PEDIGREE QUERY, VISUALIZATION, AND GENETIC
CALCULATIONS TOOL

By

MURAT KURTCEPHE

Submitted in partial fulfillment of the requirements

For the degree of Master of Science

Thesis Adviser: Dr. Z. Meral Özsoyoğlu

Department of Electrical Engineering and Computer Science

CASE WESTERN RESERVE UNIVERSITY

August, 2012
CASE WESTERN RESERVE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

We hereby approve the thesis/dissertation of

MURAT KURTEPHE

candidate for the Master of Science degree *

(signed) Dr. Z. Meral Özsoyoğlu (chair of the committee)

Dr. Jing Li

Dr. Mehmet Koyutürk

(date) June 5, 2012

*We also certify that written approval has been obtained for any proprietary material contained therein.
To My Family,
# Table of Contents

List of Tables .................................................................................................................. ix

List of Figures .................................................................................................................. x

Acknowledgments ............................................................................................................ xiii

Abstract ........................................................................................................................... xiv

Chapter 1 - Introduction ................................................................................................. 1

1.1 Motivation .................................................................................................................. 1

1.2 Contributions .......................................................................................................... 3

1.3 Thesis Structure ....................................................................................................... 4

Chapter 2 - Background ............................................................................................... 6

2.1 Pedigree Querying ................................................................................................... 6

2.2 Genetic Calculations ............................................................................................... 7

2.2.1 Genetic Concepts ............................................................................................... 7

2.2.2 Inbreeding Coefficient ....................................................................................... 8

2.2.3 Kinship Coefficient ............................................................................................. 8

2.2.4 Three Individuals - $\Phi_{abc}$ .............................................................................. 9

2.2.5 Four Individuals - $\Phi_{abcd}$ ............................................................................. 10

2.2.6 Two Pairs of Individuals - $\Phi_{ab,cd}$ ................................................................ 11

2.2.7 Identity Coefficients .......................................................................................... 12
2.3 Pedigree labeling .................................................................................................................. 13

2.3.1 Pedigrees as Directed Acyclic Graphs (DAGs)............................................................... 14

2.3.2 NodeCodes Overview ....................................................................................................... 14

2.3.3 Relationship Queries with NodeCodes ................................................................................ 16

2.3.4 Path-Based Genetic Calculations with NodeCodes ......................................................... 18

2.4 Related Work ....................................................................................................................... 19

2.5 Discussion ............................................................................................................................. 21

Chapter 3 - Pedigree Query, Visualization, and Genetic Calculations Tool (PQV Tool) 22

3.1 Overview ................................................................................................................................ 22

3.1.1 Query Interface .................................................................................................................. 22

3.1.2 Genetic Calculation Interface ............................................................................................ 23

3.1.3 Visualization ..................................................................................................................... 24

3.2 Discussion ............................................................................................................................... 25

Chapter 4 - Pedigree Query Interface ....................................................................................... 26

4.1 Overview ................................................................................................................................ 26

4.2 Query Semantics over Pedigree Graph .................................................................................. 26

4.2.1 Query Input and Output ..................................................................................................... 27

4.2.2 Queries with a single query box - Type A Queries .......................................................... 27

4.2.3 Queries with two query boxes - Type B Queries ............................................................. 28

4.2.4 Queries with three or more query boxes - Type C and D Queries ................................. 30
4.3 Visual Query Building Interface................................................................. 36
  4.3.1 Query starting step.................................................................................. 36
  4.3.2 Attribute-based conditions...................................................................... 37
  4.3.3 Axis steps................................................................................................. 41
  4.3.4 Structure-based conditions ..................................................................... 43
  4.3.5 Selecting more than one query box as output........................................ 50
  4.3.6 EBNF notation of the provided pedigree queries..................................... 51
  4.3.7 Operations over query result................................................................. 53
  4.3.8 Unavailable Query Constructs............................................................... 54

4.4 Query Interface Evaluation.......................................................................... 55
  4.4.1 Terminology ............................................................................................ 55
  4.4.2 Selectivity Conditions............................................................................. 56
  4.4.3 Query Template Parsing Algorithm....................................................... 56
  4.4.4 Mapping to Relational Algebra ............................................................... 59

4.5 Discussion..................................................................................................... 63

Chapter 5 - Visualization...................................................................................... 64
  5.1 Pedigree Data Visualization ....................................................................... 64
  5.2 Visualization Interactions ......................................................................... 65
    5.2.1 Query Result Highlighting...................................................................... 65
    5.2.2 Parent and Children Highlighting........................................................ 66
5.2.3 Operations over the Visualization ........................................... 67

