string

my $SremoveIndexButton=$StraitFrame1->Button(-text=>"Remove", -command=>sub{ my $NewString = removeATrait($SremoveTrait);$SremoveEnt->configure(-choices=>[@traitsToInclude]);$Stxts->delete(0.0,'end');$Stxts -> insert('end'," $NewString");$TraitSelectWindow->update;});

#Dummy label for formatting purposes

my $SdummyLab= $StraitFrame1->Label(-text=> "");

```

```

#Click OK Button to close the window
my $okButton = StraitFrame1->Button(-text=>"OK", -command=>sub{$TraitSelectWindow->destroy();});
#Geometry management. Put all the components where they need to go.
$TextArea->grid(-row=>2,-column=>1);
$StraitFrameLabel->grid(-row=>1,-column=>1);
$Stxts -> grid(-row=>2,-column=>1);
$Srl_y1 -> grid(-row=>2,-column=>2,-sticky=>"ns");
$Srl_x1 -> grid(-row=>3,-column=>1,-sticky=>"ew");
$StraitFrame1->grid(-row=>4,-column=>1, -columnspan=>3);
$StraitLab->grid(-row=>4, -column=>1);
$StraitEnt->grid(-row=>4,-column=>2);
$AddButton->grid(-row=>4,-column=>3);
$RemoveByIndexLab->grid(-row=>5,-column=>1);
$RemoveEnt->grid(-row=>5,-column=>2);
$RemoveIndexButton->grid(-row=>5,-column=>3);
$AddAllButton->grid(-row=>6,-column=>1);
$RemoveAllButton->grid(-row=>6,-column=>3);
$DummyLab->grid(-row=>6,-column=>2);
$okButton->grid(-row=>7,-column=>2);
my $textString = getTraits(); #Get selected traits when window is opened
$Stxts->delete('0.0','end'); #Remove text already in text field. Not sure why there would be, but to be safe...
$Stxts -> insert('end',"${textString}"); #Put current string of selected traits in the text box
$TraitSelectWindow->update; #Update the window
sub removeATrait{
    #Takes: Trait to remove
    #Returns: String representing currently selected traits
    #Use: Removes all occurrences of the entered trait from the list of traits to include in the next study
    my ($trait) = @_ ;
    #Generate list of indexes that match the entered trait. Removes all occurrences of the trait from the traits to include list
    my @del_indexes = grep { $traitsToInclude[$_] eq "$trait" } 0..$#traitsToInclude;
    #Sort indexes from largest to smallest for easy splicing
    @del_indexes = sort { $b <=> $a } @del_indexes;
    foreach my $index (@del_indexes){
        splice @traitsToInclude,$index,1;
    }
    my $string = ""; #Holds string representing currently selected traits
    my $counter = 1; #Count the amount of traits we have
    foreach my $trait(@traitsToInclude){ #Build string representing traits
        $string = $string."$counter)($trait\n";
        $counter++;
    }
    return $string;
}
sub addATrait{
    #Takes: A trait to add
    #Returns: A string representing the current list of selected traits
    #Purpose: Add a trait to include in the next report.

```



```

my $Strait = @_;
if (checkVariableName($Strait) == 1) { #if the trait exists, add it
    push(@traitsToInclude,$Strait);
} else { #If this is not a trait in the database, warn the user and leave the function.
    makeWarningWindow("UNKNOWN TRAIT ENTERED: $Strait");
}
my $Sstring = ""; #Holds string representing currently selected traits
my $Counter = 1; #Count the current trait
#Build the string
foreach my $Strait(@traitsToInclude){
    $Sstring = $Sstring.$Counter.$Strait.n";
    $Counter++;
}
return $Sstring;
}
sub addAllTraits{
    #Takes: Nothing
    #Returns: String representing current list of selected traits
    my $Sstring = ""; #Hold string representing currently selected traits
    @traitsToInclude = getVariableList(); #Get all available traits, and set my selected traits to that
    my $Counter = 1; #Count current trait
    foreach my $Strait(@traitsToInclude){ #Build string
        $Sstring = $Sstring.$Counter.$Strait.n";
        $Counter++;
    }
    return $Sstring;
}
sub getTraits{
    #Takes: Nothing
    #Returns: List of currently selected traits
    #Purpose: Generate initial string for text window
    #Notes: Could call this from above functions. W/e its simple
    my $Sstring = "";
    my $Counter = 1;
    foreach my $Strait (@traitsToInclude){
        $Sstring = $Sstring.$Counter.$Strait.n";
        $Counter++;
    }
    return $Sstring;
}
}

sub makeStudySelectWindow(){
    #Takes nothing, returns nothing.
    #Makes a window to manage the current studies to include in the resulting report.
    #NOTE: THIS IS ALMOST EXACTLY THE SAME AS THE ABOVE FUNCTION makeTraitSelectWindow. I'm not going to comment all of the same stuff again just because it does study
    stuff now instead of trait stuff.
}

```

```

my $studySelectWindow = new MainWindow();
$studySelectWindow->optionAdd(*font', 'Helvetica 14');
$studySelectWindow->title("Study Selection");
my $studyFrame1 = $studySelectWindow->Frame();
my $straitLab = $studyFrame1->Label(-text=>"Enter Study to Include In Report: ");
my $removeStudy = "";
my $removeEnt = $studyFrame1->MatchEntry(
    -multimatch=>0,
    -autoshrink=>1,
    -maxheight=>10,
    -ignorecase=>1,
    -variable => \$removeStudy,
    -state => 'normal',
    -choices => [@studiesToInclude],
    -width => 35);
my $currentStudy = "";
my $studyEnt = $studyFrame1->MatchEntry(
    -multimatch=>0,
    -autoshrink=>1,
    -maxheight=>10,
    -ignorecase=>1,
    -variable => \$currentStudy,
    -state => 'normal',
    -choices => [@studies],
    -width => 35); #create drop down menu for traits
my $textArea = $studySelectWindow->Frame();
my $studyFrameLabel=$textArea->Label(-text=>"Selected Studies");
my $txts = $textArea -> Text(-width=>60, -height=>10);
my $srL_y1 = $textArea -> Scrollbar(-orient=>'v',-command=>[yview => $txts]); #Y axis scrollbar
my $srL_x1 = $textArea -> Scrollbar(-orient=>'h',-command=>[xview => $txts]); #X axis scrollbar
my $filterLabel = $textArea-> Label(-text=>"Current Filters");
$txts -> configure(-yscrollcommand=>['set', $srL_y1],
    -xscrollcommand=>['set',$srL_x1]);
my $addButton = $studyFrame1->Button(-text=>"Add Study", -command=>sub{ my $newString = addAStudy($currentStudy);$removeEnt->configure(-choices=>[@studiesToInclude]);$txts-
>delete('0.0','end');$txts -> insert('end'," $newString");$studySelectWindow->update;});
my $addAllButton = $studyFrame1->Button(-text=>"Add All Studies", -command=>sub{ my $newString = addAllStudies();$removeEnt->configure(-choices=>[@studiesToInclude]);$txts-
>delete('0.0','end');$txts -> insert('end'," $newString");$studySelectWindow->update;});
my $removeAllButton = $studyFrame1->Button(-text=>"Remove All Studies", -command=>sub{ @studiesToInclude=();$removeEnt->configure(-choices=>[@studiesToInclude]);$txts-
>delete('0.0','end');$studySelectWindow->update;});
my $removeByIndexLab=$studyFrame1->Label(-text=>"Study to Remove: ");

my $removeIndexButton=$studyFrame1->Button(-text=>"Remove", -command=>sub{ my $newString = removeAStudy($removeStudy);$removeEnt->configure(-
choices=>[@studiesToInclude]);$txts->delete('0.0','end');$txts -> insert('end'," $newString");$studySelectWindow->update;});
my $dummyLab= $studyFrame1->Label(-text=>"");
my $okButton = $studyFrame1->Button(-text=>"OK", -command=>sub{ $studySelectWindow->destroy();});

$textArea->grid(-row=>2,-column=>1);

```

```

$studyFrameLabel->grid(-row=>1,-column=>1);
$txts -> grid(-row=>2,-column=>1);
$srly1 -> grid(-row=>2,-column=>2,-sticky=>"ns");
$srly1 -> grid(-row=>3,-column=>1,-sticky=>"ew");
$studyFrame1->grid(-row=>4,-column=>1, -columnspan=>3);
$traitLab->grid(-row=>4, -column=>1);
$studyEnt->grid(-row=>4,-column=>2);
$addButton->grid(-row=>4,-column=>3);
$removeByIndexLab->grid(-row=>5,-column=>1);
$removeEnt->grid(-row=>5,-column=>2);
$removeIndexButton->grid(-row=>5,-column=>3);
$addAllButton->grid(-row=>6,-column=>1);
$removeAllButton->grid(-row=>6,-column=>3);
$dummyLab->grid(-row=>6,-column=>2);
$okButton->grid(-row=>7,-column=>2);
my $textString = getStudies();
$txts->delete('0.0','end');
$txts -> insert('end',$textString);
$studySelectWindow->update;
sub removeAStudy{
    my ($study) = @_;
    my @del_indexes = grep { $studiesToInclude[$_] eq "$study" } 0..$#studiesToInclude;
    @del_indexes = sort { $b <> $a } @del_indexes;
    foreach my $index (@del_indexes){
        splice @studiesToInclude,$index,1;
    }
    my $string = "";
    my $counter = 1;
    foreach my $study(@studiesToInclude){
        $string = $string."$counter)\t$study\n";
        $counter++;
    }
    return $string;
}
sub addAStudy{
    my ($study) = @_;
    if (checkStudyName($study) == 1) {
        push(@studiesToInclude,$study);
    }else{
        makeWarningWindow("UNKNOWN STUDY ENTERED: $study");
    }
}

my $string = "";
my $counter = 1;
foreach my $study(@studiesToInclude){
    $string = $string."$counter)\t$study\n";
}

```

```

    $counter++;
}
return $string;
}
sub addAllStudies{
    my $string = "";
    @studiesToInclude = getStudyList();
    my $counter = 1;
    foreach my $study(@studiesToInclude){
        $string = $string.$counter.$study"\n";
        $counter++;
    }
    return $string;
}
sub getStudies{
    my $string = "";
    my $counter = 1;
    foreach my $study (@studiesToInclude){
        $string = $string.$counter.$study"\n";
        $counter++;
    }
    return $string;
}
}

sub makeYearWindow{
    #Called when year cutoff button is pushed. Simple window to enter a year
    #This window is currently not in service, but it may be put back in, who know. so I keep it
    my $yearWindow = new MainWindow;
    $yearWindow->optionAdd(*font, 'Helvetica 14');
    $yearWindow->title("Specify Years");
    my $yearFrm = $yearWindow ->Frame();
    my $yearLab = $yearFrm -> Label(-text=>"Enter oldest year for data return: "); #Label next to the entry field
    my $yearEnt = $yearFrm -> Entry(); #entry field
    #yesButton checks to make sure that the entered year is valid, then sets the global yearsSince to the value.
    #It also changes the function for the specifyYear button that opened this window to be able to flip back and forth between setting a year and returning the year cutoff global to zero.
    #my $yesButton = $yearFrm -> Button(-text=>"OK", -command=>sub{ $yearsSince = $yearEnt->get(); if($yearsSince !~/^[0-9]{4}$/){makeWarningWindow("TEXT ENTERED NOT
RECOGNIZED AS A YEAR: $yearsSince. PLEASE ENTER A 4 DIGIT YEAR"); return();} $dataForYear->configure(-text=>"Remove Year Cutoff", -command=>sub{ $yearsSince = 0;
$dataForYear->configure(-text=>"Specify Year Cutoff", -command=>sub{makeYearWindow();}); $yearWindow->destroy();});
    my $cancelButton = $yearFrm ->Button(-text=>"Cancel", -command=> sub{ $yearWindow->destroy();});
    #Geometry Managment
    $yearFrm->grid(-row=>1, -column=>1);
    $yearLab -> grid(-row=>1, -column=>1);
    $yearEnt->grid(-row=>1, -column=>2);
    #yesButton->grid(-row=>2,-column=>1);
    $cancelButton->grid(-row=>2, -column=>2);
}
}

```

```

}

sub showMainHelp{ #Show help window when help is clicked from main window

#Initialize window components
my $helpWindow = new MainWindow;
$helpWindow->optionAdd(*font', 'Helvetica 14');
$helpWindow->title("Help");
my $helpFrame = $helpWindow->Frame();
my $helpLabel = $helpFrame ->Label(-text=>"Toggle between searching for vines by their name, parentage, or vineyards/studies in which they appear by using the radio buttons below.
    \nFull or partial vine names can be used for a name or parental search.");
my $quitButton = $helpFrame ->Button(-text=>"OK",-command=>sub{$helpWindow->destroy()});
#Geometry management
$helpFrame->grid(-row=>1,-column=>1,-columnspan=>1);
$helpLabel->grid(-row=>1,-column=>1);
$quitButton->grid(-row=>2,-column=>1);
}

sub showTraitHelp{
#Opens help window from trait search screen. Called when "Help" is pressed in trait search screen
#initialize window components
my $helpWindow = new MainWindow;
$helpWindow->optionAdd(*font', 'Helvetica 14');
$helpWindow ->title("Help");
my $helpFrame = $helpWindow->Frame();
my $helpLabel = $helpFrame ->Label(-text=>"Create filters by specifying a trait, value, operator, and type. For a Single_Value filter, a single observation which meets the trait, operator, and
value requirements will pass the filter.\n For an Average filter, a vine will pass if its average observation for the trait specified passes the operator and value requirements.\n Optionally, a year
range can be supplied. Only values within the range (inclusively) will be considered for the filter.\nA vine must pass all filters to be returned in the results.\nRemove specific or all filters using the
buttons on the bottom of the screen. Press search to conduct the search.");
my $quitButton = $helpFrame ->Button(-text=>"OK",-command=>sub{$helpWindow->destroy()});
#Geometry management
$helpFrame->grid(-row=>1,-column=>1,-columnspan=>1);
$helpLabel->grid(-row=>1,-column=>1);
$quitButton->grid(-row=>2,-column=>1);
}

sub options_menu{
#Opens options window. Called when "Options" is pressed on either screen.
#Initialize window components
my $optionsWindow = new MainWindow;
$optionsWindow->optionAdd(*font', 'Helvetica 14');
$optionsWindow->title("Options");
my $frm_name = $optionsWindow-> Frame();
my $slabHOST = $frm_name -> Label(-text=>"Hostname (default $host):");
my $entHOST = $frm_name -> Entry();
my $setHOST = $frm_name -> Button(-text=>"Set", -command => sub{my $name = $entHOST -> get(); $host = $name; setWindow("Hostname",$name);});
my $slabDB = $frm_name -> Label(-text=>"Database Name(default $database):");
my $databases = getDatabases();
}

```

```

my $entDB = $frm_name->MatchEntry(
    -multimatch=>0,
    -autoshrink=>1,
    -maxheight=>10,
    -ignorecase=>1,
    -variable => \$database,
    -state => 'normal',
    -choices => $databases,
    -width => 35); #create drop down menu for traits
my $setDB = $frm_name -> Button(-text=>"Set", -command =>sub{setWindow("Database",$database)});
my $slabPORT = $frm_name -> Label(-text=>"Port Number (default $port):");
my $entPORT = $frm_name -> Entry();
my $setPORT = $frm_name -> Button(-text=>"Set", -command =>sub{my $name = $entPORT -> get(); $port = $name;setWindow("Port",$name)});
my $slabUSER = $frm_name -> Label(-text=>"User Name(default $username):");
my $entUSER = $frm_name -> Entry();
my $setUSER = $frm_name -> Button(-text=>"Set", -command =>sub{my $name = $entUSER -> get(); $username = $name;setWindow("Username",$name)});
my $openVineNameButton = $frm_name -> Button(-text=>"Manage Vine Name Synonyms", -command=>sub{makeVineNameWindow()}); #Button to open vine synonym manager
my $refreshTraitDescriptionsButton = $frm_name ->Button(-text=>"Update Trait Info", -command=> sub{updateTraitValueInfo();getTraitValueInfo()});

#Geometry management
$frm_name -> grid(-row=>1,-column=>1,-columnspan=>3);
$slabHOST -> grid(-row=>1,-column=>1);
$entHOST -> grid(-row=>1,-column=>2);
$setHOST -> grid(-row=>1,-column=>3);
$slabDB -> grid(-row=>2,-column=>1);
$entDB -> grid(-row=>2,-column=>2);
$setDB -> grid(-row=>2,-column=>3);
$slabPORT -> grid(-row=>3,-column=>1);
$entPORT -> grid(-row=>3,-column=>2);
$setPORT -> grid(-row=>3,-column=>3);
$slabUSER -> grid(-row=>4,-column=>1);
$entUSER -> grid(-row=>4,-column=>2);
$setUSER -> grid(-row=>4,-column=>3);

$openVineNameButton ->grid(-row=>5,-column=>2);
$refreshTraitDescriptionsButton->grid(-row => 5, -column=>1);

#Setters for options
sub setWindow{
    #Opens notification window to tell user a value has been changed from the options menu. Called when a "Set" button is pressed in the options menu.
    @traits = getVariableList();
    @studies = getStudyList();
    $jb1->configure(-choices=>@\@traits);
    my ($styp,$sval) = @_ ;
    my $setWindow = new MainWindow;
    $setWindow->optionAdd(*font, 'Helvetica 14');
    $setWindow->title("NOTICE");
    my $frame = $setWindow-> Frame();
    my $label = $setWindow -> Label(-text=>"$styp set to $sval");

```

```

my $QUIT = $setWindow -> Button(-text=>"Save",-command=>sub{writeConfigFile();$setWindow->destroy;});

$frame -> grid(-row=>1,-column=>1,-columnspan=>1);

$label -> grid(-row=>1,-column=>1);

$QUIT -> grid(-row=>2,-column=>1);

}

}

sub makeVineNameWindow{

#Takes nothing

#Returns nothing

#Vine name window. Synonym manager window

my $vineNameWindow = new MainWindow;

$vineNameWindow->optionAdd(*font, 'Helvetica 14');

$vineNameWindow->title("Manage Vine Name Synonyms");

my $createFrame=$vineNameWindow->Frame(-highlightbackground=>"black", -highlightthickness=>2);

my $removeFrame= $vineNameWindow->Frame(-highlightbackground=>"black", -highlightthickness=>2);

my $nameArea = $vineNameWindow -> Frame();

my $labelFrame = $vineNameWindow->Frame();

my $saveFrame = $vineNameWindow->Frame();

my $nameTxt = $nameArea -> Text(-width=>40, -height=>10); #Text area to contain list of vines resulting from each query

my $srl_yname = $nameArea -> Scrollbar(-orient=>'v',-command=>[yview => $nameTxt]); #Y axis scrollbar

my $srl_xname = $nameArea -> Scrollbar(-orient=>'h',-command=>[xview => $nameTxt]);#X axis scrollbar

$nameTxt -> configure(-yscrollcommand=>['set', $srl_yname],xscrollcommand=>['set',$srl_xname]);

my $tempString = getSynonyms(); #Get the current synonyms

my $label = $nameArea ->Label(-text=>"Current Synonyms\nHidden Name\t Shown Name"); #Currently selected synonyms label

my $quitButton = $saveFrame->Button(-text=>"Save Changes", -command=>sub{ writeVineConfigFile();$vineNameWindow->destroy();}); #Save changes and write to file button

my $removeLab = $removeFrame->Label(-text=>"Specify a Row by Hidden Name"); #Remove name label

my $removeEnt = $removeFrame->Entry(); #Remove entry field

#remove name button. Call function, and update text

my $removeButton = $removeFrame->Button(-text=>"Remove Synonym", -command=>sub{my $newString = removeSynonym($removeEnt->get()); $nameTxt->delete(0.0,'end');$nameTxt
-> insert('end',$newString); $vineNameWindow->update;});

my $removeLabel = $removeFrame->Label(-text=>"Remove a Synonym", -font=>"Helvetica 16");

my $hNameLabel = $createFrame->Label(-text=>"Hidden Name"); #Hidden name label

my $sNameLabel = $createFrame->Label(-text=>"Shown Name"); #Shown name label

my $hNameEnt = $createFrame ->Entry(); #Hidden name entry field

my $sNameEnt = $createFrame ->Entry(); #shown name entry field

#Add the current hidden name-shown name pair button. Call appropriate function, get the resulting string, and update text field

my $addButton = $createFrame ->Button(-text=>"Create Synonym", -command=>sub{my $newString = addSynonym($hNameEnt->get(),$sNameEnt->get()); $nameTxt-
>delete(0.0,'end');$nameTxt -> insert('end',$newString);$vineNameWindow->update;});

#Show big 'ol help message

my $helpButton = $saveFrame ->Button(-text=>"Help", -command=>sub{showSynonymHelp();});

my $bigLabel = $labelFrame->Label(-text=>"Synonym Management", -font =>"Helvetica 18");

my $createLabel = $createFrame->Label(-text=>"Create a Synonym", -font=>"Helvetica 16");

#my $helpCreateLabel = $createFrame->Label(-text=> "<---Same Genotype---Different Name--->");

#Geometry management. Put things where they need to be in the window

$nameArea ->grid(-row=>1,-column=>1, -columnspan=>3, -rowspan=>8);

$labelFrame ->grid(-row=>1, -column=>4, -columnspan=>4);

```

```

$bigLabel ->grid(-row=>1, -column=>1);
$label -> grid(-row=>1,-column=>1);
$createFrame->grid(-row=>2, -column=>4, -columnspan=>4);
$removeFrame->grid(-row=>6,-column=>4,-columnspan=>3);
$saveFrame->grid(-row=>10,-column=>4,-columnspan=>4);
$createLabel->grid(-row=>1, -column=>2);
$nameTxt -> grid(-row=>2,-column=>1);
$srlyname -> grid(-row=>2,-column=>2,-sticky=>"ns");
$srlyname -> grid(-row=>3,-column=>1,-sticky=>"ew");
$shNameLabel -> grid(-row=>2,-column=>1);
$shNameLabel ->grid(-row=>2,-column=>3);
$shNameEnt -> grid(-row=>3,-column=>1);
#$helpCreateLabel->grid(-row=>3,-column=>2);
$nameEnt-> grid(-row=>3,-column=>3);
$addButton -> grid(-row=>4,-column=>2);
$removeLabel ->grid(-row=>1,-column=>2);
$removeLab -> grid(-row=>2,-column=>1);
$removeEnt -> grid(-row=>3,-column=>1);
$removeButton->grid(-row=>3,-column=>3);
$quitButton -> grid(-row=>1,-column=>1);
$helpButton -> grid(-row=>1,-column=>3);

$nameTxt -> insert('end',"$tempString");
$svineNameWindow->update;

sub showSynonymHelp{
    #Takes: Noting
    #Returns nothing
    #Prupose: Show a help window explaining how to do things in the synonym manager window
    my $synonymHelpWindow = new MainWindow;
    $synonymHelpWindow->optionAdd('*font', 'Helvetica 12');
    $synonymHelpWindow->title("Synonym Help");
    my $helplabel = $synonymHelpWindow -> Label(-text=>"Create a synonym by specifying a hidden name and a shown name. A hidden name is the
corrected or new name that you want displayed in reports.

        \nA hidden name can only be used once.
        \nAdding --- to the end of the name tells the program to ignore all characters after the name entered.
        \nWhen a search is conducted a hidden or shown name, all data under both names will be considered in the search, and will be included in the report if
appropriate.

        \nRemove a synonym by specifying the hidden name.");
    my $quitButtonHelp = $synonymHelpWindow -> Button(-text=>"OK", -command=>sub{$synonymHelpWindow->destroy();});
    $helplabel->grid(-row=>1,-column=>1);
    $quitButtonHelp->grid(-row=>2,-column=>1);
}
}

sub removeSynonym{
    #Takes: A hidden name

```



```

#Returns: A string representng current synonyms (I am already sick of typing this word)
#Purpose: Given a hidden name, remove the synonym (ugh) with this hidden name. Hidden names can only appear once in all synonyms, so this is the "key"
my ($name) = @_;
chomp $name;
if (exists($VineSynonyms{$name})) {
    delete $VineSynonyms{$name};
} else { #Warn user they did not succesfully remove anything. Need to regenerate string anyways
    makeWarningWindow("HIDDEN NAME $name not recognized");
}
my $string = ""; #This did not work...
foreach my $hName (sort keys %VineSynonyms){
    $string = $string.printf("%20s %20s\n", "$hName", $VineSynonyms{$hName});
}
return ($string);
}

sub addSynonym{
    #Takes: a Hidden name and shown name
    #Returns: A string representing the current synonyms
    #Purpose: Adds a synonym to the sybonym lists
    my ($hName,$sName) = @_;
    chomp $hName;
    chomp $sName;
    if (!defined($hName)) { #No hidden name entered
        makeWarningWindow("HIDDEN NAME NOT ENTERED");
    } elsif (defined($sName)){ #No shown name entered
        makeWarningWindow("SHOWN NAME NOT ENTERED");
    } elsif (exists($VineSynonyms{$hName})){ #Cannot make two synonyms with same hidden name
        makeWarningWindow("SYNONYM ALREADY EXISTS FOR THIS HIDDEN NAME")
    } else { #Success. Add the synonym
        $VineSynonyms{$hName} = $sName;
    }
    my $string = ""; #Formatting did not work at all
    foreach my $hName (sort keys %VineSynonyms){ #Generate string
        $string = $string.printf("%20s %20s\n", "$hName", $VineSynonyms{$hName});
    }
    return ($string);
}

sub getSynonyms{
    #Takes: Nothing
    #Returns: String representing current synonyms
    #Prupose: Initialze text field based on current synonyms when window is opened
    #Could calll this in above functions instead of doing it every time. I'm dumb.
    my $string = "";
    foreach my $hName (sort keys %VineSynonyms){
        $string = $string.printf("%20s %20s\n", "$hName", $VineSynonyms{$hName});
    }
    return ($string);
}

```

```

}

sub makeWarningWindow{
    #Creates a warning window with a given message. Called from multiple locations whenever an error occurs.
    #Takes a message in the form of a string, and displays the message
    my ($message) = @_;
    #Initialize window components
    my $WarningWindow = new MainWindow;
    $WarningWindow->optionAdd('*font', 'Helvetica 14');
    $WarningWindow->title("NOTICE");
    my $warningFrame = $WarningWindow -> Frame();
    my $labMessage = $warningFrame -> Label(-text=>"$message");
    my $quitButton = $warningFrame -> Button(-text=> "OK", -command => sub{$WarningWindow->destroy;});
    $WarningWindow->update;
    #Geometry management
    $warningFrame -> grid(-row => 1, -column => 1, -columnspan => 3);
    $labMessage -> grid(-row=>1,-column=>1);
    $quitButton -> grid(-row=>3, -column =>1);
}

sub MakeFilterPromptWindow{
    my ($Strait, $operator, $value) = @_;
    my $WarningWindow = new MainWindow;
    $WarningWindow->optionAdd('*font', 'Helvetica 14');
    $WarningWindow->title("NOTICE");
    my $warningFrame = $WarningWindow -> Frame();
    my $labMessage = $warningFrame -> Label(-text=>"You have created an Average type filter with no analogous Single_Value filter.");
    my $labMessage2 = $warningFrame -> Label(-text=>"It is recommended that one be created.");
    my $labMessage3 = $warningFrame -> Label (-text=>"Would you like one created for you?");
    my $yesButton = $warningFrame -> Button(-text=> "Yes", -command => sub{addFilter($strait, $operator, $value, 1, ""); $WarningWindow->destroy;});
    my $noButton = $warningFrame -> Button(-text=> "No", -command => sub{$WarningWindow->destroy;});
    $warningFrame -> grid(-row => 1, -column => 1, -columnspan => 3);
    $labMessage -> grid(-row=>1,-column=>2);
    $labMessage2 -> grid(-row=>2,-column=>2);
    $labMessage3 -> grid(-row=>3,-column=>2);
    $yesButton -> grid(-row=>4, -column =>1);
    $noButton -> grid(-row=>4, -column =>3);
}

sub makeVineListingWindow{
    #Takes: Text representing the vines returned from query. Does not include hidden names/shown names in pair
    #Returns: Nothing
    #Called from every search type now
    #Creates a vine listing window. Shows the results from a trait based search or vineyard based search. $text contains the list of vines to show in the textarea.
    #Save report, and manage certain details of the report here
    my ($text) = @_;
    #initialize window components
    my $vineWindow = new MainWindow;

```

```

$vineWindow->optionAdd(*font', Helvetica 14');
$vineWindow->title("Search Results");
my $vinearea = $vineWindow -> Frame();
my $buttonarea = $vineWindow->Frame(-highlightbackground=>'black', -highlightthickness=>2);

my $sinfoInforArea = $vineWindow->Frame(-highlightbackground=>'black', -highlightthickness=>2);
my $sformatInfoArea = $vineWindow->Frame(-highlightbackground=>'black', -highlightthickness=>2);
my $ssaveInfoArea = $vineWindow -> Frame();
my $slabelArea = $vineWindow ->Frame();
my $vineEnt = $buttonarea->MatchEntry(
    -multimatch=>0,
    -autoshrink=>1,
    -maxheight=>10,
    -ignorecase=>1,
    -variable => \ $currentVineSelected,
    -state => 'normal',
    -choices => [@allVinesList],
    -width => 35);
my $svinetxt = $vinearea -> Text(-width=>40, -height=>15); #Text area holding selected vines
my $srl_y4 = $vinearea -> Scrollbar(-orient=>'v',-command=>[yview => $svinetxt]); #Y axis scroll
my $srl_x4 = $vinearea -> Scrollbar(-orient=>'h',-command=>[xview => $svinetxt]); #X axis scroll bar
$svinetxt -> configure(-yscrollcommand=>['set', $srl_y4],xscrollcommand=>['set',$srl_x4]);
my $slabel = $vinearea ->Label(-text=>"Vines Resulting From Query"); #label above text field
#Save button. Opens neat window and gets filename from that. Call the trait report supplying the filename and close this window
my $ssaveButton = $ssaveInfoArea->Button(-text=>"Save Report", -command=>sub{ my $filename = $vineWindow->getSaveFile( -title => 'Save File:',
-initialdir => '.' );if(defined($filename)){ $filename =~ s/\./ /; traitReport($filename); $vineWindow->destroy()}});
#Just quit and destroy the window
my $squitButton = $ssaveInfoArea->Button(-text=>"Cancel", -command=>sub{ $vineWindow->destroy()});
my $sspecifyTraitButton = $sinfoInforArea->Button(-text=>"Specify Traits to Include", -command=>sub{makeTraitSelectWindow();}); #Button to specify traits included in report
my $sspecifyStudyButton = $sinfoInforArea ->Button(-text=>"Specify Studies to Include", -command=>sub{makeStudySelectWindow();}); #Button to specify the studies included in report
#Check button to specify whether only averages will be returned. Updates $avgBool variable
my $scbAnalysis = $sformatInfoArea->Checkbutton(-text => 'Analysis Format', -onvalue=> 1, -offvalue=>0,-variable=>\$analysisBool); #Check button to make analysis report
my $scb = $sformatInfoArea->Checkbutton(-text => 'Averages Only', -onvalue=> 1, -offvalue=>0,-variable=>\$avgBool); #Poorly named checkbutton to take averages in report
my $saddVineButton = $buttonarea->Button(-text=>"Add Selected Vine", -command=>sub{addVine();$svinetxt->delete(0,0,'end');$svinetxt->insert('end',generateVineString());});
my $sremoveVineButton = $buttonarea->Button(-text=>"Remove Selected Vine", -command=>sub{removeVine();$svinetxt->delete(0,0,'end');$svinetxt->insert('end',generateVineString());});
my $sinformationLabel = $sinfoInforArea->Label(-text=>"Specify Information for Report");
my $svineLabel = $buttonarea-> Label(-text=> "Select Vines for Report");
my $smanageLabel = $slabelArea->Label(-text=>"Report Information Management",-font=>"Helvetica 18");
my $sformatLabel = $sformatInfoArea->Label(-text=>"Format Options");
my $sdummyLabel = $ssaveInfoArea->Label(-text=>" ");
#Geometry management. Put things where they need to be
$vinearea ->grid(-row=>0,-column=>1,-columnspan=>4, -rowspan=>11);
$slabelArea->grid(-row=>0,-column=>5, -columnspan=>4);
$buttonarea->grid(-row=>1, -column=>5, -columnspan=>4);
$sinfoInforArea->grid(-row=>3,-column=>6, -columnspan=>4);
$sformatInfoArea->grid(-row=>7,-column=>6,-columnspan=>4);
$ssaveInfoArea->grid(-row=>10,-column=>6,-columnspan=>4);

```

```

$vinetxt -> grid(-row=>2,-column=>1, , -rowspan =>6);
$srly4 -> grid(-row=>2,-column=>3,-sticky=>"ns", -rowspan=>10);
$srly4 -> grid(-row=>11,-column=>1,-sticky=>"ew");
$manageLabel->grid(-row=>0,-column=>1,-columnspan=>3);
$vineEnt->grid(-row=>2,-column=>1,-columnspan=>3);
$vineLabel->grid(-row=>1,-column=>2);
$addVineButton->grid(-row=>3,-column=>1);
$removeVineButton->grid(-row=>3,-column=>3);
$label->grid(-row=>1,-column=>1);
$formatLabel->grid(-row=>1,-column=>2);
$cb->grid(-row=>2,-column=>1);
$cbAnalysis -> grid(-row=>2, -column=>3);
$saveButton->grid(-row=>10,-column=>1);
$informationLabel->grid(-row=>6,-column=>2);
$specifyTraitButton->grid(-row=>7,-column=>1);
$specifyStudyButton->grid(-row=>7,-column=>3);
$vinetxt ->insert('end',$text);
$quitButton->grid(-row=>10,-column=>3);
$dummyLabel->grid(-row=>10,-column=>2);
}

sub addVine{
    #Adds the current selected vine to the list of vines of search
    #Takes: Nothing
    #Returns: Nothing
    $vines{$currentVineSelected} = 1;
}

sub removeVine{
    #Takes: Nothing
    #Returns: Nothing
    #Purpose: Removes vine from the currently selectedvines list.
    #Checks for any synonyms. Will look to see if current vine is a hidden name, remove the shown name in that case, and all other hidden names of the vine
    #print "Current vine: $currentVineSelected\n";
    foreach my $syn (keys %VineSynonyms){ #checking here to see if $name is a hidden vine name. If it is, replace it with the correct shown name
        if (my $index = (index($syn,"---"))!--1) {
            my $substring = substr($syn, 0,-3);
            if ($currentVineSelected =~ /$substring.*/) {
                foreach my $vine (keys %vines){
                    #print "is $vine equal to $substring\n";
                    if ($vine =~ /$substring.*/) {
                        delete $vines{$vine};
                    }elseif($VineSynonyms{$syn} eq $vine){
                        delete $vines{$vine};
                    }
                }
            }
        }
    }
}

```

```

    if ($currentVineSelected eq $syn) {
        delete $vines{$syn};
    }
}

foreach my $vine (keys %vines){
    if (my $index = (index($syn,"---"))!=1) {
        my $substring = substr($syn, 0,-3);
        if ($vine =~ /$substring.*/) {
            if ($VineSynonyms{$syn} eq $currentVineSelected) {
                delete $vines{$vine};
            }
        }
    }
}

delete $vines{$currentVineSelected};

}

sub checkStudyName{
    #Takes: Name of a study
    #Returns: 1 or 0 denoting existence or not of study in BMS
    #Looks up variable name in database. Returns 0 if variable does not exist. Returns 1 if it does.
    my ($name) = @_;
    # $name = escapeMySQL($name);
    if ($name eq "") {
        return(0);
    }

    my $query = "select * from project where name = \"\$name\"";
    my $dbh = DBI->connect("DBI:mysql:$database:$host:$port", $username, "", {RaiseError=>0,PrintError=>1}); #Connect to database
    my $sqlQuery;
    if (defined($dbh)) {
        $sqlQuery = $dbh->prepare($query) or die "Can't prepare $query: $dbh->errstr\n"; #Prepare the query
    }else{
        makeWarningWindow("Bad Connection Settings. Change in Options Menu");
        #options_menu();
        return ();
    }
    my $rv = $sqlQuery->execute or die "can't execute the query: $sqlQuery->errstr"; #Execute the query
    my @report = []; #Initialize array for report
    while (my @row= $sqlQuery->fetchrow_array()) { #get each row of results
        push (@report,\@row); #add pointer to row array to report array
    }
}

```

```

if ((scalar @report)< 2) {
    return(0);
}
return(1);
}

#####QUERY METHODS#####

sub nameQuery{
    #Takes: Name and a type
    #returns: nothing
    #Conduct a query based on the name of the input vine
    my ($name, $type) = @_;
    #Generate two vine names with underscores and periods
    my $vineName = $name;
    my $stringLength = length $vineName;
    my $vineName2 = $vineName;
    $vineName2 =~ s/\./_/g; #replace periods in vine name with underscores
    $vineName = $vineName2;
    $vineName =~ s/_/\./g;
    $vineName = escapeMySQL($vineName);
    $vineName2 = escapeMySQL($vineName2);
    my $query = "select e.name as 'VINE_NAME', e.dbxref_id as 'ID'
                FROM STOCK e
                JOIN LISTDATA h ON e.name = h.desig
                JOIN ND_EXPERIMENT_STOCK d ON e.stock_id = d.stock_id
                JOIN ND_EXPERIMENT_PROJECT f ON d.nd_experiment_id = f.nd_experiment_id
                JOIN ND_EXPERIMENT_PHENOTYPE b ON d.nd_experiment_id = b.nd_experiment_id

                WHERE (e.name like \"$vineName%\"
                AND h.desig like \"$vineName%\" or (e.name like \"$vineName2%\"
                AND h.desig like \"$vineName2%\") #The second number in the substring field gives the length of the search string
                GROUP BY 1
                ";
    query($query,2,""); #Get report

    #my $rows = scalar @{$reportPtr} -1;
    #stringQueryWindow("$rows rows returned from search for $name"); #display report results to stringQueryWindow
}

sub parentalQuery{
    #Takes: Name of a parent vine
    #Returns: Nothing
    #Purpose: Conduct a parental query based on a name
    my ($name) = @_;

```

```

#Generate two version of the name. One with periods and one wit underscores
my $vineName = $name;
my $stringLength = length $vineName;
my $vineName2 = $vineName;
$vineName2 =~ s/\./_/g; #replace periods in vine name with underscores
$vineName = $vineName2;
$vineName =~ s/_/\./g;
$vineName = escapeMySQL($vineName);
my $length1 = length $vineName;
$vineName2 = escapeMySQL($vineName2);
my $length2 = length $vineName2;
my $query = "select e.name as '\VINE_NAME\', e.dbxref_id as '\ID\'
FROM STOCK e
JOIN LISTDATA h ON e.name = h.desig
JOIN ND_EXPERIMENT_STOCK d ON e.stock_id = d.stock_id
JOIN ND_EXPERIMENT_PROJECT f ON d.nd_experiment_id = f.nd_experiment_id
JOIN ND_EXPERIMENT_PHENOTYPE b ON d.nd_experiment_id = b.nd_experiment_id
WHERE (substring(h.grpname,1,$length1) = \"$vineName\" or h.grpname like \"%/$vineName\") or (substring(h.grpname,1,$length2) = \"$vineName2\" or h.grpname like
\"%/$vineName2\")
GROUP BY 1
;";
query($query,2,""); #get results of query
}

sub traitQuery{
#Takes: Nothing
#Returns: Nothing
#Purpose: Conduct a trait query based on filters
my $counter = 0;
my $groupCounter = 0;
#Generate part of the query based on the filters
my @filterGroups;
my @traitHashes; #Holds hashes containing the traits of each filter group;
my @secondaryFilters; #Holds filters that will be processed after initial filters run. These filters have average and/or year restrictions.
for($counter = 0; $counter < scalar @traitFilters; $counter++){ #Generate compatible groups of filters.
#A group of filters is just a list of filters for unique traits. If a filter is a duplicate of a trait (which may happen for notes or something), it is added to a new group
if ((($traitFilters[$counter]->{3} != 1) || $traitFilters[$counter]->{4} ne "") { #If an average restriction or year restriction exists for the filter
push(@secondaryFilters, $traitFilters[$counter]);
next;
}

my $saddedBool = 0; #decides whether a filter got added to a filter list.
for(my $sinnerCount = 0; $sinnerCount < scalar @filterGroups; $sinnerCount++){ #loop over all available filter groups
if (!exists($traitHashes[$sinnerCount]->{$traitFilters[$counter]->{0}})) {
#Good to go
$saddedBool = 1;
$traitHashes[$sinnerCount]{$traitFilters[$counter]->{0}} = 1; #add trait to my hash
}
}
}

```

```

push($filterGroups[SinnerCount], $traitFilters[$counter]);

}

}

if ($saddedBool == 0) {
    #Must create a new filterGroup and hash
    $groupCounter++;
    my @newFilterGroup;
    push(@newFilterGroup,$traitFilters[$counter]);
    push(@filterGroups,@newFilterGroup);
    my %newTraitHash;
    $newTraitHash{$traitFilters[$counter]->[0]} = 1;
    push(@traitHashes,%newTraitHash);
}

}

my @returnedVines;
if ($counter == 0) {

    makeWarningWindow("NO FILTERS ENTERED"); #Warn
    return;
}

foreach my $group (@filterGroups){
    my $filterString = "(";
    for(my $counter = 0; $counter < scalar @$group; $counter++){
        my %usedTraits; #Hold a hash of trait names already used. Warn user that they have two filters for the same trait.
        #Multiple filters for the same trait are allowed.
        if (exists($usedTraits{ @$group[$counter]->[0]}) ) {
            #This should not be triggered as filters have been split into seperate groups
            makeWarningWindow("MULTIPLE FILTERS FOR SAME TRAIT: @$group[$counter]->[0]");
            return;
        }else{
            $usedTraits{ @$group[$counter]->[0]} = 1;
        }
        #Build query strings based on filters
        if ($counter == (scalar @$group)-1) {
            if (@$group[$counter]->[1] =~ /like/) {
                @$group[$counter]->[2] =~ s/"//g;
                $filterString = $filterString.(I.TRAIT = \"@$group[$counter]->[0]\" and a.value @$group[$counter]->[1] \"%@$group[$counter]->[2]%\");
            }else{
                $filterString = $filterString.(I.TRAIT = \"@$group[$counter]->[0]\" and a.value @$group[$counter]->[1] @$group[$counter]->[2]");
            }
        }else{
            if (@$group[$counter]->[1] =~ /like/) {

```



```

    @$group[$counter]->[2] =~ s//g;
    $filterString = $filterString.(I.TRAIT = \"@$group[$counter]->[0]\" and a.value StraitFilters[$counter]->[1] \"%@$group[$counter]->[2]%\") or";
}
else{
    $filterString = $filterString.(I.TRAIT = \"@$group[$counter]->[0]\" and a.value @$group[$counter]->[1] @$group[$counter]->[2] or");
}
}
}
}

#Conduct query for each group of filters
$filterString = $filterString."";
$filterString = escapeMySQL($filterString); #Remove bad MySQL characters.
my $filterNumber = scalar @$group;
my $queryString = "SELECT VINE_NAME, CONCAT(VINE_NAME,TRAIT)as GLOB, STUDY_NAME FROM
                (SELECT
                    e.dbxref_id          as \"VINE_NAME\",
                    a.value              as \"VALUE\"      ,
                    i.TRAIT              ,
                    i.TRAIT_DESCRIPTION ,
                    REPLACE(g.name, \"-PLOTDATA\", \"\") as \"STUDY_NAME\" ,
                    d.nd_experiment_id   as \"RELATED_IDENTIFIER\"
                FROM STOCK              e
                JOIN LISTDATA           h ON e.name      = h.desig
                JOIN ND_EXPERIMENT_STOCK d ON e.stock_id  = d.stock_id
                JOIN ND_EXPERIMENT_PROJECT f ON d.nd_experiment_id = f.nd_experiment_id
                JOIN ND_EXPERIMENT_PHENOTYPE b ON d.nd_experiment_id = b.nd_experiment_id
                JOIN PROJECT             g ON f.project_id = g.project_id
                JOIN PHENOTYPE          a ON b.phenotype_id = a.phenotype_id
                JOIN ( SELECT i1.value ,
                            i2.value          as \"TRAIT\"      ,
                            REPLACE(i3.value, \"&quot;\", \"\") as \"TRAIT_DESCRIPTION\"
                FROM projectprop i1
                JOIN projectprop i2 on i1.project_id = i2.project_id and i1.rank = i2.rank and (i2.type_id = 1043 or
                i2.type_id = 1048)
                JOIN projectprop i3 on i1.project_id = i3.project_id and i1.rank = i3.rank and i3.type_id = 1060
                ) i
                ON a.observable_id = i.value
                WHERE $filterString
                )
                as BIG GROUP BY GLOB;";

my $vines = query($queryString,3,"");

my %vineCounterHash = %$vines;
my @tempArray;
foreach my $key (keys %vineCounterHash){
    if ($vineCounterHash{$key} >= $filterNumber) {
        push(@tempArray, $key);
    }
}

```