5.3 Graph Layout .................................................................................... 68

5.3.1 Pedigree drawing properties ......................................................... 68

5.3.2 Sugiyama Layout ............................................................................ 69

5.4 Discussion ......................................................................................... 69

Chapter 6 - Implementation Details ......................................................... 70

6.1 SQLite ................................................................................................. 70

6.1.1 Database Schema ........................................................................... 70

6.2 Windows Presentation Foundation (WPF) Framework ...................... 71

6.2.1 Extensible Application Markup Language (XAML) ...................... 72

6.3 Model - View - ViewModel (MVVM) Pattern ...................................... 72

6.3.1 Data Binding ................................................................................... 73

6.4 Pedigree Query Language Library (PQLLib) ..................................... 74

6.5 Third Party Libraries ......................................................................... 74

6.5.1 Graph Sharp (Graph#) ................................................................. 75

6.6 Discussion ......................................................................................... 75

Chapter 7 - Future Work and Conclusion .............................................. 76

7.1 Future Work ....................................................................................... 76

7.1.1 PQV Tool improvements ............................................................... 76

7.1.2 Query Interface .............................................................................. 76
7.1.3 Genetic calculations ............................................................ 76

7.1.4 Visualization .......................................................................... 77

7.2 Conclusion .................................................................................. 77

Appendix A - User Manual ................................................................ 79

A.1 Source Code and Compilation ..................................................... 79

A.2 Download and Installation .......................................................... 79

A.3 Overview ................................................................................... 80

A.4 Query Interface .......................................................................... 81

A.4.1 Query Box .............................................................................. 81

A.4.2 Adding Conditions ................................................................. 81

A.4.3 Query Execution ..................................................................... 83

A.4.4 Query Save and Load ............................................................... 83

A.4.5 Query Results ......................................................................... 84

A.5 Visualization .............................................................................. 85

A.6 Genetic Calculations ................................................................. 88

A.7 Settings .................................................................................... 89

Appendix B - Namespaces and Class Descriptions for PQV Tool ............. 90

Bibliography .................................................................................... 96
List of Tables

TABLE 2.1 Example pedigree queries. ................................................................. 6

TABLE 2.2 Number of individuals needed for different relatedness calculations
................................................................................................................................. 8
List of Figures

FIGURE 1.1 SAMPLE PEDIGREE TREE DIAGRAM, ORIGINAL FIGURE FROM [2]............................ 1

FIGURE 2.1 THE GROUPING OF 15 IDENTITY STATES INTO 9 CONDENSED IDENTITY STATES, ORIGINAL FIGURE FROM [9] ............................................................................................................. 12

FIGURE 2.2 THE CONDENSED IDENTITY COEFFICIENTS CALCULATION FORMULAS............. 13

FIGURE 2.3 PEDIGREE GRAPH OF THE SAMPLE PEDIGREE GIVEN IN FIGURE 1.1 WITH NODE CODES LABELING, ORIGINAL FIGURE FROM [2]................................................................. 15

FIGURE 3.1 PQV TOOL: PEDIGREE QUERY OPERATIONS OVERVIEW................................. 23

FIGURE 3.2 PQV TOOL: GENETIC CALCULATIONS OVERVIEW.......................................... 24

FIGURE 4.1 (A) TYPE A QUERY WITHOUT ATTRIBUTE-BASED CONDITIONS. (B) TYPE A QUERY WITH ATTRIBUTE-BASED CONDITIONS .............................................................................. 28

FIGURE 4.2 TYPE B QUERY SEMANTICS GRAPHICAL REPRESENTATION ............................. 30

FIGURE 4.3 THE GRAPHICAL REPRESENTATION OF TYPE C QUERY SEMANTICS ................ 31

FIGURE 4.4 THE GENERALIZATION OF THE TYPE C QUERIES ........................................... 32

FIGURE 4.5 TYPE D QUERY SEMANTICS GRAPHICAL REPRESENTATION ............................. 33

FIGURE 4.6 THE EQUIVALENCE OF TYPE C AND TYPE D QUERIES WITH THREE QUERY BOXES .......................................................................................................................... 34