```

    }
  }
  push(@returnedVines, \@tempArray)
}#end group
my %vineCounts;
#Count number of filters each vine passes. Must pass all filters to be selected
foreach my $vineGroup (@returnedVines){
  foreach my $vine (@$vineGroup){
    if (exists($vineCounts{$vine})) {
      $vineCounts{$vine}++;
    }else{
      $vineCounts{$vine} = 1;
    }
  }
}

my @vinesThatPassed; #List of vines that passed initial filters
foreach my $tempVine (keys %vineCounts){
  if ($vineCounts{$tempVine} >= $groupCounter) {
    push(@vinesThatPassed,$tempVine);
  }
}

if ($groupCounter == 0) { #Search all vines if none have passed the initial test
  @vinesThatPassed = @allVinesList;
}

#Process secondary filters here

my $secondaryFilterNum = scalar @secondaryFilters;
my %secondaryVineCounts;
foreach my $quickVine (@vinesThatPassed){
  $secondaryVineCounts{$quickVine} = 0;
}

foreach my $secondaryFilterRef (@secondaryFilters){
  my @secondaryFilter = @$secondaryFilterRef;

  foreach my $passedVine (@vinesThatPassed){
    my $svalue = $secondaryFilter[2];
    my $secondaryQueryString = "SELECT DISTINCT

a.value          as \"VALUE\",
REPLACE(g.name, \"-PLOTDATA\", \"\") as \"STUDY_NAME\"

```

```

FROM STOCK          e
JOIN LISTDATA       h ON e.name          = h.desig
JOIN ND_EXPERIMENT_STOCK d ON e.stock_id   = d.stock_id
JOIN ND_EXPERIMENT_PROJECT f ON d.nd_experiment_id = f.nd_experiment_id
JOIN ND_EXPERIMENT_PHENOTYPE b ON d.nd_experiment_id = b.nd_experiment_id
JOIN PROJECT        g ON f.project_id     = g.project_id
JOIN PHENOTYPE      a ON b.phenotype_id   = a.phenotype_id
JOIN ( SELECT i1.value ,
            i2.value          as \"TRAIT\"      ,
            REPLACE(i3.value, '\"&quot;\";', '\"') as \"TRAIT_DESCRIPTION\"
        FROM projectprop i1
        JOIN projectprop i2 on i1.project_id = i2.project_id and i1.rank = i2.rank and (i2.type_id = 1043 or i2.type_id = 1048)
        JOIN projectprop i3 on i1.project_id = i3.project_id and i1.rank = i3.rank and i3.type_id = 1060
      ) i          ON a.observable_id   = i.value
WHERE (e.dbxref_id = \"$PassedVine\" or h.desig = \"$PassedVine\") and i.TRAIT = \"$SecondaryFilter[0]\";
my $values = query($secondaryQueryString,4,$secondaryFilter[4]);
my @valueArray = @$values;
#print $secondaryQueryString.\"n\";
my $increaseBool = 0; #bool to increase secondaryVineCounts for this vine
#print $valueArray[0].\"n\";
if (defined($valueArray[0])) {
  if (((looks_like_number($valueArray[0])) && ($secondaryFilter[2] ne \"DATE_OBSERVED\") && ($secondaryFilter[3] == 2)) {
    my $average = sum(@valueArray)/(scalar @valueArray);
    ##SWITCH MODULE COULD NOT LOAD IM SORRY
    if ($secondaryFilter[1] eq \"=\") {
      if ($average == $secondaryFilter[2]) {
        $increaseBool = 1;
      }
    }elseif($secondaryFilter[1] eq \"!\"){
      if ($average != $secondaryFilter[2]) {
        $increaseBool = 1;
      }
    }elseif($secondaryFilter[1] eq \"<\"){
      if ($average < $secondaryFilter[2]) {
        $increaseBool = 1;
      }
    }elseif($secondaryFilter[1] eq \"<=\"){
      if ($average <= $secondaryFilter[2]) {
        $increaseBool = 1;
      }
    }elseif($secondaryFilter[1] eq \">\"){
      if ($average > $secondaryFilter[2]) {
        $increaseBool = 1;
      }
    }elseif($secondaryFilter[1] eq \">=\"){
      if ($average >= $secondaryFilter[2]) {

```

```

        $increaseBool = 1;
    }
}
else{
    makeWarningWindow("Average Filter With Non-Numeric Operator Was Ignored in Search");
}
}
elseif($secondaryFilter[3]==1){
    if (looks_like_number($valueArray[0])) {
        foreach my $value (@valueArray){
            ##SWITCH MODULE COULD NOT LOAD IM SORRY
            if ($secondaryFilter[1] eq "=") {
                if ($value == $secondaryFilter[2]) {
                    $increaseBool = 1;
                }
            }
            elseif($secondaryFilter[1] eq "!="){
                if ($value != $secondaryFilter[2]) {
                    $increaseBool = 1;
                }
            }
            elseif($secondaryFilter[1] eq "<"){
                if ($value < $secondaryFilter[2]) {
                    $increaseBool = 1;
                }
            }
            elseif($secondaryFilter[1] eq "<="){
                if ($value <= $secondaryFilter[2]) {
                    $increaseBool = 1;
                }
            }
            elseif($secondaryFilter[1] eq ">"){
                if ($value > $secondaryFilter[2]) {
                    $increaseBool = 1;
                }
            }
            elseif($secondaryFilter[1] eq ">="){
                if ($value >= $secondaryFilter[2]) {
                    $increaseBool = 1;
                }
            }
        }
    }
}
}

}
else{
    foreach my $value (@valueArray){
        if ($secondaryFilter[1] eq "=") {
            if ($value eq $secondaryFilter[2]) {
                $increaseBool= 1;
            }
        }
        elseif($secondaryFilter[1] eq "!="){
            if ($value != $secondaryFilter[2]) {
                $increaseBool = 1;
            }
        }
    }
}
}

```

```

    }elseif($secondaryFilter[1] eq "like"){
        if ($value =~ /.*$secondaryFilter[2].*/) {
            $increaseBool = 1;
        }
    }elseif($secondaryFilter[1] eq "not like"){
        if ($value !~ /.*$secondaryFilter[2].*/) {
            $increaseBool = 1;
        }
    }
}

}

}

}

if ($increaseBool == 1){
    if (exists($secondaryVineCounts{$spassedVine})) {
        $secondaryVineCounts{$spassedVine} = $secondaryVineCounts{$spassedVine} + 1;
    }else{
        $secondaryVineCounts{$spassedVine} = 1;
    }
}

}elseif($secondaryFilter[3] == 1){

}

}

}

%vines=();
my $vineString = "";
my @keys = keys %secondaryVineCounts;
@keys = sort @keys;
my $indexCounter = 1;
my %vinesSeen; #hash to hold list of unique vines added to vineString

foreach my $key (@keys){
    #print $groupCounter."$vineCounts{$key}\n";
    if ($secondaryVineCounts{$key} >= $secondaryFilterNum) {
        my $name = $key;

        if ($groupCounter != 0) {
            $name = getNameFromID($key);
        }

        foreach my $syn (keys %VineSynonyms){ #checking here to see if $name is a hidden vine name. If it is, replace it with the correct shown name

            if (my $index = (index($syn,"---"))!=-1) {
                my $substring = substr($syn, 0,-3);

```

```

        if ($name =~ /Substring:*/) {

            $name = $VineSynonyms{$syn};

        }else{

            #print "$syn does not match $name\n";

        }

        }else{

            #print "$syn does not contain ---\n";

            if ($name eq $syn) {

                $name = $VineSynonyms{$syn};

            }

        }

    }

    if (!exists($vinesSeen{$name})) {

        #VineString = $vineString."$indexCounter)t$name\n";

        $vines{$name} = $key;

        #indexCounter++;

        $vinesSeen{$name} = 1;

    }

}

}

}

my @vinesList = keys %vinesSeen;

@vinesList = sort @vinesList;

foreach my $vine (@vinesList){

    $vineString = $vineString."$indexCounter)t$vine\n";

    $indexCounter++;

}

#Finally make window with vines that pass the filtering

makeVineListingWindow($vineString);

}

sub vineyardQuery{

    #Takes: Vineyard name

    #Returns: Nothing

    #Purpose: Query for vines based on vineyard name

    my ($name) = @_;

    $name = escapeMySQL($name);

    my $length = length $name;

    my $queryString = "select e.name as 'VINE_NAME', e.dbxref_id as 'ID'

        from nd_experiment c

        JOIN nd_experiment_stock d on c.nd_experiment_id = d.nd_experiment_id

        JOIN stock e on d.stock_id = e.stock_id

        JOIN nd_experiment_project f on c.nd_experiment_id = f.nd_experiment_id

        JOIN project g on f.project_id = g.project_id";

```

```

        where substring(g.name ,1,$length) = \'$name\'
        GROUP BY 1
        ;";
query($queryString,2,"");
}

#####MORE QUERY PROCESSING#####

sub query{
    #Takes: Query string and a query type
    #Returns: Pointer to vine listing.
    #Purpose: Conducts a query based on an input string ($query) and a number ($type). 1 denotes a vine name or parent search: return all observations for each vine. 2 denotes a vineyard or trait
search: first return all vines that meet the search requirements, then get all observations for those vines.
    my ($query,$type,$yearString) = @_ ;
    my $dbh = DBI->connect ("DBI:mysql:$database:$host:$port", $username, "", {RaiseError=>0,PrintError=>1}); #Connect to database
    my $sqlQuery ;
    if (defined($dbh)) {
        $sqlQuery = $dbh->prepare($query) or die makeWarningWindow("Can't prepare $query:". $dbh->errstr); #Prepare the query
    }else{
        makeWarningWindow("Bad Connection Settings. Change in Options Menu");
        #options_menu();
        return ();
    }
    my $rv = $sqlQuery->execute or die makeWarningWindow("Can't execute the query: ".$sqlQuery->errstr); #Execute the query
    my $counter = 0; #initialize counter for rows
    my @report = []; #Initialize array for report
    while (my @row= $sqlQuery->fetchrow_array()) { #get each row of results
        push (@report,@row); #add pointer to row array to report array
    }
    if (($type == 1)) { #NO LONGER BEING USED!Type 1 indicates a parental or name query. 7 columns returned.
        return(\@report);
    }elseif($type==2){ #Type 2 indicates a name , parental, or vineyard query. returns list of vines.
        %vines = ();
        my @vineNameStrings;
        my $string = "";
        my $indexCounter = 1;
        foreach my $row (@report){
            if ((defined($row->[0])) && (defined($row->[1]))) {
                $vines{$row->[0]}=$row->[1];
            }
        }
        @vineNameStrings = sort keys %vines;
        my %vinesSeen; #Hash that holds all seen vines added to list
        foreach my $vine (@vineNameStrings){
            foreach my $key (keys %VineSynonyms){ #checking here to see if $name is a hidden vine name. If it is, replace it with the correct shown name

```

```

        if (my $index = (index($key, "---")!)-1) {
            my $substring = substr($key, 0, -3);
            if ($vine =~ /$substring.*/) {
                $vine = $VineSynonyms{$key};
            }
        } else {
            if ($vine eq $key) {
                $vine = $VineSynonyms{$key};
            }
        }
    }
}

if (!exists($vinesSeen{$vine})) { # $string = $string."$indexCounter)t$vine\n";

    # $indexCounter++;
    $vinesSeen{$vine} = 1;
}

}

my @vinesList = keys (%vinesSeen);
@vinesList = sort @vinesList;
foreach my $vine (@vinesList) {
    $string = $string."$indexCounter)t$vine\n";
    $indexCounter++;
}

makeVineListingWindow($string); # Make vine listing window with list of vines returned
} elsif ($type == 3) { # Type 3 just returns a report as an array
    # $data[0] is vine counts hash. $data[1] is year hash
    my %vineCounterHash;
    my @years;
    if ($yearString ne "") {
        # Assume if it is defined it is valid at this point. validation occurs at entry
        @years = split("-", $yearString);
    } else {
        # set all valid years
        my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) =
            localtime(time);
        $years[0] = 0000;
        $year += 1900; # Year is originally years since 1900, so I add 1900 to get the current year
        $years[1] = $year;
    }
}

foreach my $row (@report) {
    my $vine = $row->[0];
    my $year = -1;
    if (defined($vine)) {
        if (exists($VineSynonyms{$vine})) {
            $vine = $VineSynonyms{$vine};
        }
    }
}

```



```

if (defined($row->[2])) {
    $row->[2] =~ m/. * *([0-9]{4}) *.* /;
    #print "Study: $row->[2]\n";
    $year = $1;

    "a" =~ /a/;
}
#Vine name exists and year is equal to one of the years given, or in between them
if (exists($vineCounterHash{$vine}) && ( ($year == $years[0]) || ($year == $years[1]) || (($year > $years[0]) && ($year < $years[1])) )) {
    $vineCounterHash{$vine} = $vineCounterHash{$vine} + 1;
    #year is equal to or between given year range
} elsif ( ($year == $years[0]) || ($year == $years[1]) || (($year > $years[0]) && ($year < $years[1])) ) {
    $vineCounterHash{$vine} = 1
}
}
}
"a" =~ /a/; #resets $!. dont delete
}

return(%vineCounterHash);
} elsif ($type == 4) { #return values for a query given a vine name and trait name

my @values;
my @years;
if ($yearString ne "") {
    #Assume if it is defined it is valid at this point. validation occurs at entry
    @years = split("-", $yearString);
} else {
    #set all avlid years
    my ($sec,$min,$hour,$mday,$mon,$year,$yday,$dwday,$yday,$isdst) =
        localtime(time);
    $years[0] = 0000;
    $year += 1900; #Year is originally years since 1900, so I add 1900 to get the current year
    $years[1] = $year;
}
foreach my $row (@report) {
    foreach my $thing (@$row) {
    }
    my $year = -1; #If no year on the study, get rid of it
    if (defined($row->[0])) {
        my $vine = $row->[0];
        if (exists($vineSynonyms{$vine})) {
            $vine = $vineSynonyms{$vine};
        }

        if (defined $row->[1]) {
            $row->[1] =~ m/. * *([0-9]{4}) *.* /;
            $year = $1;

```

```

    "a" =~ /a/;
}
if (($year == $years[0]) || ($year == $years[1]) || (($year > $years[0]) && ($year < $years[1]))) {

    push(@values,$vine);
}
}

#print "Value total $valueTotal\n";
return \@values;

# foreach my $row (@report){
#     foreach my $value (@$row){ #print each value with a tab after it
#         push(@vines,$value);
#     }
# }
# return (\@vines);
}

}

#####REPORT FUNCTIONS#####

sub traitReport{
    #Takes: Output file name
    #Returns: Nothing:

    #Purpose: Generates a report from a vine list. Used by al search types

    my ($outFile) = @_ ;
    #Open temp file. This file gets deleted soon after
    open(my $OUTFILE, ">", "$dir\tempoutfileforreport.txt") or die makeWarningWindow("Could not open $dir\tempoutfileforreport.txt: $!\n"); #Open the file for appending
    print $OUTFILE "VINE_NAME\tCROSS\tVALUE\tTRAIT\tTRAIT_DESCRIPTION\tSTUDY\tRELATED_IDENTIFIER\n"; #Print cokumm headers to file
    close $OUTFILE;
    my @altLookups;
    #The next few loops just make sure I am including data for all synonyms
    foreach my $vineName (keys %vines){
        foreach my $key (keys %VineSynonyms){
            if (my $index = (index($key,"---"))!=-1) {
                my $substring = substr($key, 0,-3);
                if ($vineName =~ /$substring.*/ ) {
                    my $addFlag = 1;
                    foreach my $value (keys %vines){
                        if ($value eq $VineSynonyms{$key}) {
                            $addFlag = 0;
                        }
                    }
                }
            }
        }
    }
}

```

```

}if($addFlag == 1){
    my $id = getIdFromName($VineSynonyms{$key});
    if (defined($id)) {
        $vines{$VineSynonyms{$key}} = $id
    }else{
        #makeWarningWindow("Could not get GID for vine: $VineSynonyms{$key}");
    }

}
}
}else{
    if ($vineName eq $key) {
        my $addFlag = 1;
        foreach my $value (keys %vines){
            if ($value eq $VineSynonyms{$key}) {
                $addFlag = 0;
            }

}if($addFlag == 1){
    my $id = getIdFromName($VineSynonyms{$key});
    if (defined($id)) {
        $vines{$VineSynonyms{$key}} = $id
    }else{
        makeWarningWindow("Could not get GID For vine: $VineSynonyms{$key}");
    }

}
}
}

}

foreach my $key (keys %AltLookups){
    if (my $index = (index($key,"---"))!=-1) {
        push(@altLookups,@{$AltLookups{$key}});
    }else{
        if ($vineName eq $key) {
            push(@altLookups,@{$AltLookups{$key}});
        }

}

}

}

}

#Pull out all data for all vines here
my @a = keys %vines;

```

```

my $total = (scalar @a)+(scalar @altLookups);
my $counter = 0;
foreach my $nameVine (keys %vines){
    if ((defined($vines{$nameVine}))) {

        if (my $index = (index($nameVine,"---")!=-1) {
            print "Searching for $nameVine\n";
            vineQuerier($nameVine, 2);

            $counter++;
            print $nameVine."\n";
            my $percent = ($counter/$total) * 100;
            print "$percent% complete\n";
        }else{
            if ((defined($vines{$nameVine}))) {
                print "Searching for $nameVine\n";
                vineQuerier($vines{$nameVine}, 1);
                $counter++;
                print $nameVine."\n";
                my $percent = ($counter/$total) * 100;
                print "$percent% complete\n";
            }
        }
    }
}

#Alt lookups are a list of synonyms to search and include as well
foreach my $nameVine (@altLookups){
    if (my $index = (index($nameVine,"---")!=-1) {
        print "Searching for $nameVine\n";
        vineQuerier($nameVine, 2);
        $counter++;
        print $nameVine."\n";
        my $percent = ($counter/$total) * 100;
        print "$percent% complete\n";
    }else{
        if ((defined($vines{$nameVine}))) {
            print "Searching for $nameVine\n";
            vineQuerier($vines{$nameVine}, 1);
            $counter++;
            print $nameVine."\n";
            my $percent = ($counter/$total) * 100;
            print "$percent% complete\n";
        }
    }
}
}

```

```

#Now call reportMaker2 to make the report
my $optionsString = "";
if ($avgBool == 1) {
    $optionsString = $optionsString." --average";
}
if ($analysisBool == 1) {
    $optionsString = $optionsString." --analysis";
}
open(my $TRAITFILE, ">", "$dir\traitfileforreport.txt") or makeWarningWindow("Could not open $dir\traitfileforreport.txt: $!");
foreach my $trait (@traitsToInclude){
    my $scale = getTraitScale($trait);
    if (defined($scale)) {
        print $TRAITFILE "$trait\t$scale\n";
    }else{
        print $TRAITFILE "$trait\t\n";
    }
}
close $TRAITFILE;
open(my $STUDYFILE, ">", "$dir\studyfileforreport.txt") or die makeWarningWindow("Could not open $dir\studyfileforreport.txt: $!");
foreach my $study (@studiesToInclude){
    print $STUDYFILE "$study\n";
}
close $STUDYFILE;
$dir =~ tr{\\}{};

system "perl \"$dir/reportMaker2.pl\" \"$dir/tempoutfileforreport.txt\" \"$outFile.csv\" \"$dir\traitfileforreport.txt\" \"$dir\studyfileforreport.txt\" $yearsSince $optionsString"; #call
reportMaker.pl with correct arguments

if (1 == 1) { #If I don't do this the temp file gets deleted before reportMaker2.pl is done with it. Sorta hacks, but it worked

    $dir =~ tr{/}{};

    unlink "$dir/tempoutfileforreport.txt";

    unlink "$dir\traitfileforreport.txt";
    unlink "$dir\studyfileforreport.txt";
}
}

#####Additional Utility functions#####
sub vineQuerier{
    #Takes: Name of vine, and type of search (1 or 2); 1 = search by ID, 2 = search by name
    #Returns: Nothing
    #Purpose: Query the vine
    my ($vineName, $type) = @_;
    open(my $OUTFILE, ">>", "$dir\tempoutfileforreport.txt") or die makeWarningWindow("Could not open $dir\tempoutfileforreport.txt: $!\n"); #Open the file for appending
    if (!defined($vineName) || !defined($type)) {
        makeWarningWindow("POSSIBLE ERROR IN SEARCH TERM, OR UNKNOWN VINE IN SYNONYM: $vineName, $type");
        return;
    }
}

```

```

#print "Processing $vineName\n";

$vineName = escapeMySQL($vineName);
my $stringLength = length $vineName;
if ($type == 1) {
    $vineName = "e.dbxref_id = $vineName AND
                h.gid = $vineName";
}elsif($type == 2){
    my $index = index($vineName,"-");
    $vineName = substr($vineName,0,-3);
    $vineName = "e.name like \"$vineName%\" AND h.desig like \"$vineName%\"";
}
my $query = "SELECT e.name          as \"VINE_NAME\" ,
                h.grpname         as \"CROSS\" ,
                a.value            as \"VALUE\" ,
                i.TRAIT            ,
                i.TRAIT_DESCRIPTION ,
                REPLACE(g.name, \"-PLOTDATA\", \"\") as \"STUDY_NAME\" ,
                d.nd_experiment_id  as \"RELATED_IDENTIFIER\"
FROM STOCK      e
JOIN LISTDATA   h ON e.name      = h.desig
JOIN ND_EXPERIMENT_STOCK d ON e.stock_id = d.stock_id
JOIN ND_EXPERIMENT_PROJECT f ON d.nd_experiment_id = f.nd_experiment_id
JOIN ND_EXPERIMENT_PHENOTYPE b ON d.nd_experiment_id = b.nd_experiment_id
JOIN PROJECT    g ON f.project_id = g.project_id
JOIN PHENOTYPE  a ON b.phenotype_id = a.phenotype_id
JOIN ( SELECT i1.value ,
                i2.value      as \"TRAIT\" ,
                REPLACE(i3.value, \"&quot;\", \"\") as \"TRAIT_DESCRIPTION\"
FROM projectprop i1
JOIN projectprop i2 on i1.project_id = i2.project_id and i1.rank = i2.rank and (i2.type_id = 1043 or i2.type_id = 1048)
JOIN projectprop i3 on i1.project_id = i3.project_id and i1.rank = i3.rank and i3.type_id = 1060
) i          ON a.observable_id = i.value
WHERE $vineName #The second number in the substring field gives the length of the search string
GROUP BY 1,3,4,5,6,7
;";