FIGURE 4.7 THE GENERAL FORM OF TYPE D QUERIES ...................................................... 35

FIGURE 4.8 A QUERY REPRESENTATION WHERE TYPE C AND TYPE D SEMANTICS ARE COMBINED .......................................................................................................................... 35

FIGURE 4.9 THE QUERY STARTING STEP, A SINGLE QUERY BOX USED (TYPE A) ............ 37

FIGURE 4.10 AN EXAMPLE QUERY WITH ATTRIBUTE-BASED CONDITIONS, A SINGLE QUERY BOX USED (TYPE A) ........................................................................................................ 39
FIGURE 6.2 THE XAML EXAMPLE FROM PQV TOOL IMPLEMENTATION ................................ 72
FIGURE 6.3 INTERACTIONS BETWEEN THE ELEMENTS IN MVVM PATTERN ......................... 73
FIGURE A.1 GENERAL VIEW OF THE PQV TOOL ........................................................................ 80
FIGURE A.2 THE QUERY BOX IN PQV TOOL ............................................................................... 81
FIGURE A.3 THE SCREENS PROVIDED BY CONDITION BUTTONS ............................................... 82
FIGURE A.4 THE ATTRIBUTE-BASED CONDITIONS IN PQV TOOL ........................................... 82
FIGURE A.5 QUERY EXECUTION BUTTONS IN PQV TOOL .......................................................... 83
FIGURE A.6 QUERY SAVE AND SAVED QUERIES MANAGEMENT WINDOW .............................. 84
FIGURE A.7 THE QUERY RESULT INTERFACES PROVIDE BY THE PQV TOOL ........................... 85
FIGURE A.8 THE ZOOM IN & OUT AND HIGHLIGHTING IN VISUALIZATION INTERFACE ....... 86
FIGURE A.9 THE INTERACTION BETWEEN DIFFERENT DATA VIEWS .................................... 87
FIGURE A.10 THE BASIC QUERY AND GENETIC CALCULATIONS VIA VISUALIZATION .......... 88
FIGURE A.11 PERFORMING GENETIC CALCULATIONS BY USING DIFFERENT DATA VIEWS ... 88
FIGURE A.12 SAVING GENETIC CALCULATION RESULTS TO A CSV FILE ............................ 89
Acknowledgments

This thesis would not have been possible without the great support I have received from some special people. First of all, I am heartily thankful to my supervisor Professor Meral Özsoyoğlu. Her guidance kept me on track, allowed me to conduct my research, and produce this work. It was a great pleasure working with her in this thesis.

I would like to thank to Professor Jin Li and Professor Mehmet Koyutürk for accepting to read and review this thesis. Their comments and insights are very valuable.

I would like to thank members of Database Lab. for their valuable friendship. I would specially like to thank to my collage Dr. En Cheng for her valuable comments and collaboration during my research. In addition, I would like to thank Dr. James Church and the Cleveland Clinic’s Familial Polyposis Registry for kindly providing us their real pedigree data.

Last but not least, I would like to thank my parents, Pervin and Yahya for their love and support that always kept me motivated. Without them I would not be able to come so far. Finally, I would like to thank my beloved fiancée Betül for her undying support and being in my life. She is my inspiration and nothing would be the same without her.
Pedigree Query, Visualization, and Genetic Calculations Tool

Abstract

By

Murat Kurtcephe

Family trees, a.k.a. pedigrees, are becoming increasingly important in human genetics, as pedigrees can be utilized to trace a genetic disorder or trait and to calculate disease risks. In this study, we present a new system for pedigree query, visualization, and genetic calculations. A novel query interface is proposed where users can form complicated queries via an easy-to-use graphical user interface with no need for any knowledge of high level query language such as SQL or XPath. A graph encoding method called NodeCodes enables our system to efficiently evaluate relationship-based queries without traversing the graph or using recursive query calls. The visualization of the pedigree data as a dynamic drawing enables the analysis of query results in a more understandable form. The system also provides genetic calculations including inbreeding, kinship, and identity coefficients. Proposed system performs these calculations by using path-based formulas coupling with NodeCodes to achieve efficiency and scalability.
Chapter 1 - Introduction

The processing of the ‘family tree’ data, which is also known as pedigree, is the main interest of this thesis. The specific definition of pedigree is as follows: “A simplified diagram of a family’s genealogy that shows family members’ relationships to each other and how a specific trait, abnormality, or disease has been inherited” [1]. A sample pedigree diagram is shown in Figure 1.1. In this figure, male individuals are represented by square symbol whereas circle symbol represents females.