#print $query."n";

my $dbh = DBI->connect("DBI:mysql:$database:$host:$port", $username, "", {RaiseError=>0,PrintError=>1}); #Connect to database
my $sqlQuery;
if (defined($dbh)) {
    $sqlQuery = $dbh->prepare($query) or die makeWarningWindow("Can't prepare $query: ".$dbh->errstr); #Prepare the query
}else{
    makeWarningWindow("Bad Connection Settings. Change in Options Menu" );
    #options_menu();
    return ();
}
my $rv = $sqlQuery->execute or die makeWarningWindow("Can't execute the query: ".$sqlQuery->errstr); #Execute the query

```

```

my @report = [];
my $somethingCounter = 1;
while (my @row= $sqlQuery->fetchrow_array()) { #get each row of results
    push (@report,@row); #add pointer to row array to report array
    #print "Row: $somethingCounter\n";
    $somethingCounter++;
}
#Print data to output file
my $printCounter = 1;
foreach my $row (@report){ #print all values in report array
    foreach my $value (@$row){
        print SOUTFILE $value."t";
    }
    #print "Printing $printCounter\n";
    $printCounter++;
    print SOUTFILE "\n";
}
my $num = scalar @report;

close(SOUTFILE); #Close temp file
}

sub getVariableList{
    #Takes: Nothing
    #Returns: Array of traits in BMS
    #Purpose: Get list of all traits in BMS
    #queries the database and returns an array of all variables in the database

    my $dbh = DBI->connect("DBI:mysql:$database:$host:$port", $username, "", {RaiseError=>0,PrintError=>1}); #Connect to database
    my $query = "select * from projectprop where type_id = 1043 or type_id = 1048";
    if (defined($dbh)) {

        my $sqlQuery = $dbh->prepare($query) or die makeWarningWindow("Can't prepare $query:". $dbh->errstr); #Prepare the query
        my $rv = $sqlQuery->execute or die makeWarningindow("Can't execute the query:". $sqlQuery->errstr); #Execure the query
        my @report = [];
        my %traitUnique;
        my @traits;
        while (my @row= $sqlQuery->fetchrow_array()) { #get each row of results
            if (exists($row[3])) {
                $traitUnique{$row[3]} = 1;
            }
        }
        @traits = keys %traitUnique;
        @traits = sort @traits;
    }
}

```

```

        return(@traits);
    }else{
        makeWarningWindow("Bad Connection Settings. Change in Options Menu" );
        #options_menu();
        return ();
    }
    return (); #returns an empty list
}

sub getStudyList{
    #Takes: Nothing
    #Returns: Array of study names
    #Purpose: Get list of all studies from BMS
    #RETURN LIST OF STUDIES IN DATABASE
    my $dbh = DBI->connect("DBI:mysql:$database:$host:$port", $username, "", {RaiseError=>0,PrintError=>1}); #Connect to database
    my $query = "select name from project;";
    if (defined($dbh)) {
        my $sqlQuery = $dbh->prepare($query) or die makeWarningWindow("Can't prepare Query:". $dbh->errstr); #Prepare the query
        my $rv = $sqlQuery->execute or die makeWarningWindow("Can't execute the query:". $sqlQuery->errstr); #Execute the query
        my @report = [];
        my %studyUnique;
        my @studies;
        while (my @row= $sqlQuery->fetchrow_array()) { #get each row of results
            if (exists($row[0])) {
                if($row[0]!~/.*PLOTDATA/ &&($row[0] !~/.*ENVIRONMENT/)&&($row[0]!~/.*#[0-9]+/)){
                    $studyUnique{$row[0]} = 1;
                }
            }
        }
        @studies = keys %studyUnique;
        @studies = sort @studies;
        return(@studies);
    }else{
        makeWarningWindow("Bad Connection Settings. Change in Options Menu" );
        #options_menu();
        return ();
    }
    return (); #returns an empty list
}

sub escapeMySQL{
    #Takes: String to be put into MySQL
    #Returs: A string that removes MySQL special characters
    #Purpose: No MySQL shenanigans, but its your DB anyways, so IDK
    #returns string that does not contain mysql stuff in it
    my ($string) = @_ ;
    $string =~ s/0'bnrtz|%-_|0|\||b|n|r|t|z|\\\|\\|_|%|_|/g;
    return ($string);
}

```



```

}

sub writeConfigFile{
    #Takes: Nothing
    #Returns: Nothing
    #Writes DB connection settings to appropriate file
    open(my $CONFIG_FILE,">","$dir\\Configuration\\search.conf");
    print $CONFIG_FILE "#Change the values if quotes if necessary. See Section 11 in Workflow documentation for how to determine values for these fields
#DO NOT ALTER ANYTHING ABOUT THIS FILE OTHER THAN THE VALUES IN QUOTES
\"$database\"\\t#Database name
\"$port\"\\t#Port Number
\"$host\"\\t#HostName
\"$username\"\\t#User name
firstTimeFlag\\tFALSE";

    close $CONFIG_FILE;
}

sub getDatabases{
    #Takes: Nothing
    #Returns pointer to array of database names
    #Get list of available databases
    my $dbh = DBI->connect("DBI:mysql:information_schema:$host:$port", $username, "",{RaiseError=>0,PrintError=>1}); #Connect to database
    my $databases = $dbh->selectcol_arrayref('show databases');
    return ($databases);
}

sub writeVineConfigFile{
    #Takes: Nothing
    #Returns: Nothing
    #Purpose: Write vine synonyms to file, and read in those synonyms to file (only way to load new synonyms)
    open(my $CONFIG_FILE,">","$dir\\Configuration\\vineNames.conf") or makeWarningWindow("Could not open $dir\\Configuration\\vineNames.conf");
    my @keyNames = keys %VineSynonyms;
    for(my $counter = 0; $counter < scalar @keyNames; $counter++){
        print $CONFIG_FILE "$keyNames[$counter]=$VineSynonyms{$keyNames[$counter]}";
        #print "$keyNames[$counter]=$VineSynonyms{$keyNames[$counter]}\\n";
        if ($counter != ((scalar @keyNames)-1)) {
            print $CONFIG_FILE "\\n";
        }
    }

    close $CONFIG_FILE;
    readVineNames(); #Read in new data
}

sub getIDFromName{
    #Takes: GID From BMS
    #Returns: Vine name from BMS
    #Purpose: Get the name of a vine given an ID for the purposes of displaying the vine name to screen
    my ($name) = @_;
    my $dbh = DBI->connect("DBI:mysql:$database:$host:$port", $username, "",{RaiseError=>0,PrintError=>1}); #Connect to database

```

```

my $query = "select dbxref_id from stock where name =\"$name\"";
if (defined($dbh)) {
    my $sqlQuery = $dbh->prepare($query) or die makeWarningWindow("Can't prepare $query:". $dbh->errstr); #Prepare the query
    my $rv = $sqlQuery->execute or die makeWarningWindow("Can't execute the query:". $sqlQuery->errstr); #Execute the query
    my %idUnique;
    while (my @row= $sqlQuery->fetchrow_array()) { #get each row of results
        if (exists($row[0])) {
            $idUnique{$row[0]} = 1;
        }
    }
    foreach my $id (keys %idUnique){
        return ($id);
    }
} else {
    makeWarningWindow("Bad Connection Settings. Change in Options Menu" );
    #options_menu();
    return ();
}
return (); #returns an empty value
}

sub getNameFromID{
    #Takes: Name of a vine
    #Returns: GID from BMS
    #Purpose: COntvert a vine name to an ID for the purposes of looking up the GID for a further search
    my ($id) = @_ ;
    my $dbh = DBI->connect("DBI:mysql:$database:$host:$port", $username, "", {RaiseError=>0,PrintError=>1}); #Connect to database
    my $query = "select name from stock where dbxref_id = $id;";
    #print $query."n";
    if (defined($dbh)) {
        my $sqlQuery = $dbh->prepare($query) or die makeWarningWindow("Can't prepare $query:". $dbh->errstr); #Prepare the query
        my $rv = $sqlQuery->execute or die makeWarningWindow("Can't execute the query:". $sqlQuery->errstr); #Execute the query
        my %uniqueName;
        while (my @row= $sqlQuery->fetchrow_array()) { #get each row of results
            if (exists($row[0])) {
                $uniqueName{$row[0]} = (); #{$row[0]} = 1;
            }
        }
        foreach my $name (keys %uniqueName){
            return ($name);
        }
    } else {
        makeWarningWindow("Bad Connection Settings. Change in Options Menu" );
        #options_menu();
        return ();
    }
}
return (); #returns an empty value

```

```

}

sub getTraitScale{
    #Takes: Name of trait
    #Returns: String representing scale of trait
    #Purpose: Get the scale of a trait
    my ($trait) = @_.;
    my $dbh = DBI->connect("DBI:mysql:$database:$host:$port", $username, "", {RaiseError=>0, PrintError=>1}); #Connect to database
    my $query = "select scale from standard_variable_details where stdvar_name = \"\$trait\";";
    if (defined($dbh)) {
        my $sqlQuery = $dbh->prepare($query) or die makeWarningWindow("Can't prepare $query: ". $dbh->errstr); #Prepare the query
        my $rv = $sqlQuery->execute or die makeWarningWindow("Can't execute the query: ". $sqlQuery->errstr); #Execute the query
        my $scale;
        while (my @row= $sqlQuery->fetchrow_array()) { #get each row of results
            if (exists($row[0])) {
                $scale = $row[0];
            }
        }
        return $scale;
    }else{
        makeWarningWindow("Bad Connection Settings. Change in Options Menu" );
        #options_menu();
        return ;
    }
    return;
}

sub getTraitValueInfo{
    #Takes: Nothing
    #Returns: Hash where keys are trait names and values are string representing trait info
    #If trait is numeric, string will show average and standard deviation. If trait is categorical, string will show all valid values
    my $filename = "$dir\\Configuration\\traitDescriptions.conf";
    my %traitValuesHash;
    open(my $CONFIG_FILE, "<", $filename) or makeWarningWindow("Could not open $filename: ${!}n");
    while (eof($CONFIG_FILE)) {
        my $line = readline($CONFIG_FILE);
        chomp $line;
        my @splitLine = split("t", $line);
        $splitLine[1] =~ s/^\n/g;

        $traitValuesHash{$splitLine[0]} = $splitLine[1];
    }
    return \%traitValuesHash;
}

sub updateTraitValueInfo{
    #Takes: Nothing
    #Returns: Nothing
    #Purpose: Update configuration file which contains descriptors for all traits to help users search

```

```

#Run when user clicks button to run this method
my $totalObservations = 0;
my @traits = getVariableList(); #Gets list of all variables in database
my %traitValuesHash; #Hash to describe trait values. Key is trait name, value is description string
foreach my $trait (@traits){
    if ($trait ne "DATE_OBSERVED") {

        my $traitString = "i.TRAIT = \"$trait\"";
        my $queryString = "SELECT
                                a.value          as 'VALUE'

                                FROM ND_EXPERIMENT_STOCK d
                                JOIN ND_EXPERIMENT_PROJECT f ON d.nd_experiment_id = f.nd_experiment_id
                                JOIN ND_EXPERIMENT_PHENOTYPE b ON d.nd_experiment_id = b.nd_experiment_id
                                JOIN PROJECT g ON f.project_id = g.project_id
                                JOIN PHENOTYPE a ON b.phenotype_id = a.phenotype_id
                                JOIN ( SELECT i1.value ,
                                        i2.value          as \"TRAIT\" ,
                                        REPLACE(i3.value, \"&quot;\", \"\") as \"TRAIT_DESCRIPTION\"
                                FROM projectprop i1
                                JOIN projectprop i2 on i1.project_id = i2.project_id and i1.rank = i2.rank and (i2.type_id = 1043 or i2.type_id = 1048)
                                JOIN projectprop i3 on i1.project_id = i3.project_id and i1.rank = i3.rank and i3.type_id = 1060
                                ) i          ON a.observable_id = i.value

                                WHERE $traitString #The second number in the substring field gives the length of the search string

                                ";

        my $dbh = DBI->connect("DBI:mysql:$database:$host:$port", $username, "", {RaiseError=>0, PrintError=>1}); #Connect to database
        my $sqlQuery;
        if (defined($dbh)) {

            $sqlQuery = $dbh->prepare($queryString) or die makeWarningWindow("Can't prepare $queryString:". $dbh->errstr); #Prepare the query
        }else{
            makeWarningWindow("Bad Connection Settings. Change in Options Menu" );
            #options_menu();
            return ();
        }

        my $rv = $sqlQuery->execute or die makeWarningWindow("Can't execute the query:". $sqlQuery->errstr); #Execute the query
        my @report = [];

        while (my @row= $sqlQuery->fetchrow_array()) { #get each row of results
            push (@report, @row); #add pointer to row array to report array
        }

        my @list;

        foreach my $row (@report){
            foreach my $value (@$row){
                push(@list, $value);
            }
        }
    }
}

```

```

    }
}
$queryString = "SELECT e.name          as \"VINE_NAME\"

FROM STOCK          e
JOIN LISTDATA       h ON e.name      = h.desig
JOIN ND_EXPERIMENT_STOCK d ON e.stock_id = d.stock_id
JOIN ND_EXPERIMENT_PROJECT f ON d.nd_experiment_id = f.nd_experiment_id
JOIN ND_EXPERIMENT_PHENOTYPE b ON d.nd_experiment_id = b.nd_experiment_id
JOIN PROJECT        g ON f.project_id = g.project_id
JOIN PHENOTYPE      a ON b.phenotype_id = a.phenotype_id
JOIN ( SELECT i1.value ,
           i2.value          as \"TRAIT\" ,
           REPLACE(i3.value, \"&quot;\", \"\") as \"TRAIT_DESCRIPTION\"
FROM projectprop i1
JOIN projectprop i2 on i1.project_id = i2.project_id and i1.rank = i2.rank and (i2.type_id = 1043 or i2.type_id = 1048)
JOIN projectprop i3 on i1.project_id = i3.project_id and i1.rank = i3.rank and i3.type_id = 1060
) i          ON a.observable_id = i.value
WHERE StraitString #The second number in the substring field gives the length of the search string
GROUP BY 1;";

my $dbh2 = DBI->connect("DBI:mysql:$database:$host:$port", $username, "", {RaiseError=>0, PrintError=>1}); #Connect to database
my $sqlQuery2;
if (defined($dbh2)) {

    $sqlQuery2 = $dbh2->prepare($queryString) or die makeWarningWindow("Can't prepare $queryString:". $dbh2->errstr); #Prepare the query
} else {
    makeWarningWindow("Bad Connection Settings. Change in Options Menu");
    #options_menu();
    return ();
}
my $rv2 = $sqlQuery2->execute or die makeWarningWindow("Can't execute the query:". $sqlQuery2->errstr); #Execute the query
my @report2 = [];
while (my @row = $sqlQuery2->fetchrow_array()) { #get each row of results
    push (@report2, @row); #add pointer to row array to report array
}
my @listVines;
foreach my $row (@report2) {
    foreach my $value (@$row) {
        push(@listVines, $value);
    }
}
my $totalVines = scalar @listVines;
if (looks_like_number($list[0])) {
    #want average and standard deviation
    my $average = sum(@list)/(scalar @list);
    my $standardDeviation = 0;
}

```

```

if (scalar(@list) == 1) {
    #standard deviation is still 0
}else{
    my $sqtotal = 0;
    foreach my $value (@list){
        $sqtotal += ($average-$value)**2;
    }
    my $total = scalar @list;
    $standardDeviation = $sqtotal / ($total-1);
    $standardDeviation = $standardDeviation**.5;
}
my $max = max(@list);
my $min = min(@list);
my $total = scalar @list;
$traitValuesHash{$trait} = ".Avg: $average,StdDev: $standardDeviation,Min: $min,Max: $max,No. Obs.: $total,No. Vines: $totalVines";
$totalObservations += $total;
}else{
    #categorical data
    my %categoriesHash; #Just holds all of the unique values for the category
    foreach my $value (@list){
        if (!exists($categoriesHash{$value})) {
            $categoriesHash{$value} = 1;
        }
    }
    my $sampleCategories = ".Example Values.";
    my @keyList = keys %categoriesHash;
    for (my $counter = 0; $counter < 5 && $counter < scalar @keyList; $counter++){
        my $num = $counter+1;#Just for formatting in the display. List displayed starts at 1 and not zero.
        $sampleCategories.= $num." ".keyList[$counter].",";
    }
    my $total = scalar @list;
    $sampleCategories=".No Obs.: $total, No. Vines: $totalVines";
    $traitValuesHash{$trait} = $sampleCategories;
    $totalObservations += $total;
}

}

}

my $filename = "$dir\\Configuration\\traitDescriptions.conf";
open(my $CONFIG_FILE, ">", $filename) or makeWarningWindow("Could not open $filename: $!");
foreach my $key (keys %traitValuesHash){
    print $CONFIG_FILE $key."t".$traitValuesHash{$key}."\n";
}
close($CONFIG_FILE);
print "TOTAL OBSERVATIONS = $totalObservations\n";
}

```

```

sub updateVineAverageInfo{
    #Called to update the average value for each trait per vine. This infor is written to a file
    #first get vine list
    #Then for each vine get averages
    #Print it all to a file
    my $queryString = "select desig from LISTDATA group by desig;";
    my $dbh = DBI->connect("DBI:mysql:Database:$host:$port", $username, "", {RaiseError=>0,PrintError=>1}); #Connect to database
    my @vinelist = @getAllVines();

    #Got list of all vines. Time to execute a new query
    my %vineToTrait; #hash that maps vine name(key) to a pointer to a hash. The hash that is pointed to has structure of trait(key) and average (value).
    foreach my $vine(@vinelist){
        #I want to get all trait data now
        print "Searching for trait info for $vine\n";
        my $queryString2 = "SELECT e.name          as \"VINE_NAME\",
            h.grpname          as \"CROSS\"      ,
            a.value            as \"VALUE\"      ,
            i.TRAIT            ,
            i.TRAIT_DESCRIPTION ,
            REPLACE(g.name, \"-PLOTDATA\", \"\") as \"STUDY_NAME\" ,
            d.nd_experiment_id  as \"RELATED_IDENTIFIER\"

        FROM STOCK            e
        JOIN LISTDATA         h ON e.name          = h.desig
        JOIN ND_EXPERIMENT_STOCK d ON e.stock_id    = d.stock_id
        JOIN ND_EXPERIMENT_PROJECT f ON d.nd_experiment_id = f.nd_experiment_id
        JOIN ND_EXPERIMENT_PHENOTYPE b ON d.nd_experiment_id = b.nd_experiment_id
        JOIN PROJECT          g ON f.project_id    = g.project_id
        JOIN PHENOTYPE        a ON b.phenotype_id  = a.phenotype_id
        JOIN ( SELECT i1.value ,
            i2.value          as \"TRAIT\"      ,
            REPLACE(i3.value, \"&quot;\", \"\") as \"TRAIT_DESCRIPTION\"

        FROM projectprop i1
        JOIN projectprop i2 ON i1.project_id = i2.project_id and i1.rank = i2.rank and (i2.type_id = 1043 or i2.type_id = 1048)
        JOIN projectprop i3 ON i1.project_id = i3.project_id and i1.rank = i3.rank and i3.type_id = 1060
        ) i
        ON a.observable_id  = i.value
        WHERE e.name = \"$vine\" AND h.desig = \"$vine\" #The second number in the substring field gives the length of the search string
        GROUP BY 1,3,4,5,6,7;";

        my $sqlQuery2;
        if (defined($dbh)) {

            $sqlQuery2 = $dbh->prepare($queryString2) or die makeWarningWindow("Can't prepare $queryString2.".$dbh->errstr); #Prepare the query
        }else{
            makeWarningWindow("Bad Connection Settings. Change in Options Menu" );
            #options_menu();
            return ();
        }
    }
}

```

```

my $rv2 = $sqlQuery2->execute or die makeWarningWindow("Can't execute the query:". $sqlQuery2->errstr); #Execute the query
my @report2 = [];
while (my @row= $sqlQuery2->fetchrow_array()) { #get each row of results
    push (@report2,\@row); #add pointer to row array to report array
}
#organize data into hash of lists where key is trait name, value is pointer to list of values for that trait
my %traitValueListHash; #See above statement
foreach my $row(@report2){
    if ((looks_like_number($row->[2])) && ($row->[3] ne "DATE_OBSERVED")) {
        #If the value of the trait

        if (exists($traitValueListHash{$row->[3]}) {
            #trait has already been seen. list of values exists
            my $listRef = $traitValueListHash{$row->[3]}; #get list pointer from hash
            my @tempList = @$listRef; #retrieve array from pointer
            push(@tempList, $row->[2]); #push value to array
            $traitValueListHash{$row->[3]} = \@tempList; #set hash list to the updated list
        }else{
            #Trait has not been seen yet
            my @tempList; #make temp array
            push(@tempList, $row->[2]); #push value to temp list;
            $traitValueListHash{$row->[3]} = \@tempList; #set hash value to be pointer to array
        }
    }
}
#traitValueListHash has been fully populated for this vine. Now calculate the average off all the lists
my %traitToAverageHash; #hash that maps the trait (key) to the average value (value)
foreach my $trait (keys %traitValueListHash){
    my $listRef = $traitValueListHash{$trait};
    my @valueList = @$listRef;
    my $totalInList = scalar @valueList;
    my $listTotal = sum(@valueList);
    my $average = $listTotal/$totalInList;
    my @tempList;
    push(@tempList, $average);
    push(@tempList, $listTotal);
    $traitToAverageHash{$trait} = \@tempList;
}
$vineToTrait{$vine} = \%traitToAverageHash;
}
#all vines have been processed and %vineToTrait hash has been fully populated.
#I need to collapse vines based on their synonyms
print "Collapsing averages info based on vine synonym names\n";
foreach my $name(keys %vineToTrait){
    print $name."\n";
    foreach my $syn (keys %VineSynonyms){ #checking here to see if $name is a hidden vine name. If it is, replace it with the correct shown name

```



```

if (my $index = (index($syn,"---"))!=1) {
  my $substring = substr($syn, 0,-3);
  if ($name =~ /$substring.*/) {
    #name is current vine (bad name). $VineSynonyms{$syn} is new name. Need to go look up new name hash, and merge all shared traits

    my $newName = $VineSynonyms{$syn};
    my $badHashRef = $vineToTrait{$name};
    my %badHash;
    if (defined($badHashRef)) {
      %badHash = %$badHashRef;
    }

    print $name."\n";
    print $badHashRef."\n";

    if (exists($vineToTrait{$newName})) {
      #there is info for the vine under the new name. this will happen almost always
      my %goodHash = %{$vineToTrait{$newName}};
      foreach my $trait(keys %badHash){
        if (exists($badHash{$trait})) {
          #Overlap between traits. Merge average values
          my @badInfo = @{$badHash{$trait}};
          my @goodInfo;
          my $newAverage;
          my $newTotal;
          my @newInfo;
          if(exists($goodHash{$trait})){
            @goodInfo= @{$goodHash{$trait}};
            $newAverage = $badInfo[0]*$badInfo[1]+$goodInfo[0]*$goodInfo[1];
            $newTotal = $badInfo[1]+$goodInfo[1];
          }else{
            @newInfo = @badInfo;
          }

          push(@newInfo, $newAverage);
          push(@newInfo, $newTotal);
          $goodHash{$trait} = \@newInfo;
        }

      }

      #Set the hash pointer to the updated hash
      $vineToTrait{$newName} = \%goodHash;
      delete $vineToTrait{$name};
    }

  }else{
    #print "$syn does not match $name\n";
  }
}

```

```

}else{
    #print "$syn does not contain ---\n";
    if ($name eq $syn) {

        my $newName = $VineSynonyms{$syn};
        my %badHash = %{$VineToTrait{$name}};
        if (exists($VineToTrait{$newName})) {
            #there is info for the vine under the new name. this will happen almost always
            my %goodHash = %{$VineToTrait{$newName}};
            foreach my $trait(keys %badHash){
                if (exists($badHash{$trait})) {
                    #Overlap between traits. Merge average values
                    my @badInfo = @{$badHash{$trait}};
                    my @goodInfo = @{$goodHash{$trait}};
                    my $newAverage = $badInfo[0]*$badInfo[1]+$goodInfo[0]*$goodInfo[1];
                    my $newTotal = $badInfo[1]+$goodInfo[1];
                    my @newInfo;
                    push(@newInfo, $newAverage);
                    push(@newInfo, $newTotal);
                    $goodHash{$trait} = \@newInfo;
                }
            }
            #Set the hash pointer to the updated hash
            $VineToTrait{$newName} = \%goodHash;
            delete $VineToTrait{$name};
        }
    }
}

print "Writing averages to file\n";
my $filename = "$dir\\Configuration\\traitAverages.conf";
open(my $CONFIG_FILE, ">", $filename) or makeWarningWindow("Could not open $filename: $!\n");
foreach my $vine (keys %vineToTrait){
    my $shashRef = $VineToTrait{$vine};
    my %traitHash = %$shashRef;
    foreach my $trait (keys %traitHash){
        print $CONFIG_FILE "$vine\t$trait\t$traitHash{$trait}\n";
    }
}
close($CONFIG_FILE);
}

sub getTraitAveragesInfo{
    #Take no parameters
    #returns hash of vine-value pairs
    #Reads in vine trait average info from config file

```

```

my %vineTraitAveragesHash; #key is "vine_trait" value is average
my $filename = "$dir\\Configuration\\traitAverages.conf";
open(my $CONFIG_FILE,"<", $filename) or makeWarningWindow("Could not open $filename:$!\n");
while (eof($CONFIG_FILE)) {
    my $line = readline($CONFIG_FILE);
    chomp $line;
    my @splitLine = split("\t", $line);
    $vineTraitAveragesHash{"$splitLine[0]_splitLine[1]"} = $splitLine[3];
}
return \%vineTraitAveragesHash;
}

sub getAllVines{
    #Takes: Nothing
    #Returns: list of all vines in database.
    #Purpose: Get list of all vines in database
    my $queryString = "select desig from LISTDATA group by desig;";
    my $dbh = DBI->connect("DBI:mysql:database:$host:Sport", $username, "", {RaiseError=>0,PrintError=>1}); #Connect to database
    my $sqlQuery;
    if (defined($dbh)) {
        $sqlQuery = $dbh->prepare($queryString) or die makeWarningWindow("Can't prepare $queryString:". $dbh->errstr); #Prepare the query
    }else{
        makeWarningWindow("Bad Connection Settings. Change in Options Menu" );
        #options_menu();
        return ();
    }
    my $rv = $sqlQuery->execute or die makeWarningWindow("Can't execute the query:". $sqlQuery->errstr); #Execute the query
    my @report = [];
    while (my @row= $sqlQuery->fetchrow_array()) { #get each row of results
        push (@report,\@row); #add pointer to row array to report array
    }
    my @vinelist; #holds list of all viens in the system
    foreach my $row (@report){
        foreach my $value (@$row){
            push(@vinelist, $value);
        }
    }
    return \@vinelist;
}

sub getAllIDs{
    #Takes: Nothing
    #Returns: list of all vine IDs in database.
    #Purpose: Get list of all vine IDs in database
    my @vines = getAllVines();
    my @IDs;
    foreach my $vine (@vines){
        my $id = getIDFromName($vine);
        push (@IDs, $id);
    }
}

```

```

}
return \@IDs;
}
sub generateVineString{
#generate a string for a list of vines. Used to make vineListingWindow strings. Also updates the current vine selections based on synonyms
#takes nothing
#returns the string
my $indexCounter = 1;
my $vineString = "";
my %vinesSeen;
foreach my $name (sort keys %vines){
    foreach my $syn (keys %VineSynonyms){ #checking here to see if $name is a hidden vine name. If it is, replace it with the correct shown name

        if (my $index = (index($syn,"---"))!= -1) {
            my $substring = substr($syn, 0,-3);
            if ($name =~ /$substring.*/) {

                $name = $VineSynonyms{$syn};

            }else{
                #print "$syn does not match $name\n";
            }
        }else{
            #print "$syn does not contain ---\n";
            if ($name eq $syn) {

                $name = $VineSynonyms{$syn};

            }
        }
    }
    if (!exists($vinesSeen{$name})) {
        #VineString = $vineString."$indexCounter)t$name\n";
        $vines{$name} = getIDFromName($name);
        #indexCounter++;
        $vinesSeen{$name} = 1;
    }

}

my @vinesList = keys %vinesSeen;
@vinesList = sort @vinesList;
foreach my $vine (@vinesList){
    $vineString = $vineString."$indexCounter)t$vine\n";
    $indexCounter++;
}

return $vineString;

```

```
}
```

## **Appendix VI: reportMaker2.pl**

```
#!/usr/bin/perl -w

use strict;

use warnings;

use Getopt::Long;

use Tk;

use Scalar::Util qw(looks_like_number);

use Cwd;

use File::Spec::Functions qw(rel2abs);

use File::Basename;

my $dir = dirname(rel2abs($0));

my $helpFlag;

my $averageFlag;

my $analysisFlag;

my $result = GetOptions("help"=> \$helpFlag, "average" => \$averageFlag, "analysis"=> \$analysisFlag);

if (($helpFlag) || ((scalar @ARGV) > 5)) { #If there are not two arguments the person probably doesn't know how to use this
```

```
print <<USAGE;
```

```
    USAGE: perl $0 [input_file] [output_file] [trait file] [study file]
```

```
        perl $0 --help
```

```
        EXAMPLE: perl $0 "C:\\Users\\Seem Lab\\Documents\\data.txt" "C:\\Users\\Seem Lab\\Documents\\Chardonnay_report.txt"
```

```
    NOTE: IT IS NOT LONGER RECCOMENDED TO USE THIS SCRIPT AS A STANDALONE (THOUGH IT IS POSSIBLE). USE
search.pl INSTEAD
```

This script formats a tab delimited file from a MySQL query into a tab delimited report.

It takes file at [input], reformats the data, and prints to file [output].

If the input or output file is not in the same folder as this script you may have to supply the full path.

Example: This script is in C:\\Users\\Seem Lab\\Documents\\Scripts and the data is on C:\\Users\\Seem Lab\\Desktop. You will need to supply "C:\\Users\\Seem Lab\\Desktop\\[input]" as the filename. This includes the quotes (needed if there is a space in the filename)

[trait file] = file containing list of traits to include in the report. One trait per line, exactly as they appear in the input file.

[study file] = file containing list of studies to include in report. Ones tudy per line, exactly as they appear in the input file.

If you get a lot of errors that never stop, you may have saved the MySQL report as a .csv file, even though the file extension is .txt.

#### Options:

--help: Display this message

[year]: Optionally specify a cutoff year. Only trait values observed in studies after (or including) the cutoff year will be reported.

--average: Show averages for all variables for each vine.

--analysis: Format report for downstream analysis. Can only be specified in --average is on.

#### USAGE

```
    exit;  
}
```

```
#####READ IN REPORT#####
```

```
#get comand line args
```

```
my $infile = $ARGV[0];
```

```
my $outfile = $ARGV[1];
```

```
my $traitFile = $ARGV[2];
```

```
my $studyFile = $ARGV[3];
```

```
my $year = 0;
```

```
if (defined($ARGV[4])) {
```

```
    $year = $ARGV[4];
```

```
    if (($year!~/^[0-9]{4}$/) && ($year != 0)) {
```

```
        print "ERROR: UNKNOWN YEAR ENTERED: $year\n";
```

```
        exit();
```

```
    }
```

```
}
```

```
print "Generating Report\n";
```

```
#Initialize and read in file
```

```
my %traits;
```

```
my %seenTraits;
```

```
my %scales;
```

```
my @file = []; #File will be stored in a 2d array. To get a specific value at "row x" and "column y": my $value = $file[x]->[y]
```

```
open(my $TRAITS, $traitFile) or die "Could not open $traitFile: $!\n";
```

```

while (!eof($TRAITS)) {
    my $line = readline($TRAITS);
    chomp $line;
    my @splitLine = split("\t", $line);
    $traits{$splitLine[0]} = 1;
    $scales{$splitLine[0]} = $splitLine[1];
}
close($TRAITS);
my %studies;
open(my $STUDY, $studyFile) or die "Could not open $studyFile: $!\n";
while (!eof($STUDY)) {
    my $line = readline($STUDY);
    chomp $line;
    $studies{$line} = 1;
}
close($STUDY);
#Read in vine synonyms
my %vineSynonyms;
open(my $VINE_FILE, "$dir\\Configuration\\vineNames.conf") or print "Could not open $dir\\Configuration\\vineNames.conf: $!\n";
while (!eof($VINE_FILE)) {
    my $line = readline($VINE_FILE);
    chomp $line;
    my @splitLine = split("=", $line);
    if ((scalar @splitLine) != 2) {
        print "ERROR: Formatting error in vineNames.conf\n";
    }else{
        $vineSynonyms{$splitLine[0]} = $splitLine[1];
    }
}
close ($VINE_FILE);
open(my $FILEHANDLE, $infile) or die ("Could not open $infile: $!\n");
while (!eof($FILEHANDLE)) {
    my $line = readline($FILEHANDLE); #read one line from the input file
    chomp $line;
    if ($line ne "") {

```

```
my @tempLine = @ {processLine($line)}; #get the resulting array from the processLine subroutine
```

```
#Sometimes the line doesn't have all of the columns in it and they hang over onto next line. Sometimes there is even an extra blank line in between. Its dumb
```

```
while (((scalar @tempLine) < 7) &&!eof($FILEHANDLE)) { #As long as my line doesnt have 7 values, keep reading and adding values
```

```
    my $nextLine = readline($FILEHANDLE);  
    if ($nextLine ne "") {  
        my @nextTempLine = @ {processLine($nextLine)};  
        push (@tempLine,@nextTempLine);  
    }  
}
```

```
##end while
```

```
foreach my $key (keys %vineSynonyms){
```

```
    if (my $index = (index($key,"---"))!=-1) {  
        my $substring = substr($key, 0,-3);  
        if ($tempLine[0] =~ /$substring.*/) {  
            $tempLine[0] = $vineSynonyms{$key};  
        }  
    }else{  
        if ($tempLine[0] eq $key) {  
            $tempLine[0] = $vineSynonyms{$key};  
        }  
    }  
}
```

```
}
```

```
#Check if the study name on tempLine exists in our allowed studies. If not, check if the study name is just a year. If that year matches the year of the study, add the data
```

```
if (exists($traits{$tempLine[3]})){  
    if (exists($studies{$tempLine[5]})) {  
        push(@file,\@tempLine);#add this line to the file  
    }else{
```





```

my $outputString = ""; #initialize string that will hold entire output string
my $headerString = "";
my $footerString = "";

my @studyList; #Hash to organize studies by their year.
my $vines = getUniqueVines(0); #The unique vine names in this report
my %traitMasterListNumeric; #Used for analysis reports. Holds master list of all numeric traits seen
my %traitMasterListText; #Used for analysis reports. Holds master list of all text traits seen
my %vineTraitValue; #Used for analysis reports. Holds hash where "vine_name" -> hash where "traitName" -> value
my %analysisFullHash;
my %vineCross; #Hash to hold vine and cross for analysis format

my @sortedVines = sort @$vines;
my $avgHeaderCount = 0; #See line 186
foreach my $vine (sort @sortedVines){ #Iterate over each vine. Each vine has a separate section of the resulting report
    my $cross = getCross($vine); #get parents
    $vineCross{$vine} = $cross;
    my $line="";
    if (!$saverageFlag) {
        $line = "Vine,Cross\n$vine,$cross\n\nStudy,Traits\n"; #Create the string for the vine and cross portions of the report. , is a tab. \n is a
        newline
    }elseif(!$analysisFlag && $saverageFlag){
        if ($avgHeaderCount == 0) {
            #Only put "Vine,Cross, at the top of the first entry
            $line = "Vine,Cross\n";
        }
    }

    }

if (!$analysisFlag) {
    $outputString = $outputString.$line"; #append line to output string
}

my $studies = getUniqueStudies($vine); #get all unique studies for this vine

```

```

my %traitAverages; #Holds the average trait value. Keys are [trait_name] -> running total. [trait_name]_avg -> total observations of this trait.
my %traitCounts; #Holds the amount of times each numeric trait was seen
my %stringTraits; #Holds an instance of each string trait. will only take the first returned value for each.
my %stringTraitYear; #Holds the most recent year observed for a string trait
my %stringStudy; #Store the study that each string trait value came from.

my %vineHash;
my @sortedStudies = sort @$studies;

foreach my $study(sort @sortedStudies){ #iterate over the studies. Each study gets its own subsection in each vine
    $study =~ m/.*{[0-9]{4}}.*#/;

    my $studyYear = $1;

    if (!defined($studyYear)) {
        $studyYear = 1; #Will include studies if year cannot be identified and no year supplied. Will preclude studies if year is supplied and
        studies cannot be identified.
    }

    if (((($study !~ /.*[1-9]/) && ($studyYear >= $year))) { #Do not report studies with format [study_name]#[numbers]. These are
        deleted studies that should not be reported on.

        if (!$averageFlag && !$analysisFlag) {
            my $line = ",$study"; #Add the study name to the outoustring
            $outputString = $outputString.$line; #append the new line
        }

        my $relatedIdentifiers = getUniqueIdentifiers([$vine,$study]); #Get all related identifiers for the study
        my %traits; #Start new hash of traits for the study. This is an easy way to keep all unique traits together
        foreach my $relatedIdentifier (@$relatedIdentifiers){ #Count total traits observed in study for this vine. Do not assume all reps have
        same traits

            my $traits = getTraits ($vine,$study,$relatedIdentifier); #Get all traits for each related_identifier
            foreach my $trait (@$traits){ #Check to see if trait is in hash. If it is not, add it to hash
                if (!exists($traits{$trait})) {
                    $traits{$trait} = 1;
                }

                if (($averageFlag)) { #If calculating averages and the trait looks like a number
                    if ((looks_like_number(getValue([$vine,$study,$relatedIdentifier,$trait]))) {

                        if (exists($traitAverages{$trait})) {

```

```

        $traitAverages{ $trait } = $traitAverages{ $trait } + getValue([$vine,$study,$relatedIdentifier,$trait]);
        $traitCounts{ $trait } = $traitCounts{ $trait }+1;
    }else{
        $traitAverages{ $trait } = getValue([$vine,$study,$relatedIdentifier,$trait]);
        $traitCounts{ $trait } = 1;
    }
}elseif(getValue([$vine,$study,$relatedIdentifier,$trait]) ne ""){
    $study =~ m/.* *([0-9]{4}) *.*;/ #Get year
    if (exists($stringTraits{ $trait})) {
        if (defined($1)) {
            if ($1 > $stringTraitYear{ $trait}) { #if year is greater than current largest year, update year and
value
                $stringTraitYear{ $trait } = $1;
                $stringTraits{ $trait } = getValue([$vine,$study,$relatedIdentifier,$trait]);
                $stringStudy{ $trait } = $study;
            }
        }
    }

}elseif(!exists($traitAverages{ $trait})){
    $stringTraits{ $trait } = getValue([$vine,$study,$relatedIdentifier,$trait]);
    $stringStudy{ $trait } = $study;
    if ((defined($1))) {
        $stringTraitYear{ $trait } = $1;
    }else{
        $stringTraitYear{ $trait } = 0; #No year
    }
}
"a" =~ /a/; #reset $1. Seriously don't delete this.
} ##Else the value is nothing
}
}
}

if (!$averageFlag) {

```

```

my @traitList = keys %traits; #get list of all traits in study
for(my $traitCounter = 0; $traitCounter < scalar @traitList; $traitCounter++){ #Iterate over list of traits. Check to see if
the trait is empty in all related identifiers.

    my $emptyCheck = checkIfEmptyValue([$vine,$relatedIdentifiers,$traitList[$traitCounter],$study]); #Check if
trait is empty in all related identifiers

    if ($emptyCheck == 0) { #Zero is returned if trait is empty
        delete $traits{$traitList[$traitCounter]}; #remove trait from hash
    }
}

@traitList = keys %traits; #Re-generate trait list from hash. Will be smaller than before b/c of deleted items
@traitList = sort @traitList;

my $dateIndex = checkDate(@traitList); #See if there is a "DATE" trait observed. will be far left trait in study if it exists
my $notesIndex = getNotes(@traitList); #See if there is a "notes" trait. Will be far right trait if there is one.
if ($dateIndex != -1) { #If there is a date, add it to the report first

    $outputString = $outputString.",$traitList[$dateIndex]"; #add in date column to output string
}

for (my $counter = 0; $counter < scalar @traitList; $counter++){ #Iterate over all traits to add them to the report

    if (($counter == $dateIndex) || ($counter == $notesIndex)) { #Skip if I come to notes or date trait
        next; #skip
    }

    $outputString = $outputString.",$traitList[$counter]"; #Add each trait to growing outputstring
}

if ($notesIndex != -1) { #If there is a notes field, add it last

    $outputString = $outputString.",$traitList[$notesIndex]"; #Add notes field
}

$outputString = $outputString."\n"; #add newline to outputstring

foreach my $relatedIdentifier (@$relatedIdentifiers){ #For each identifier...here I start to add values for each repetition
in the study

    my %tempHash;

    my @values; #Initialize list of values

    foreach my $trait (@traitList){ #For each trait in this study...

```

my \$value = getValue([\$vine,\$study,\$relatedIdentifier,\$trait]); #Get the value for the unique vine, study,  
trait, related identifier tuple.