Figure 1.1 Sample pedigree tree diagram, original figure from [2]

The aspects of pedigree processing that have been investigated in this thesis are pedigree querying, calculation of relatedness and the pedigree data visualization. The motivation behind this effort is going to be discussed in the next section.

1.1 Motivation

Pedigree data has a wide application domain. In human genetics, pedigrees can be utilized to trace a genetic disorder or trait, to calculate disease risks, and to facilitate
genetic counseling. In addition to human genetics, pedigrees are also employed in animal breeding or plant breeding.

The availability of the pedigree data is also important. There are some projects which already have large amount of stored pedigree data. Utah population database currently has 1.6 million family history records that are linked to a large medical record set [3]. The Jagelman Registries of Cleveland Clinic [4] is another example of pedigree databases that has been used actively in medical and genetic researches.

In addition to the local pedigree databases, web-based pedigree storage tools have been available, as well. As the technology behind the data storage improves with lower cost, these systems started to associate more data with their current basic family history data to create richer datasets. One of these web-sites with 1.8 million subscribers started gathering DNA data from their users [5]. As another example, new version of the family health portrait tool by surgeon general allows users to associate medical data with their family records including their relatives [6].

As the pedigree data becomes more available and rich, the pedigree queries become more complex, as well. As shown in [7], the recent pedigree queries are more complicated and simple relationship queries do not satisfying the current needs. As a result of this, query interfaces that can answer complicated structure-based pedigree queries are necessary.

One of the important applications of pedigree data is calculating the cancer risk of an individual in hereditary cancers [8]. This type of risk estimations requires the calculation of relatedness between individuals. Therefore, pedigree processing tools
should provide efficient and easy-to-use ways to obtain measure of relationship between individuals.

All in all, efficient and easy to use pedigree processing tools are necessary. The motivations provided in this section are addressed by the next section where the contributions of this work are discussed.

1.2 Contributions

The proposed pedigree processing tool in this thesis combines querying, visualization and genetic calculations in an intuitive graphical user interface. The main contributions of this thesis are as follows:

- A dynamic and visual pedigree querying interface where both pedigree structure and the data associated with pedigree can be used to form complex queries.
- The query interface produces unambiguous queries that can be mapped to a formal pedigree query language. However, the underlying schema information or the knowledge of high level query language is not necessary to form queries.
- An algorithm that processes the query interface to produce actual SQL queries is proposed. During the evaluation of the pedigree queries, graph encoding method, named NodeCodes [7], is employed to expedite query execution.
- Several genetic calculations are provided via the user interface. Path-based formulas that outperform the traditional methods are used to conduct these calculations.
- An interactive data visualization interface provided to display query result in a more understandable way.
1.3 Thesis Structure

This thesis is organized in eight chapters and appendix. In this chapter, introductory information related to pedigrees and pedigree applications is given. The motivation behind this thesis is explained. The main contributions of this thesis to the pedigree processing domain are discussed.

In Chapter 2, the background information needed for the pedigree data processing will be covered. First, the pedigree queries will be discussed in detail with example pedigree queries. Then, genetic calculations supported by the proposed system will be covered with their definitions and formulas. The graph encoding method named NodeCodes, which is used in this thesis over pedigree data, will be discussed in detail. Finally, the related work section in the Chapter 2 will go over the existing pedigree processing tools and their capabilities.

Chapter 3 is the introduction to the Pedigree Query, Visualization and Genetic Calculations Tool (PQV Tool). This chapter will cover the general capabilities of the tool as an overview. First, the overview of the query interface, which is provided by PQV Tool, is provided. Then, the capabilities of the genetic calculations interface are reviewed. Chapter concludes with the overview of the functionalities provided by the visualization interface.

In Chapter 4, the query aspect of the proposed tool will be covered. This chapter will begin with the overview of the proposed query interface. Then, the semantics of the pedigree graph querying is discussed. Afterwards, the query building constructs of the proposed query interface will be introduced. In addition, the type of queries that can be
answered by the query interface will be covered with example queries. Then, chapter will conclude with the formal query evaluation algorithm.

Chapter 5 is based on the visualization capabilities of the PQV Tool. The graphical representation of the pedigrees, highlighting relationships and layout used in the visualization interface will be discussed in this section.