```
if ($value ne "") {
    $seenTraits{$trait} = 1;
}

$tempHash{$trait} = $value;
push (@values,$value); #add value to list of values
}
$vineHash{$relatedIdentifier} = \%tempHash;
if ($dateIndex != -1) { #If there is a date column
    $outputString = $outputString.".,$values[$dateIndex]"; #add the date value to the output
}
my $counter = 0;
for ($counter = 0; $counter < scalar @values; $counter++){ #Iterate over the traits
    if ($counter == $dateIndex || $counter == $notesIndex) { #If I hit the date or notes column, skip it
        if (($counter == $notesIndex) && $counter == 0) {
            $outputString = $outputString.".";
        }

        next;
    }if(($counter == 0) && ($dateIndex == -1)){
        $outputString = $outputString.".";
    }
    if ($dateIndex != -1) {
        #code
    }

    $outputString = $outputString.".,$values[$counter]"; #Add the value to the output string
}

if ($notesIndex != -1) { #If There is a notes column add it now
    $values[$notesIndex] =~ s/,//g;
    if ($counter == 0) {
        $outputString = $outputString.".";
    }
}
```

```

    }

    $outputString = $outputString.",$values[$notesIndex]"; #add notes value
}
$outputString=$outputString."
"; #Add a newline in between related_identifiers (study repetitions)
}
$analysisFullHash{$vine} = \%vineHash;

}#End of not avergae flag check
}
if(!$averageFlag && !$analysisFlag){
    $outputString=$outputString."
"; #Add a new line in between study sections
}

"a" =~ /a/; #Don't delete. resets $1
}
if ($averageFlag) {
    my @traitList = keys %traitAverages;
    my @stringTraits = keys %stringTraits;
    my @valueList;
    @traitList = sort @traitList;
    @stringTraits = sort @stringTraits;
    my %tempHash;
    foreach my $trait(@traitList){
        my $value = ($traitAverages{$trait}/$traitCounts{$trait})*1.00;
        if (!exists($traitMasterListNumeric{$trait})){
            $traitMasterListNumeric{$trait} = 1;
        }
        $tempHash{$trait} = $value;
        push(@valueList,$value);
    }
    foreach my $stringTrait(@stringTraits){
        my $value = $stringTraits{$stringTrait};
        $value =~ s/,//g; #remove commas from string traits
    }
}

```

```

        if (!exists($traitMasterListText{$stringTrait})){
            $traitMasterListText{$stringTrait} = 1;
        }
        $tempHash{$stringTrait} = $value;
        push(@valueList,$value);
    }

    $vineTraitValue{$vine} = \%tempHash;
    if (!$analysisFlag) {
        $outputString = $outputString."$vine,$cross,";
        foreach my $trait (@traitList){
            $outputString = $outputString.",$trait n=$traitCounts{$trait}";
        }
        foreach my $stringTrait(@stringTraits){
            $outputString=$outputString.",$stringTrait ($stringStudy{$stringTrait})";
        }
        $outputString = $outputString."\n,,,";
        foreach my $value(@valueList){
            $outputString = $outputString."$value,";
        }
    }

}

if (!$analysisFlag) {
    $outputString=$outputString."\n"; #add two new lines in between vine sections
}

$avgHeaderCount++;
}#No more vines.
if ($analysisFlag) {
    if ($averageFlag) {

        my @numericTraits = keys %traitMasterListNumeric;
        my @stringTraits = keys %traitMasterListText;
        @numericTraits = sort @numericTraits;
    }
}

```



```

@stringTraits = sort @stringTraits;
$outputString = "VINE_NAME,CROSS,";
foreach my $numericTrait (@numericTraits){
    $outputString = $outputString.$numericTrait,";
}
$outputString = $outputString.",,";
foreach my $stringTrait (@stringTraits){
    $outputString = $outputString.$stringTrait,"
}
$outputString = $outputString."\n";
my @vines = keys %vineTraitValue;
@vines = sort @vines;
foreach my $vine (@vines){
    $outputString = $outputString.$vine,$vineCross{$vine},";
    foreach my $numericTrait (@numericTraits){
        if (exists($vineTraitValue{$vine}->{$numericTrait})) {
            $outputString = $outputString.$vineTraitValue{$vine}->{$numericTrait},";
        }else{
            $outputString = $outputString."NA,"
        }
    }
}
$outputString = $outputString.",,";
foreach my $stringTrait(@stringTraits){
    if (exists($vineTraitValue{$vine}->{$stringTrait})) {
        $outputString = $outputString.$vineTraitValue{$vine}->{$stringTrait},";
    }else{
        $outputString = $outputString."NA,"
    }
}
$outputString = $outputString."\n";
}
}else{
$outputString = "VINE_NAME,CROSS,";
my @numericTraits = sort keys %seenTraits;

```

```

foreach my $numericTrait (@numericTraits){
    $outputString = $outputString.$numericTrait.", ";
}
$outputString = $outputString."\n";
foreach my $vine (keys %analysisFullHash){
    foreach my $identifier (keys %{$analysisFullHash{$vine}}){
        $outputString = $outputString."$vine,$vineCross{$vine},";
        foreach my $numericTrait (@numericTraits){
            if (exists($analysisFullHash{$vine}{$identifier}{$numericTrait})) {
                $outputString = $outputString."$analysisFullHash{$vine}{$identifier}{$numericTrait},";
            }else{
                $outputString = $outputString."NA,";
            }
        }
        $outputString = $outputString."\n";
    }
}
}
}

```

#####DONE PROCESSING REPORT #####

##### #OUTPUT #####

```

if (!$analysisFlag) {
    $outputString = $outputString.getTraitDescriptions(); #Get the list of all traits and trait descriptions in the report
}

```

open(my \$OUTFILE, ">>", \$outfile) or die makeWarningWindow("could not open \$outfile: \$!\n"); #Open output file

print \$OUTFILE \$outputString; #Print to outut file

close(\$OUTFILE); #Close output file

#gg

#####END OF MAIN FUNCTION#####

#####HELPER FUNCTIONS#####

```

sub processLine{
    #Arguments: $line -- A String to format
    #Returns @tempLine -- An array, where each element in the array is a value from the line
    #Purpose: Given a raw line from the file, return an array reference of all the values in the line

    my ($line) = @_;
    chomp($line);
    $line =~ s/"//g;
    $line =~ s/\n+//g;
    $line =~ s/\r+//g;
    $line =~ s/[\n\r]+//g;
    $line =~ s/^\t//g;

    my @tempLine = split("\t", $line);
    return(\@tempLine);
}

```

```

sub getUniqueVines{
    #Arguments: $column -- The number of the column to perform this routine on
    #Returns: A list of unique values from the column
    #Purpose: Used for getting unique vine names

    my ($column) = @_;
    my %uniqueHash; #Used to quickly check if unique name has been seen
    my @uniqueArray; #Stored list of seen uniques for easy iteration

    for my $j (1..$#file) {

        if (!exists($uniqueHash{$file[$j]->{$column}})) {
            $uniqueHash{$file[$j]->{$column}} = 1;
            push(@uniqueArray, $file[$j]->{$column});
        }
    }

    return \@uniqueArray;
}

```

```

sub getUniqueStudies{
    #Arguments: $vine -- The name of a vine

```

```

#Returns: \@uniqueArray -- An array reference to all of the unique studies the vine was used in
#Purpose: Given the name of a vine, return all of the unique trials it was used in
my ($vine) = @_;
my %uniqueHash;
my @uniqueArray;
for my $j (1..$#file) {
    if ((!exists($uniqueHash{$file[$j]->[5]}))&&($file[$j]->[0] eq $vine)) {
        $uniqueHash{$file[$j]->[5]} = 1;
        push(@uniqueArray, $file[$j]->[5]);
    }
}
return \@uniqueArray;
}

sub getUniqueIdentifiers{
    #Arguments: $vine -- The name of a vine, a string
    #           $study -- The name of a study, a string
    #Returns: \@uniqueArray -- A reference to an array containing all of the related identifiers associated with this vine and study
    #Purpose: Given a vine and a study name, get all of the related identifiers associated with them
    my ($checks) = @_;
    my $vine = @$checks[0];
    my $study = @$checks[1];
    my %uniqueHash;
    my @uniqueArray;
    for my $j (1..$#file) {
        if ((!exists($uniqueHash{$file[$j]->[6]}))&&($file[$j]->[0] eq $vine) && ($file[$j]->[5] eq $study)) {
            $uniqueHash{$file[$j]->[6]} = 1;
            push(@uniqueArray, $file[$j]->[6]);
        }
    }
    return \@uniqueArray;
}

sub getTraits{

```

```

#Arguments: $vine -- The name of a vine, a string
#           $study -- The name of a study, a string
#           $identifier -- A related identifier, an integer (or string. There is no difference in perl)
#Returns: \@uniqueArray -- A reference to an array containing all of the unique traits associated with the vine, related identifier, and study
#Purpose: Given a vine, and identifier and a study name, get all of the traits associated with them

my ($checks) = @_;
my $vine = @$checks[0];
my $study = @$checks[1];
my $identifier = @$checks[2];

my %uniqueHash;
my @uniqueArray;

for my $j (1..$#file) {
    if (!(exists($uniqueHash{$file[$j]->[3]}))&&($file[$j]->[0] eq $vine) && ($file[$j]->[5] eq $study) &&($file[$j]->[6] eq $identifier))
    {
        $uniqueHash{$file[$j]->[3]} = 1;
        push(@uniqueArray, $file[$j]->[3]);
    }
}

return \@uniqueArray;
}

sub getValue{
    #Arguments: $vine -- The name of a vine, a string
    #           $study -- The name of a study, a string
    #           $identifier -- A related identifier, an integer (or string. There is no difference in perl)
    #           $trait -- The name of a trait
    #Returns: \@uniqueArray -- The value of the trait for that vine, study, and identifier, or an empty string
    #Purpose: Given a vine, study, identifier, and trait, return the value associated with those
    #NOTE: You may not need to pass the study name, and can remove it as a check. The related identifier, vine, and trait should be enough to
    uniquely identify the value

    my ($checks) = @_;
    my $vine = @$checks[0];
    my $study = @$checks[1];
    my $identifier = @$checks[2];

```

```

my $trait = @$checks[3];
for my $j (1..$#file) {
    if (($file[$j]->[0] eq $vine) && ($file[$j]->[5] eq $study) && ($file[$j]->[6] eq $identifier) && ($file[$j]->[3] eq $trait)) {
        $file[$j]-> [2] =~ s/./:/g;
        return($file[$j]-> [2]);
    }
}
return (""); #Could probably be "" as the return value. I don't think it matters.
}

sub checkDate{
    #Arguments: @traits -- A list of trait names
    #Returns: $counter -- The index of a variable with "DATE" in it, or a negative number if there is no trait that matches this. THIS is just a
    wrapper function to apply index() to an array
    #Purpose: Given a list of trait names, return the index (or a negative number) or the trait which has "DATE" in it
    my (@traits) = @_;
    for(my $counter = 0; $counter < scalar @traits; $counter++){
        if (index($traits[$counter],"DATE") !=-1) {
            return ($counter);
        }
    }
    return (-1);
}

sub getNotes{
    #Arguments: @traits -- A list of trait names
    #Returns: $counter -- The index of a variable with "NOTES" in it, or a negative number if there is no trait that matches this. THIS is just a
    wrapper function to apply index() to an array
    #Purpose: Given a list of trait names, return the index (or a negative number) or the trait which has "NOTES" in it
    #NOTE: You can probably combine this with the function above (checkDate), pass the array as a reference, and the string of the thing to
    check, like "DATE" or "NOTES". There is no reason to have two functions. I should not have wrote it this way
    my (@traits) = @_;
    for(my $counter = 0; $counter < scalar @traits; $counter++){
        if ((index($traits[$counter],"NOTES") !=-1) || (index($traits[$counter],"COMMENTS") !=-1) ) {

```

```

        return ($counter);
    }

}

return (-1);

}

sub getCross{
    #Arguments: $vine -- A string representation of the vine name
    #Returns: The cross of the vine as a string
    #Purpose: Input the vine name, get the cross from the report
    my ($vine) = @_;
    for my $j (1..$#file){
        if ($file[$j]->[0] eq $vine) {
            return $file[$j]->[1];
        }
    }
    return ("");
}

sub getTraitDescriptions{
    #Returns: A string of trait nmes and their descriptions
    #Purpose: Get a formatted string of all the traits in this report and their descriptipons
    my %uniqueHash;
    my @uniqueArray;
    my %traitToDesc;
    for my $j (1..$#file) {
        if (!(exists($uniqueHash{$file[$j]->[3]}))) {
            $uniqueHash{$file[$j]->[3]} = 1;
            push(@uniqueArray, $file[$j]->[3]);
            $traitToDesc{$file[$j]->[3]} = $file[$j]->[4];
        }
    }
}

```

```

my $string = ",Trait,Definition,Scale\n";
@uniqueArray = keys %traitToDesc;
@uniqueArray = sort @uniqueArray;
for (my $j =0; $j < scalar @uniqueArray; $j++){
    $traitToDesc{$uniqueArray[$j]} =~ s//g;
    if (defined($scales{$uniqueArray[$j]})) {
        $string = $string.",$uniqueArray[$j],$traitToDesc{$uniqueArray[$j]},$scales{$uniqueArray[$j]}\n\n";
    }else{
        $string = $string.",$uniqueArray[$j],$traitToDesc{$uniqueArray[$j]}\n\n"
    }
}
}
return($string)
}

```

```

sub checkIfEmptyValue{

```

```

    #Arguments:

```

```

    #           $vine -- The name of a vine

```

```

    #           $identifiers -- An array reference to a list of identifiers

```

```

    #           $trait -- The name of a trait

```

```

    #           $study -- The name of a study

```

```

    #Returns: 1 if there is a value, or 0 if there is not

```

#Purpose: Checks if any of the related identifiers for a given study, vine, trait, have a value associated with them. This eliminates the printing of traits with no values for any related identifier. This happens because of an artifact of the BMS. Usually, empty traits should not be here

#By artifact, I mean that if you reupload data to a trial/nursery and is now no data where data used to be, there will be a variable with no value associated with it in the MySQL report.

```

    my ($checks) = @_;

```

```

    my $vine = @$checks[0];

```

```

    my $identifiers = @$checks[1];

```

```

    my $study = @$checks[3];

```

```

    my $trait = @$checks[2];

```

```

    foreach my $identifier (@$identifiers){

```



```

my $value = getValue([$vine,$study,$identifier,$trait]);
if ($value ne "") {
    return(1);
}

}

return(0);
}

sub makeWarningWindow{
    my ($message) = @_;
    my $WarningWindow = new MainWindow;
    $WarningWindow->title("ERROR");
    my $warningFrame = $WarningWindow -> Frame();
    my $labMessage = $warningFrame -> Label(-text=>"$message");
    my $quitButton = $warningFrame -> Button(-text=> "OK", -command => sub{$WarningWindow->destroy;});
    $warningFrame -> grid(-row => 1, -column => 1, -columnspan => 3);
    $labMessage -> grid(-row=>1,-column=>1);
    $quitButton -> grid(-row=>3, -column =>1);

}

```

## **Appendix VII: nurseryFormat.pl**

```

#!/usr/bin/perl -w
use strict;
use warnings;
use Getopt::Long;

my $helpFlag;
my $entryType = "T";
my $result = GetOptions("help" => \$helpFlag, "entrytype=s" => \$entryType);

#If exactly 3 arguments are not supplied to the program, the Description message is displayed.
#Optionally, if --help is supplied as an option, the help message will be displayed
#For all the scripts I write, if --help is entered as an option, the help message will be displayed.

```

```
if (($helpFlag) || (scalar @ARGV < 3)) {
```

```
    print <<USAGE;
```

```
Usage:
```

```
    perl $0 [input spreadsheet file] [year] [vineyard number]
```

Description:

NOTE: NOT WORKING YET. NEEDS TO BE TWEAKED

Produces a nursery fieldbook based on the input excel spreadsheet file. The input file is a BMS excel book, saved as a Tab Delimited file.

Expected input is the observation sheet from the excel fieldbook that the BMS produces for Nurseries.

You should have sorted the data by the Row and Group column, replaced the "/" separating the parents with " X ", and saved this sheet as a text file.

The text file is the expected input for this script.

The output is the name of the file, with "\_NurseryBook.txt" appended to it. The file will appear in the directory the input file is in.

Note that the script will append to the output file. Thus if a file already exists under the output name, it will attempt to append to the end of it.

This will be blocked if the file is open in word, and appending to the end of an existing file is probably not what you want anyways.

USAGE

```
    exit;
```

```
}
```

```
#These 3 variables hold the options entered on the command line
```

```
my $spreadsheet = $ARGV[0]; #The full path of the spreadsheet file
```

```
my $year = $ARGV[1]; #Year
```

```
my $vineyardNumber = $ARGV[2]; #Vineyard number
```

```
open(my $INPUT, $spreadsheet) or die "Could not open file:$!\n"; #Open the input file for reading
```

```
my @file = []; #The entire file is stored in a 2D array. Thus a specific value can be accessed by calling $file[ROW_NUM][COLUMN_NUM]
```

```
while (!eof) { #read until I get to the end of the file
```

```
    my $line = readline($INPUT); #read a line from the file, and store it in $line
```

```
    chomp($line); #Remove any trailing whitespace or newline characters
```

```
    my @splitLine = split("\t", $line); #Split the line on tab characters. This gives me a list of all of the fields in the current row
```

```
    $line =~ s//g; #Remove quotation marks. Sometimes the row numbers for the vines are in quotes. This is a regular expression. The general form is /[PATTERN TO MATCH]/[REPLACEMENT PATTERN]/. The "s" denotes search and replace. The "g" denotes that I want to replace every occurrence of the quote
```

```
    $line =~ s//g;
```

```

    push(@file,\@splitLine); #add this row to my growing file
}

close($INPUT); #Close the input file

my $rowNum; #Will hold the column number for the row
my $repNum; #Will hold the column number for the number of replicates
my $notesNum;
my $plotNum;
my $nameNum;
my $crossNum;
my %plotToRow; #Hash maps row and plot to rows in file;

    for (my $j = 0; $j < @{$file[1]}; $j++) { #Loop over the columns. When a column header matches one of the strings below, store the column
number in the appropriate variable

        if($file[1][$j] eq "VINE_ROW"){
            $rowNum = $j;
            print $rowNum."\n";
        }elseif($file[1][$j] eq "NOTES"){
            $notesNum = $j;
        }elseif($file[1][$j] eq "VINE_NUMBERS"){
            $plotNum = $j;
        }elseif(($file[1][$j] eq "VINE_NAME") || ($file[1][$j] eq "DESIGNATION")){
            $nameNum = $j
        }elseif($file[1][$j] eq "CROSS"){
            $crossNum = $j
        }

    }

for (my $i = 2; $i < scalar(@file); $i++) {
    my $row = $file[$i][$rowNum];
    my $rowLength = scalar @{$file[$i]};
    #print $file[$i][0]."\t".$row."\t".$rowLength."\n";
    my @plots = expandPlot($file[$i][$plotNum]);
    foreach my $plot(@plots){

```

```

#print $plot."n";
my $key = $row*1000000+$plot;
my $temp = $file[$i];
@{$temp}[$plotNum] = $plot;
$plotToRow{$key} = $temp;
}

}

my $outString = ""; #This will hold the entire fieldbook, and will be generated piece by piece
my $currentRow = 1; #initialize counting variables that are used for formatting purposes
my $currentPlot = 1;
my $Counter = 1;
my @keys = sort {$a <=> $b} keys %plotToRow;
for (my $i = 0; $i < scalar(@keys); $i++) { #Loop over each row in the file. Ignore the header row
    if($plotToRow{$keys[$i]}[$rowNum] != $currentRow){ #If the row I am looking at is different than the previous row, I need to reset some
counters, and update the current row to be the one I am looking at
        $currentRow = $plotToRow{$keys[$i]}[$rowNum] ; #Update the current row to the one I am looking at
        $currentPlot = 0; #reset necessary formatting counters to indicate I am processing a new row
        $Counter = 1;
    }
    $keys[$i] =~ m/([1-9]+)0+([0-9]+)/;
    my $actualPlot = $2;
    #Iterate the following process for the amount of replicates specified
    if (($currentPlot == 0) && ($i != 0)) { #If I just started a new row, insert a new page
        $outString = $outString."f";#Insert a page break
    }

    if ($Counter == 1) { #If this is the first plot of the page, print the page header
        $outString = $outString."VINE\t VINEYARD\tvineyardNumber\t ROW\t$currentRow\t YEAR\t$year\n\n";
    }
    elsif($Counter == 9){ #If this is the last plot of the page, reset the plot counter for the page
        $Counter = 0;
    }
}

```

```

    $outString = $outString.$actualPlot."\\t".$plotToRow{$keys[$i]}[$nameNum]."\\t\\t".$plotToRow{$keys[$i]}[$crossNum]; #Print the
germplasm, and parents

    if (exists($plotToRow{$keys[$i]}[$notesNum])) { #If there is a notes field for this germplasm, I need to print it. If there isn't move on

        #Print the notes line, and the line with all of the variables to collect

        $plotToRow{$keys[$i]}[$notesNum] = substr($plotToRow{$keys[$i]}[$notesNum],0,60);

        $outString = $outString."\\nNOTES: $plotToRow{$keys[$i]}[$notesNum]\\n T B FL DM PM CR VG CS CC BS BC SD FL
TX S C R";

    }else{ #There are no notes, just leave the space after "Notes" blank

        #Print the notes line, and the line with al of the variables to collect

        $outString = $outString."\\nNOTES:\\n T B FL DM PM CR VG CS CC BS BC SD FL TX S C R"; #Print the notes (if there
are any) and print the variabls to collect

    }

    if($Counter!= 0){

        $outString = $outString."\\n\\n-----\\n";

    }else{

        $outString = $outString."\\f";

    }

    $currentPlot++; #Increment format counters

    $Counter++;

}

my @name = split(/\\./,$spreadsheet); #Split the name of the input file on the period

my $outName = $name[0]."_SeedlingBook.txt"; #Append the new ending to the filename

open(my $OUTFILE, ">>",$outName) or die "Could not open $outName: $!\\n"; #Open the output file

print $OUTFILE $outString."\\n"; #Write the output fieldbook to the file

close($OUTFILE);

sub expandPlot{

    my ($string) = @_ ;

    my @numbers;

    my @parts = split(";", $string);

    foreach my $part (@parts){

        if (index($part, "-")!=-1) {

```

```

my @range = split("-", $part);
for (my $counter = $range[0]; $counter <= $range[1]; $counter++){
    $counter =~ s/^0*(\d+)/$1/;
    push(@numbers, $counter);
}
} else {
    $part =~ s/^0*(\d+)/$1/;
    push @numbers, $part;
}
}
return(@numbers);
}

```

### **Appendix VIII: 2ndtestFormat.pl**

```

#!/usr/bin/perl -w

use strict;

use warnings;

use Getopt::Long;

my $helpFlag;

my $entryType = "T";

my $result = GetOptions("help" => \$helpFlag, "entrytype=s" => \$entryType);

#If exactly 3 arguments are not supplied to the program, the Description message is displayed.

#Optionally, if --help is supplied as an option, the help message will be displayed

#For all the scripts I write, if --help is entered as an option, the help message will be displayed.

if (($helpFlag) || (scalar @ARGV != 3)) {

    print <<USAGE;

```

Usage:

```
perl $0 [input spreadsheet file] [year] [vineyard number]
```

Description:

Produces a field notebook for a second-test nursery.

Expected input is a vineyard report for the search tool, saved with "Analysis Format" box only checked.

You should have sorted the data by the Row and Group column, replaced the "/" separating the parents with " X ", and saved this sheet as a text file.

The text file is the expected input for this script.

The output is the name of the file, with "\_NurseryBook.txt" appended to it. The file will appear in the directory the input file is in.

Note that the script will append to the output file. Thus if a file already exists under the output name, it will attempt to append to the end of it.

This will be blocked if the file is open in word, and appending to the end of an existing file is probably not what you want anyways.

Note that if a more than 7 vines exist in a block together, only the first 7 numbers of the block will appear in the fieldbook.

USAGE

```
    exit;
}

#The values given on the command line for the input spreadsheet, year, and vineyard number are stored in these variables.

my $spreadsheet = $ARGV[0]; #Full path of the input spreadsheet

my $year = $ARGV[1]; #Year

my $vineyardNumber = $ARGV[2]; #Vineyard number

#Open the file for reading.
```

```

open(my $INPUT, $spreadsheet) or die "Could not open file:$!\n";

my @file = []; #The entire file is stored in a 2D array. Thus a specific value can be accessed by calling $file[ROW_NUM][COLUMN_NUM]

while (!eof) { #While I am not at the end of the file

    my $line = readline($INPUT); #Read in a single line from the file

    chomp($line); #Remove any trailing whitespace characters, as well as the newline character

    $line =~ s//g; #Remove quotation marks. Sometimes the row numbers for the vines are in quotes. This is a regular expression. The general
form is /[PATTERN TO MATCH]/[REPLACEMENT PATTERN]/. The "s" denotes search and replace. The "g" denotes that I want to replace
every occurrence of the quote

    $line =~ s//g;

    my @splitLine = split("\t", $line); #Split the line based on the tab character. This produces a list where each element in the list is a value from
the spreadsheet

    push(@file, \@splitLine); #Add this list to my list of files

}

close ($INPUT); #Close the input file

# I calculate the column numbers for the Row, plot, notes, and source. I cannot assume that these column numbers will be the same for every
input spreadsheet

my $rowNum;

my $plotNum;

my $notesNum;

my $sourceNum;

my $vineNum;

my $crossNum;

my %plotToRow; #Maps plot numbers to row in file for sorting

#When a column header matches one of these fields, I store the number for that column in the appropriate variable.

for (my $j = 0; $j < @{$file[1]}; $j++) {

```



```

if ($file[1][$j] eq "VINE_NUMBERS") {

    $plotNum = $j;

}elsif($file[1][$j] eq "VINE_ROW"){

    $rowNum = $j;

}elsif($file[1][$j] eq "NOTES"){

    $notesNum = $j;

}elsif(($file[1][$j] eq "SOURCE") || ($file[1][$j] eq "ACCESSION_NUMBER)){

    $sourceNum = $j

}elsif(($file[1][$j] eq "VINE_NAME") || ($file[1][$j] eq "DESIGNATION")){

    $vineNum = $j

}elsif($file[1][$j] eq "CROSS"){

    $crossNum = $j

}

}

for(my $i =2; $i < scalar @file; $i++){

    my $row = $file[$i][$rowNum];

    my @plots = expandPlot($file[$i][$plotNum]);

    my $key = $row*1000000+$plots[0];

    $plotToRow{$key} = $file[$i];

}

my $outString = ""; #This variable holds the entire fieldbok in it, and is built piece by piece. The form $outString = $outString.[something]
appends [something] to the end of the string; thus the string can be grown and added to as I choose.

#These variables are counters I use for formatting purposes

my $currentRow = 1;

my $currentPlot = 1;

```

```

my $Counter = 1;

my $pagePlot = 0;

my @keys = sort {$a <=> $b} keys %plotToRow;

for (my $i = 0; $i < scalar(@keys); $i++) { #Ignore the heading column, and loop over every row in the file

    print $keys[$i]."\n";

    if($plotToRow{$keys[$i]}[$rowNum] != $currentRow){ #if this row does not match the last

        $currentRow = $plotToRow{$keys[$i]}[$rowNum];#Get the row for the next plot

        $currentPlot = 0; #reset the necessary formatting counters

        $Counter = 1;

    }

    #Check to make sure the row for this entry is the same as the last

    if ($currentPlot == 0) { #If this is the first plot of this row, I might need to insert a page break

        if($pagePlot!= 4){ #If there were not 4 plots on the previous page, I need to insert a page break

            $outString = $outString."\f"; #Insert a page break

        }

    }

    if($pagePlot== 3){ #If I have plotted 4 times (it looks like 3, but it is 4), reset the counter for plots on the page

        $pagePlot = 0;

    }

    $pagePlot++; #Incremenet the counter for plots on the page

    if ($Counter == 1) { #COUNTER tracks when I should print the header. I need to print the header here

```

```
$outString = $outString."VINE\t VINEYARD\t$vineyardNumber\t ROW\t$currentRow\t YEAR\t$year\n\n"; #Append the header to  
the growing fieldbook
```

```
}
```

```
elseif($Counter == 4){ #Reset the header counter if I have put 4 plots down
```

```
$Counter = 0;
```

```
}
```

```
#This section creates a list of numbers based on the row number input. Note that it can only handle numbers in one of 3 forms: a single  
number, 2 numbers separated by a comma (2,3), or two numbers separated by a dash (2-6)
```

```
my $var = $plotToRow{$keys[$i]}{$plotNum}; #Isolate the row numbers for this row in the spreadsheet
```

```
my @nums = expandPlot($var);
```

```
for(my $counter = 0; $counter < scalar @nums; $counter++){
```

```
$nums[$counter] =~ s/^0+//g;
```

```
}
```

```
$outString = $outString."$nums[0]\t".$plotToRow{$keys[$i]}{$vineNum}."\t\t".$plotToRow{$keys[$i]}{$crossNum};#Print the first  
number in the array, the cultivar name, and its parents.
```

```
#The remaining print statements each have two options depending on if there are more plot numbers left to print. If there are more plot  
numbers left to print, the first option is used. Else the second option is used
```

```
if (scalar(@nums > 1)) {
```

```
$outString = $outString."\n$num[1]\tNOTES:";
```

```
}else{
```

```
$outString = $outString."\n\tNOTES:";
```

```
}
```

```
if (exists($plotToRow{$keys[$i]}[$notesNum])) { #Check to see if there the notes field has a value or not
```

```
    $outString = "$outString$plotToRow{$keys[$i]}[$notesNum]\n";
```

```
  }else{
```

```
    $outString = "$outString\n";
```

```
  }
```

```
if (scalar @nums > 2) {
```

```
    $outString = $outString."$nums[2]\tSource: $plotToRow{$keys[$i]}[$sourceNum]\n";
```

```
  }else{
```

```
    $outString = $outString." \tSource: $plotToRow{$keys[$i]}[$sourceNum]\n";
```

```
  }
```

```
if (scalar @nums >3) {
```

```
    $outString = $outString."$nums[3]\t BUD BRK   DM-FRT   CROP   BERRY   FLVR   R1-R5\n";
```

```
  }else{
```

```
    $outString = $outString." \t BUD BRK   DM-FRT   CROP   BERRY   FLVR   R1-R5\n";
```

```
  }
```

```
if (scalar @nums >4) {
```

```
    $outString = $outString."$nums[4]\tWI TRUNK   LVS   VIGOR   SIZE   TEXT\n";
```

```
  }else{
```

```
    $outString = $outString." \tWI TRUNK   LVS   VIGOR   SIZE   TEXT\n";
```

```
  }
```

```
if (scalar(@nums > 5)) {
```

```
    $outString = $outString."$nums[5]\t WI BUD   PM-FRT   CLU-SZ   COLOR   SHAPE\n";
```

```
  }else{
```

```
    $outString = $outString." \t WI BUD   PM-FRT   CLU-SZ   COLOR   SHAPE\n";
```

```

}

if (scalar(@nums > 6)) {

    $outString = $outString."$nums[6]\t BLOOM    LVS  COMPAC  SEED\n";

}

}

}

}

```

```

if (scalar @nums > 7) {

    $outString = $outString."$nums[8]\t FLOWER  PDERM\n";

}

}

}

}

```

#If this is not the last entry on the page, print the following string

```

if($Counter!= 0){

    $outString = $outString."\\n\\n-----\\n";

```

}else{ #Otherwise just print a bunch of newlines so that the next plot is not partially at the bottom of this page and partialy on the next page

```

    $outString = $outString."\\f";

}

```

```

$Counter++;

```

```

    $currentPlot++;

}

my @name = split(/\./,$spreadsheet); #Split the filename based on the period

my $outName = $name[0]."_2ndTestBook.txt";#append the appropriate end the the filename

open(my $OUTFILE, ">>",$outName" ) or die "Could not open $outName: $!\n"; #Open the file with the new name

print $OUTFILE $outString."\n"; #Write the fieldbook string to the file

close($OUTFILE);#Close the file

sub expandPlot{

    my ($string) = @_ ;

    my @numbers;

    my @parts = split(",",$string);

    foreach my $part (@parts){

        if (index($part,"-")!= -1) {

            my @range = split("-", $part);

            if ((scalar @range) != 2) {

                $part =~ s/-//g;

                push @numbers, $part;

            }else{

                for (my $counter = $range[0]; $counter <= $range[1]; $counter++){

                    push(@numbers,$counter);

                }

            }

        }else{


```

```

        push @numbers,$part;

    }

}

return(@numbers);

}

```

## **Appendix IX: dragonParser2.pl**

```

#!/usr/bin/perl -w
use strict;
use warnings;
use Scalar::Util qw(looks_like_number);
use Getopt::Long;
my $helpFlag;
#1: Dragon Data.txt

#2: Vineyard Report
#3: Output file
if (($helpFlag) || (scalar @ARGV != 2)) {
    print <<USAGE;
Usage:
    perl $0 [Input Data Sheet] [Output File Name]

```

### Description:

Converts Dragon Speech-to-text output to a spreadsheet. Takes an input data sheet, A vineyard report sheet (a comma separated file) with columns "VINE\_ROW", "VINE\_NUMBERS", "VINE\_NAME", and "CROSS". These data columns can be in any order.

The row and number columns designate the position in the vineyard, which is used to match a location from the input file. Name and cross are included in the final report. "VINE\_NUMBERS" can include numbers separated by semi-colons and hyphens. Thus 1;3-5;7 is vines 1,3,4,5,7.

Single numbers also works.

A "dragonParse.conf" file must be in the directory. This contains all of the variables the script will recognize. One variable per line (except the first line of the file). The variables should be written as lowercase only. One word variables can be part of two word variables; thus "bud" and "bud break" and "break bud" and "break" can all be valid variables.

The script will warn if it encounters an unknown variable; this needs to be corrected in the input file. It will also warn if it finds a value it cannot interpret as a number (see source code: line 28 for text that can be recognized as numbers. Actual digits will be processed as the number that they are). Incorrect values can be corrected in the output file.

Please see example data sheet for input, as well as example configuration files and vineyard file. Each row in the data sheet must contain a "vine" and "row" variable to find the location from the input vineyard sheet.

#### USAGE

```
    exit;  
}
```

#Text which can be converted to numbers

```
my %textToNumbers = ("zero"=>0,  
    "one"=>1,  
    "two"=>2,  
    "to"=>2,  
    "tew" =>2,  
    "three"=>3,  
    "four"=>4,  
    "for"=>4,  
    "five"=>5,  
    "six"=>6,  
    "seven"=>7,  
    "eight"=>8,  
    "nine"=>9,  
    "ten"=>10,  
    "eleven"=>11,  
    "twelve"=>12,  
    "thirteen"=>13,  
    "fourteen"=>14,  
    "fifteen"=>15,  
    "sixteen"=>16,  
    "seventeen"=>17,  
    "eighteen"=>18,  
    "nineteen"=>19,  
    "twenty"=>20,
```



```

    "thirty"=>30,
    "forty"=>40,
    "fifty"=>50,
    "sixty"=>60,
    "seventy"=>70,
    "eighty"=>80,
    "ninety"=>90);

my %traitMap; #hash to map expected trait words to their spreadsheet headers

my %flagMap; #hash to map trait words to any flags associated with them. Key is trait word, value is pointer to array of flags.

my $conf_counter = 1; #Count which line of configuration file I am on

open(my $CONFIG_FILE, "dragonParse.conf") or die "Could not open dragonParse.conf $!\n"; #open configuration file
while (!eof($CONFIG_FILE)) { #Read all lines of configuration file

    my $line = readline($CONFIG_FILE);
    if($conf_counter > 22){
        chomp $line;
        my @splitLine = split("=", $line); #split line on equals sign
        my @flags = split("-", $splitLine[1]); #Get all flags for variable

        $traitMap{$splitLine[0]} = $flags[0]; #Map the read-in variable to the name of the vvariable to be displayed in the spreadsheet
        shift @flags; #Pop the fron of the arrayu off. This is the spreadsheet display variable
        $flagMap{$splitLine[0]} = \@flags; #Flag map points read-in variable to list of flags associated with him

    }

    $conf_counter++;
}

close $CONFIG_FILE;

#Above covers all numbers 0-20, and every multiple of ten thereafter until 90. Also "to" and "for"

my %traits; #stores all observed traits

my @vineToTraitToValue; #Links vine to all of its traits to all ov its values

my %location;

my @locationFile;

#open(my $READFILE, $vineyardReportFile) or die "Could not open: $vineyardReportFile: $!\n";

my $locationCount = 0;

my $rowColumn = 0;

```

```

my $plotColumn = 0;
my $nameColumn = 0;
my $crossColumn = 0;
my @file;
#Read in the location stuff
#while (!eof($READFILE)) {
# my $line = readline($READFILE);
# chomp $line;
# my @splitLine = split ("",$line);
# push(@file,\@splitLine);
#
# if ($locationCount == 0) {
#   my $headerCount = 0;
#   foreach my $header (@splitLine){
#     if ($header eq "VINE_ROW") {
#       $rowColumn = $headerCount;
#     }elseif($header eq "VINE_NUMBERS"){
#       $plotColumn = $headerCount;
#     }elseif($header eq "VINE_NAME"){
#       $nameColumn = $headerCount;
#     }elseif($header eq "CROSS"){
#       $crossColumn = $headerCount;
#     }
#     $headerCount++;
#   }
#   $locationCount++;
# }
#close($READFILE);
open(my $infile, $ARGV[0]) or die "Could not open $ARGV[0]: $!\n";
my $lineCounter = 0; #Count which line I am on
while (!eof($infile)) {
  $lineCounter++;
  my $line = readline($infile);
  $line =~ s/,//g; #Remove commas from line

```

```

chomp $line;
$line = lc $line; #Lowercase line
my @splitLine = split(/ /, $line); #split on spaces
my %traitToValue; #Hash to hold this vines trait to value pairs
my $traitEncounteredFlag = 0; #Flag to let me know if I hhave processed a trait for this vine/line. This will only be turned on by traits that do
not have a newline flag

for (my $counter = 0; $counter < scalar @splitLine; $counter++){
    my $ogTRAIT = $splitLine[$counter]; #Hold the trait for this current position
    my $ogCounter = $counter; #Hold the current counter
    my $trait = ""; #Initialize trait variable
    $counter = scalar @splitLine; #Counter is in the last position on the line now
    while(!(exists($traitMap{$trait})) && ($counter >= $ogCounter)) { #As long as the counter is greater than or equal to the ogCounter (Where
we just were) and the trait I have does not exist
        $counter--; #Decriment the counter
        $trait = join (" ", @splitLine[$ogCounter..$counter]); #Calculate the new observed trait. Basically I process the whole line from back to
front
        #in order to get the longest variable possible. If "downy" and "downy mildew" are variables, I want the
longer one, because otherwise I will take downy and leave "mildew". This is bad
    }
    if (exists($traitMap{$trait})) { #If I found a trait...
        my @flag = @{$flagMap{$trait}}; #Get the flags for the trait
        my %flags = map { $_ => 1 } @flag;
        if ((exists($flags{"I"})) && ($traitEncounteredFlag == 1)) {
            #This trait has a newline flag and I have encountered a trait before
            $traitEncounteredFlag = 0; #Reset the flag
            $lineCounter++; #Increment line counter
            splice @splitLine, 0, $counter-1; #Cut off the part of the line that has already been processed
            $counter = 0; #Set counter back to zero
            my %tempHash = %traitToValue; #Create a new tempHash
            push(@vineToTraitToValue, \%tempHash); #Add it to arraylist of pointers to these hashes
            %traitToValue = (); #Reset general array
            next; #Next iteration of for loop
        }elseif(exists($flags{"N"})){
            #I have a NOTES flag

```

```

$counter++;

$traitEncounteredFlag = 1; #I have

my $notesString = ""; #String to hold my notes

my $traitTemp = ""; #Trait I am looking at in this loop

my $stopFlag = 0; #Flag to stop adding to notes

while (($stopFlag == 0) && ($counter < scalar @splitLine)) { #While the stop flag is false, and I am not running over the end of my
line

    $traitTemp = ""; #Reset temporary trait each loop

    my $tempCounter= scalar @splitLine; #Reset tempCounter

    while ((!exists($traitMap{$traitTemp})) && ($tempCounter >= $counter)) { #Similar loop to when I was finding my variable above

        $tempCounter--; #Count backwards and find largest variable possible

        $traitTemp = join (" ", @splitLine[$counter..$tempCounter]);

    }

    if (exists($traitMap{$traitTemp})) { #If I found a trait

        #get the flags

        my @flagTemp = @{$flagMap{$traitTemp}};

        my %flagsTemp = map { $_ => 1 } @flagTemp;

        if (exists($flagsTemp{"n"})) { #If i have a "n" flag, stop reading notes

            $stopFlag = 1; #Set stop flag to true.

            $counter--; #Decrement counter

        }else{

            $notesString .= " $splitLine[$counter]"; #No flag, add word to notes

        }

    }else{ #No trait. Add word to notes

        $notesString.= " $splitLine[$counter]";

    }

    $counter++; #increment counter

}

#Done creating notes string. Add notes to my seen variables, and add notes string to the value list

$traitToValue{$traitMap{$trait}} = $notesString;

$traits{$traitMap{$trait}} = 1;

$counter--; #Decremenet counter so I can see the variable that stopped the notes

next;

```

}elseif(exists(\$flags{"w"})){#"w" flag encountered. Treat the value for this trait as the trait itself. This will be reported in the output spreadsheet

```
    $traitEncounteredFlag = 1;

    $traitToValue{$traitMap{$trait}} = $splitLine[$counter];

    $traits{$traitMap{$trait}} = 1;

    next;
}

}else{ #No flags

my $value = $splitLine[$counter+1];

if (!exists($flags{"l"})) { #If variable does not have a newline flag, set traitEncountered Flag to true

    $traitEncounteredFlag = 1;

}

if (looks_like_number($value) || exists($textToNumbers{$value})) { #Process the value

    if (exists($textToNumbers{$value})) {

        $traitToValue{$traitMap{$trait}} = $textToNumbers{$value};

    }else{

        $traitToValue{$traitMap{$trait}} = $value;

    }

    $traits{$traitMap{$trait}} = 1;

    $counter++;

    next;

}

}else{ #I do not recognize the value

if (!exists($traitMap{$value})) { #Look ahead and see if I am looking at a variable. Note it only looks one ahead. Copuld cause

problems

        $traitToValue{$traitMap{$trait}} = $value;

        $traits{$traitMap{$trait}} = 1;

        $counter++;

        print "Unknown value $value at line $lineCounter: $splitLine[$counter-1] $splitLine[$counter] $splitLine[$counter+1]\n";

    }else{

        print "Potentially missing value for trait $trait. Next observed word is $value: $splitLine[$counter-1] $splitLine[$counter]

$splitLine[$counter+1]\n";

    }

}

}

#print "Added $value to $trait\n";

next;
```

```

    }
}

}else{
    print("Could not identify trait beginning with $ogTRAIT: $splitLine[$counter-1] $splitLine[$counter] $splitLine[$counter+1]\n");

    $counter = $ogCounter;
    next;
}

}

my %tempHash = %traitToValue;
push(@vineToTraitToValue, \%tempHash);

}

close ($infile);

open(my $OUTFILE, ">>", $ARGV[1]) or die "Could not open $ARGV[2] $!:\n";
#print $OUTFILE "VINE_NAME,CROSS,";

foreach my $trait (keys %traits){
    if (exists($traits{$trait})) {
        print $OUTFILE $trait.", ";
    }
}

}

print $OUTFILE "\n";

foreach my $vine (@vineToTraitToValue){
    ##print $OUTFILE getName($vineToTraitToValue{$vine}){"vine"},
    $vineToTraitToValue{$vine}{"row"}.", ".getCross($vineToTraitToValue{$vine}{"vine"}, $vineToTraitToValue{$vine}{"row"}).";
    my %hash = %{$vine};
    foreach my $trait (keys %traits){
        if (exists($hash{$trait})) {
            print $OUTFILE $hash{$trait}.", ";
        }else{
            print $OUTFILE ", ";
        }
    }
}

```

```

    }

}

print $OUTFILE "\n";
}

sub getCross{
    return "DUMMY_CROSS";
    my ($number, $row) = @_;
    foreach my $line(@file){
        if (($line->[$rowColumn] == $row)) {
            my @numbers = expandPlot($line->[$plotColumn]);
            foreach my $plot (@numbers){
                if ($plot == $number) {
                    return $line->[$crossColumn];
                }
            }
        }
    }
}

sub getName{
    return "DUMMY_NAME";
    my ($number, $row) = @_;
    foreach my $line(@file){
        if (($line->[$rowColumn] == $row)) {
            my @numbers = expandPlot($line->[$plotColumn]);
            foreach my $plot (@numbers){
                if ($plot == $number) {
                    return $line->[$nameColumn];
                }
            }
        }
    }
}

```

```

}
}
sub expandPlot{
  my ($string) = @_;
  my @numbers;
  my @parts = split(";", $string);
  foreach my $part (@parts){
    if (index($part, "-") != -1) {
      my @range = split("-", $part);
      for (my $counter = $range[0]; $counter <= $range[1]; $counter++){
        push(@numbers, $counter);
      }
    }else{
      push @numbers, $part;
    }
  }
}
return(@numbers);
}

```



**Appendix X: Second Test Format Report Example**

VINE      VINEYARD 33      ROW 84      YEAR      2016

1      05.0403.13      -  
 2      NOTES:R3-13  
 3      Source: 34-71-007  
 4      BUD BRK      DM-FRT      CROP      BERRY      FLVR      R1-R5  
      WI TRUNK      LVS      VIGOR      SIZE      TEXT  
      WI BUD      PM-FRT      CLU-SZ      COLOR      SHAPE  
      BLOOM      LVS      COMPAC      SEED  
      FLOWER      PDERM

-----  
 13     05.0428.01      -  
 14     NOTES:R4-13  
      Source: 34-75-068  
      BUD BRK      DM-FRT      CROP      BERRY      FLVR      R1-R5  
      WI TRUNK      LVS      VIGOR      SIZE      TEXT  
      WI BUD      PM-FRT      CLU-SZ      COLOR      SHAPE  
      BLOOM      LVS      COMPAC      SEED  
      FLOWER      PDERM

-----  
 17     05.0435.02      -  
 18     NOTES:R3-13  
      Source: 34-79-008  
      BUD BRK      DM-FRT      CROP      BERRY      FLVR      R1-R5  
      WI TRUNK      LVS      VIGOR      SIZE      TEXT  
      WI BUD      PM-FRT      CLU-SZ      COLOR      SHAPE  
      BLOOM      LVS      COMPAC      SEED  
      FLOWER      PDERM

-----  
 19     06.0528.01      -  
 20     NOTES:  
      Source: 34-80-041  
      BUD BRK      DM-FRT      CROP      BERRY      FLVR      R1-R5  
      WI TRUNK      LVS      VIGOR      SIZE      TEXT  
      WI BUD      PM-FRT      CLU-SZ      COLOR      SHAPE  
      BLOOM      LVS      COMPAC      SEED  
      FLOWER      PDERM

VINE VINEYARD 33 ROW 84 YEAR 2016

21 06.0530.02 -

22 NOTES:

Source: 34-81-092

BUD BRK	DM-FRT	CROP	BERRY	FLVR	R1-R5
WI TRUNK	LVS	VIGOR	SIZE	TEXT	
WI BUD	PM-FRT	CLU-SZ	COLOR	SHAPE	
BLOOM	LVS	COMPAC	SEED		
FLOWER	PDERM				

-----  
23 06.0530.03 -

24 NOTES:

Source: 34-82-057

BUD BRK	DM-FRT	CROP	BERRY	FLVR	R1-R5
WI TRUNK	LVS	VIGOR	SIZE	TEXT	
WI BUD	PM-FRT	CLU-SZ	COLOR	SHAPE	
BLOOM	LVS	COMPAC	SEED		
FLOWER	PDERM				

-----  
25 06.0531.02 -

26 NOTES:

Source: 34-83-023

BUD BRK	DM-FRT	CROP	BERRY	FLVR	R1-R5
WI TRUNK	LVS	VIGOR	SIZE	TEXT	
WI BUD	PM-FRT	CLU-SZ	COLOR	SHAPE	
BLOOM	LVS	COMPAC	SEED		
FLOWER	PDERM				

-----  
27 06.0534.01 -

28 NOTES:

Source: 34-84-043

BUD BRK	DM-FRT	CROP	BERRY	FLVR	R1-R5
WI TRUNK	LVS	VIGOR	SIZE	TEXT	
WI BUD	PM-FRT	CLU-SZ	COLOR	SHAPE	
BLOOM	LVS	COMPAC	SEED		
FLOWER	PDERM				

VINE VINEYARD 33 ROW 84 YEAR 2016

31 06.0536.02 -  
32 NOTES:CHECK EARLY  
Source: 34-86-048  
BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
WI TRUNK LVS VIGOR SIZE TEXT  
WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

---

33 06.0537.01 -  
34 NOTES:  
Source: 34-87-034  
BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
WI TRUNK LVS VIGOR SIZE TEXT  
WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

---

35 07.0618.01 -  
36 NOTES:CHECK EARLY  
Source: 34-90-005  
BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
WI TRUNK LVS VIGOR SIZE TEXT  
WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

---

37 07.0621.01 -  
38 NOTES:  
Source: 34-91-027  
BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
WI TRUNK LVS VIGOR SIZE TEXT  
WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

VINE VINEYARD 33 ROW 84 YEAR 2016

39 07.0621.02 -

40 NOTES:

Source: 34-91-049

BUD BRK	DM-FRT	CROP	BERRY	FLVR	R1-R5
WI TRUNK	LVS	VIGOR	SIZE	TEXT	
WI BUD	PM-FRT	CLU-SZ	COLOR	SHAPE	
BLOOM	LVS	COMPAC	SEED		
FLOWER	PDERM				

-----  
41 07.0621.03 -

42 NOTES:

Source: 34-92-003

BUD BRK	DM-FRT	CROP	BERRY	FLVR	R1-R5
WI TRUNK	LVS	VIGOR	SIZE	TEXT	
WI BUD	PM-FRT	CLU-SZ	COLOR	SHAPE	
BLOOM	LVS	COMPAC	SEED		
FLOWER	PDERM				

-----  
43 07.0621.04 -

44 NOTES:

Source: 34-92-014

BUD BRK	DM-FRT	CROP	BERRY	FLVR	R1-R5
WI TRUNK	LVS	VIGOR	SIZE	TEXT	
WI BUD	PM-FRT	CLU-SZ	COLOR	SHAPE	
BLOOM	LVS	COMPAC	SEED		
FLOWER	PDERM				

-----  
45 08.0721.01 -

46 NOTES:

47 Source: 34-94-069

48 BUD BRK	DM-FRT	CROP	BERRY	FLVR	R1-R5
WI TRUNK	LVS	VIGOR	SIZE	TEXT	
WI BUD	PM-FRT	CLU-SZ	COLOR	SHAPE	
BLOOM	LVS	COMPAC	SEED		
FLOWER	PDERM				



VINE VINEYARD 33 ROW 85 YEAR 2016

1 08.0710.01 -  
2 NOTES:  
3 Source: 34-93-069  
4 BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
WI TRUNK LVS VIGOR SIZE TEXT  
WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

---

5 08.0716.01 -  
6 NOTES: ?/-/-/-/- ;  
7 Source: 34-93-085  
8 BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
WI TRUNK LVS VIGOR SIZE TEXT  
WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

---

9 08.0721.02 -  
10 NOTES: ?/-/-/-/- ;  
11 Source: 34-94-081  
12 BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
WI TRUNK LVS VIGOR SIZE TEXT  
WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

---

13 89.0607.04 -  
14 NOTES:  
15 Source: 33-69-022-24; 33-63-033; 34-09-056  
16 BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
17 WI TRUNK LVS VIGOR SIZE TEXT  
18 WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

VINE VINEYARD 33 ROW 85 YEAR 2016

19 98.0228.02 -  
20 NOTES:SEE EARLY  
21 Source: 33-70-044-46; 34-39-024  
BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
WI TRUNK LVS VIGOR SIZE TEXT  
WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

---

22 98.0234.02 -  
23 NOTES:  
24 Source: 33-68-016-18; 34-41-040  
25 BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
26 WI TRUNK LVS VIGOR SIZE TEXT  
27 WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

---

28 99.0410.02 -  
29 NOTES:SEE EARLY  
30 Source: 33-74-007-9; 43-01-073  
31 BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
32 WI TRUNK LVS VIGOR SIZE TEXT  
33 WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

---

34 99.0415.01 -  
35 NOTES:  
36 Source: 33-73-035-39; 43-03-022  
BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
WI TRUNK LVS VIGOR SIZE TEXT  
WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

VINE VINEYARD 33 ROW 85 YEAR 2016

37 01.0621.01 -  
38 NOTES:  
39 Source: 33-78-001-3; 43-14-056  
BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
WI TRUNK LVS VIGOR SIZE TEXT  
WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

---

40 01.0625.01 -  
41 NOTES:EARLY  
42 Source: 33-78-007-9; 43-17-046  
43 BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
44 WI TRUNK LVS VIGOR SIZE TEXT  
45 WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

---

46 03.0224.01 -  
47 NOTES:  
48 Source: 33-78-010-12; 34-53-014  
BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
WI TRUNK LVS VIGOR SIZE TEXT  
WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

---



VINE VINEYARD 33 ROW 86 YEAR 2016

1 05.0425.01 -

2 NOTES:

Source: 34-74-054

BUD BRK	DM-FRT	CROP	BERRY	FLVR	R1-R5
WI TRUNK	LVS	VIGOR	SIZE	TEXT	
WI BUD	PM-FRT	CLU-SZ	COLOR	SHAPE	
BLOOM	LVS	COMPAC	SEED		
FLOWER	PDERM				

-----  
5 06.0537.02 -

6 NOTES:EARLY

Source: 34-87-037

BUD BRK	DM-FRT	CROP	BERRY	FLVR	R1-R5
WI TRUNK	LVS	VIGOR	SIZE	TEXT	
WI BUD	PM-FRT	CLU-SZ	COLOR	SHAPE	
BLOOM	LVS	COMPAC	SEED		
FLOWER	PDERM				

-----  
7 06.0538.01 -

8 NOTES:DOUBLE BERRY

Source: 34-88-056

BUD BRK	DM-FRT	CROP	BERRY	FLVR	R1-R5
WI TRUNK	LVS	VIGOR	SIZE	TEXT	
WI BUD	PM-FRT	CLU-SZ	COLOR	SHAPE	
BLOOM	LVS	COMPAC	SEED		
FLOWER	PDERM				

-----  
9 06.0539.01 -

10 NOTES:

Source: 34-88-018

BUD BRK	DM-FRT	CROP	BERRY	FLVR	R1-R5
WI TRUNK	LVS	VIGOR	SIZE	TEXT	
WI BUD	PM-FRT	CLU-SZ	COLOR	SHAPE	
BLOOM	LVS	COMPAC	SEED		
FLOWER	PDERM				

VINE VINEYARD 33 ROW 86 YEAR 2016

11 07.0621.05 -

12 NOTES:

Source: 34-91-020

BUD BRK	DM-FRT	CROP	BERRY	FLVR	R1-R5
WI TRUNK	LVS	VIGOR	SIZE	TEXT	
WI BUD	PM-FRT	CLU-SZ	COLOR	SHAPE	
BLOOM	LVS	COMPAC	SEED		
FLOWER	PDERM				

-----  
13 07.0621.06 -

14 NOTES:

Source: 34-91-053

BUD BRK	DM-FRT	CROP	BERRY	FLVR	R1-R5
WI TRUNK	LVS	VIGOR	SIZE	TEXT	
WI BUD	PM-FRT	CLU-SZ	COLOR	SHAPE	
BLOOM	LVS	COMPAC	SEED		
FLOWER	PDERM				

-----  
15 07.0621.07 -

16 NOTES:

Source: 34-92-019

BUD BRK	DM-FRT	CROP	BERRY	FLVR	R1-R5
WI TRUNK	LVS	VIGOR	SIZE	TEXT	
WI BUD	PM-FRT	CLU-SZ	COLOR	SHAPE	
BLOOM	LVS	COMPAC	SEED		
FLOWER	PDERM				

-----  
17 08.0701.01 -

18 NOTES:

19 Source: 34-93-011

20 BUD BRK	DM-FRT	CROP	BERRY	FLVR	R1-R5
WI TRUNK	LVS	VIGOR	SIZE	TEXT	
WI BUD	PM-FRT	CLU-SZ	COLOR	SHAPE	
BLOOM	LVS	COMPAC	SEED		
FLOWER	PDERM				

VINE VINEYARD 33 ROW 86 YEAR 2016

21 08.0702.01 -  
22 NOTES:  
23 Source: 34-93-021  
24 BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
WI TRUNK LVS VIGOR SIZE TEXT  
WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

---

25 08.0702.02 -  
26 NOTES:  
27 Source: 34-93-042  
28 BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
WI TRUNK LVS VIGOR SIZE TEXT  
WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

---

29 08.0702.03 -  
30 NOTES:  
31 Source: 34-93-043  
32 BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
WI TRUNK LVS VIGOR SIZE TEXT  
WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

---

33 08.0710.02 -  
34 NOTES:  
35 Source: 34-93-061  
36 BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
WI TRUNK LVS VIGOR SIZE TEXT  
WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

VINE VINEYARD 33 ROW 86 YEAR 2016

37 08.0721.03 -  
38 NOTES:  
39 Source: 34-94-083  
40 BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
WI TRUNK LVS VIGOR SIZE TEXT  
WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

---

41 08.0722.01 -  
42 NOTES:  
43 Source: 34-94-026  
44 BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
WI TRUNK LVS VIGOR SIZE TEXT  
WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

---

45 08.0722.02 -  
46 NOTES:  
47 Source: 34-94-049  
48 BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
WI TRUNK LVS VIGOR SIZE TEXT  
WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

---

VINE VINEYARD 33 ROW 87 YEAR 2016

1 08.0726.01 -  
2 NOTES:  
3 Source: 34-95-014  
4 BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
WI TRUNK LVS VIGOR SIZE TEXT  
WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

---

5 06.0530.04 -  
6 NOTES:  
Source: 34-82-061  
BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
WI TRUNK LVS VIGOR SIZE TEXT  
WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

---

7 07.0618.02 -  
8 NOTES:V.V. EARLY  
Source: 34-90-020  
BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
WI TRUNK LVS VIGOR SIZE TEXT  
WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

---

9 07.0620.02 -  
10 NOTES:V. EARLY  
Source: 34-90-088  
BUD BRK DM-FRT CROP BERRY FLVR R1-R5  
WI TRUNK LVS VIGOR SIZE TEXT  
WI BUD PM-FRT CLU-SZ COLOR SHAPE  
BLOOM LVS COMPAC SEED  
FLOWER PDERM

VINE VINEYARD 33 ROW 87 YEAR 2016

11 08.0732.02 -

12 NOTES:

Source: 34-95-078

BUD BRK	DM-FRT	CROP	BERRY	FLVR	R1-R5
WI TRUNK	LVS	VIGOR	SIZE	TEXT	
WI BUD	PM-FRT	CLU-SZ	COLOR	SHAPE	
BLOOM	LVS	COMPAC	SEED		
FLOWER	PDERM				

---

13 08.0732.03 -

14 NOTES:

Source: 34-96-078

BUD BRK	DM-FRT	CROP	BERRY	FLVR	R1-R5
WI TRUNK	LVS	VIGOR	SIZE	TEXT	
WI BUD	PM-FRT	CLU-SZ	COLOR	SHAPE	
BLOOM	LVS	COMPAC	SEED		
FLOWER	PDERM				

---

**Appendix XI: Seedling Test Format Report Example**

VINE VINEYARD 36 ROW 1 YEAR 2016

7 88.0507.01 66.0795.01/MI# 2

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

12 88.0514. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

13 88.0514. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

14 88.0514. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

16 88.0514. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

18 88.0514. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

19 88.0514. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

22 88.0514. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

23 88.0514.03 -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R



VINE VINEYARD 36 ROW 1 YEAR 2016

24 88.0514. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

25 88.0514. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

26 88.0514.01 -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

28 88.0514. -

NOTES: perfect (somewhat reflex?)

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

29 88.0514. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

30 88.0514. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

31 88.0514. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

32 88.0514. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

34 88.0514. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 1 YEAR 2016

36 88.0514. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

38 88.0514. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

39 88.0514. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

40 88.0514. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

42 88.0514.06 -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

45 88.0514. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

47 88.0514. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

48 88.0514. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

50 88.0514.04 -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 1 YEAR 2016

51 88.0514. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

52 88.0514. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

53 88.0514. 5-1-6 -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

55 88.0514. 5-2-2 -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

57 88.0514. 5-2-1 -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

58 88.0514. 5-2-4 -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

59 88.0514.02 5-1-5 -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

61 88.0514. 5-1-1 -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

62 88.0514. 5-1-2 -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 1 YEAR 2016

63 88.0514. 5-1-7 -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

-----  
65 88.0514. 5-2-7 -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

-----  
66 88.0514. 5-1-3 -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

-----  
67 88.0514. 5-3-5 -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

-----  
69 88.0514. 5-4-1 -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

-----  
72 88.0514. 5-3-8 -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

-----  
73 88.0514. 5-4-8 -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

-----  
75 88.0514. 5-4-6 -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

-----  
76 88.0514. 5-4-4 -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 1 YEAR 2016

77 88.0514. 5-2-8 -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

78 88.0514. 5-3-1 -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

79 88.0514. 5-3-4 -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

80 88.0514. 5-4-5 -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

81 88.0514. 5-3-9 -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

83 88.0514. 5-4-7 -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

84 88.0514. 5-6-8 -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

85 88.0514. 5-5-8 -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

89 88.0514. 5-5-3 -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 1 YEAR 2016

90 88.0514. 5-6-1 -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

92 88.0514. 5-5-2 -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

93 88.0514. 5-6-3 -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

94 88.0514. 5-6-5 -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

95 88.0514. 5-6-6 -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

96 88.0514. 5-5-5 -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

VINE VINEYARD 36 ROW 3 YEAR 2016

2 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

3 96.0801. -

NOTES:

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

5 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

6 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

7 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

8 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

10 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

11 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

12 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 3 YEAR 2016

14 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

15 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

16 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

17 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

19 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

20 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

21 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

22 96.0801. -

NOTES:

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

23 96.0801. -

NOTES: male; blk rot susc.

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R



VINE VINEYARD 36 ROW 3 YEAR 2016

24 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

25 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

27 Chancellor /

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

28 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

30 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

31 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

32 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

34 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

35 96.0801. -

NOTES:

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 3 YEAR 2016

36 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

37 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

40 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

41 96.0801. -

NOTES:

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

43 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

44 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

46 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

47 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

48 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 3 YEAR 2016

49 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

50 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

51 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

52 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

53 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

55 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

57 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

58 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

59 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 3 YEAR 2016

60 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

61 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

62 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

63 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

64 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

65 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

66 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

67 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

68 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 3 YEAR 2016

69 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

70 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

72 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

73 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

74 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

75 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

76 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

77 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

81 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 3 YEAR 2016

82 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

86 96.0801. -

NOTES: male?

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

87 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

88 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

89 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

90 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

91 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

92 96.0801. -

NOTES:

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

95 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 3 YEAR 2016

96 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

97 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

98 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

99 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

100 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

101 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

102 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

104 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

105 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 3 YEAR 2016

106 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

-----  
110 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R  
-----



VINE VINEYARD 36 ROW 4 YEAR 2016

1 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

2 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

3 96.0801. -

NOTES: male?

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

4 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

5 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

8 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

9 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

10 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

12 96.0801. -

NOTES: perfect?

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 4 YEAR 2016

14 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

15 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

16 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

17 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

21 Steuben /

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

22 PI 200569 -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

23 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

26 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

27 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 4 YEAR 2016

29 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

30 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

32 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

33 96.0801. -

NOTES:

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

34 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

35 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

36 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

37 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

38 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 4 YEAR 2016

39 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

40 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

44 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

45 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

46 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

47 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

48 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

49 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

50 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 4 YEAR 2016

51 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

52 96.0801. -

NOTES:

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

53 96.0801. -

NOTES:

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

54 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

55 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

57 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

59 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

61 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

62 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 4 YEAR 2016

64 96.0801. -

NOTES: perfect?

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

65 96.0801. -

NOTES:

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

66 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

69 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

70 96.0801. -

NOTES:

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

71 96.0801. -

NOTES:

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

72 96.0801. -

NOTES:

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

73 Concord /

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

74 Chancellor /

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 4 YEAR 2016

75 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

76 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

78 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

79 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

80 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

82 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

83 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

86 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

VINE VINEYARD 36 ROW 5 YEAR 2016

4 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

5 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

6 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

7 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

10 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

12 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

13 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

15 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

17 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R



VINE VINEYARD 36 ROW 5 YEAR 2016

18 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

19 96.0801. -

NOTES:

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

20 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

21 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

23 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

24 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

25 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

27 96.0801. -

NOTES: perfect; R2-'01

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

28 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 5 YEAR 2016

29 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

30 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

32 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

35 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

36 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

37 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

38 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

39 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

41 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 5 YEAR 2016

42 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

44 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

45 96.0801. -

NOTES:

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

47 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

48 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

49 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

51 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

52 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

54 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 5 YEAR 2016

55 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

57 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

59 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

61 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

62 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

63 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

64 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

65 96.0801. -

NOTES: "perfect; R2-'01, R4-'02"

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

67 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 5 YEAR 2016

70 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

74 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

75 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

76 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

77 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

78 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

79 Concord /

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

81 Chancellor /

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

83 96.0801. -

NOTES: "perfect; R2-'01, R3-'02"

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 5 YEAR 2016

84 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

85 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

86 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

87 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

88 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

90 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

91 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

92 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

93 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 5 YEAR 2016

96 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

97 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

99 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

101 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

103 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

104 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

106 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

107 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

108 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 5 YEAR 2016

109 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

-----



VINE VINEYARD 36 ROW 6 YEAR 2016

2 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

3 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

5 96.0801. -

NOTES: perfect; R2-'01&'02

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

6 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

7 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

8 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

13 96.0801. -

NOTES: perfect; R2-'01&'02

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

15 96.0801. -

NOTES: perfect; R2-'02

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

17 96.0801. -

NOTES: perfect; R2-'01&'02

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 6 YEAR 2016

18 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

20 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

22 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

23 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

25 96.0801. -

NOTES: perfect; R2-'02

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

26 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

27 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

29 96.0801. -

NOTES: perfect; R2-'01&'02

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

30 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 6 YEAR 2016

32 96.0801. -

NOTES: "perfect; R2-'01, R4-'02"

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

34 96.0801. -

NOTES: "perfect; R2-'01, R4-'02"

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

35 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

38 96.0801. -

NOTES: "perfect; R2-'01, R2-'02"

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

39 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

40 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

41 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

42 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

43 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 6 YEAR 2016

45 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

46 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

47 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

48 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

49 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

50 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

53 Chancellor /

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

54 Concord /

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

55 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 6 YEAR 2016

56 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

59 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

60 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

64 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

65 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

66 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

67 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

68 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

69 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 6 YEAR 2016

70 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

72 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

74 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

75 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

76 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

78 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

83 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

85 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

86 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 6 YEAR 2016

87 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

88 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

91 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

93 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

94 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

95 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

97 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

98 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

100 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 6 YEAR 2016

103 96.0801. -

NOTES: perfect; self/outcross?

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

104 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

105 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

106 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

107 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

108 Chancellor /

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

109 Steuben /

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

110 Concord /

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---



VINE VINEYARD 36 ROW 7 YEAR 2016

1 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

3 96.0801. -

NOTES:

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

4 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

5 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

6 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

7 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

9 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

11 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

16 96.0801. -

NOTES:

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 7 YEAR 2016

18 96.0801. -

NOTES: perfect; R2-'01&'02

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

19 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

20 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

21 96.0801. -

NOTES:

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

22 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

23 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

26 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

31 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

34 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 7 YEAR 2016

35 96.0801. -

NOTES:

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

36 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

37 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

38 96.0801. -

NOTES: male or perfect? Check!

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

39 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

40 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

41 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

42 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

---

43 96.0801. -

NOTES: "perfect; R2-'01, R4-'02"

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 7 YEAR 2016

47 96.0801. -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

-----  
48 96.0801. -NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

-----  
49 96.0801. -

NOTES: male

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

-----  
51 96.0801. -

NOTES: perfect; Sel. in no-spray nurs.

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

-----  
52 96.0801. -

NOTES: male; Sel. in no-spray nurs.

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

-----  
66 96.0804.01 -

NOTES: "perfect; R3-'00, R2-'02, R1-'04"

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

-----  
79 96.0805.01 -

NOTES: "perfect; R3-'02, R1-'04"

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

-----  
80 Chancellor -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

-----  
81 Steuben -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

VINE VINEYARD 36 ROW 7 YEAR 2016

82 Concord -

NOTES: perfect

T B FL DM PM CR VG CS CC BS BC SD FL TX S C R

-----