Chapter 6 is the discussion of the technologies behind this thesis. The database schema, frameworks and the third party libraries used in the prototype of PQV Tool will be presented in this section. Chapter 7 will conclude the thesis with proposed future work and extensions. Appendix A will cover the user manual of PQV Tool and Appendix B will cover namespaces and class descriptions of PQV Tool implementation.
Chapter 2 - Background

In the previous chapter, the introductory information about the pedigrees, their applications and the importance of the pedigree data have been discussed. In this chapter, the types of pedigree queries and genetic calculations addressed in this thesis are explained. The pedigree labeling method, named NodeCodes, is covered with applications to pedigree querying and relatedness calculations. Finally, existing work on pedigree data processing systems is investigated in detail.

2.1 Pedigree Querying

The pedigree queries, which are the main focus of this work, can be based on the structure of pedigree data or the data associated with pedigree or both. Some example pedigree queries are provided in Table 2.1. The queries, Q1 and Q2, defined in this table can be answered on relational databases, as they are only based on the data associated with the pedigree. However, a query such as Q3 needs to traverse the pedigree. And there can be more complex queries such as Q4-Q6 where both traversal of pedigree and the data associated with pedigree are involved.

<table>
<thead>
<tr>
<th>Query #</th>
<th>Query Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>Find all male individuals who are older than 70 in a pedigree.</td>
</tr>
<tr>
<td>Q2</td>
<td>Find the average age of cancer individuals in a given pedigree.</td>
</tr>
<tr>
<td>Q3</td>
<td>Find all ancestors of an individual.</td>
</tr>
<tr>
<td>Q4</td>
<td>Find all individuals who have at least one ancestor with cancer</td>
</tr>
<tr>
<td>Q5</td>
<td>Find the daughters of female individuals who have at least one ancestor with cancer on the maternal side.</td>
</tr>
<tr>
<td>Q6</td>
<td>Find all individuals who have at least one First Degree Relative (FDR) and one Second Degree Relative (SDR) with cancer.</td>
</tr>
</tbody>
</table>

Table 2.1 Example pedigree queries.
2.2 Genetic Calculations

In this thesis, we provide three different relatedness calculations: inbreeding, kinship and condensed identity coefficients. In the next section, the necessary background information related to genetics will be reviewed. Then, the definitions and formulas of these calculations will be provided in each following section.

2.2.1 Genetic Concepts

Some of the genetic concepts have to be covered before introducing the calculations provided in this work. The genetic data is stored in chromosomes. The structurally similar chromosomes are defined as autosomes. Chromosomes contain genes, which are the codes for protein production. The position of a gene on a chromosome is defined as locus. The different forms of a gene are called alleles. Most of the inherited traits or diseases are related to the combination of two alleles at a specific locus [9].

The relatedness calculations used in this work are based on the concept identical by decent (IBD) [10]. The definition of IBD is as follows: two alleles are IBD if one is a physical copy of the other or if they are both physical copies of the same ancestral alleles [2].

The genetic calculations described in this section are available over specific number of individuals and each calculation is denoted by a symbol. Therefore, the number of individuals needed for each genetic calculation is provided with their designated symbols in Table 2.2. The following sections will review each of these genetic calculations.
### Coefficient

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Applies to - denoted by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inbreeding</td>
<td>Single Individual - $F$</td>
</tr>
<tr>
<td>Kinship</td>
<td>Two - $\Phi_{ab}$, Three - $\Phi_{abc}$, Four Individuals - $\Phi_{abcd}$ or Two Pairs of Individuals - $\Phi_{ab,cd}$</td>
</tr>
<tr>
<td>Identity</td>
<td>Two Individuals - $\Delta_1, \Delta_2 \ldots \Delta_9$</td>
</tr>
</tbody>
</table>

Table 2.2 Number of individuals needed for different relatedness calculations

#### 2.2.2 Inbreeding Coefficient

The first relatedness calculation provided by the PQV Tool is inbreeding coefficient which was defined by S. Wright in 1922 [11]. The coefficient is denoted by the symbol $F$. The definition of inbreeding coefficient is as follows: $F$ is the probability that two alleles at one autosomal locus are IBD [9].

The path-counting based formula for inbreeding coefficient is as follows:

$$F_n = \begin{cases} 0 & \text{otherwise} \\ \sum_A \sum_P \left( \frac{1}{2} \right)^{r+s+1} \left( 1 + F_A \right) & \text{if } n \text{ has inbreeding} \end{cases}$$

where $F_n$ is the inbreeding coefficient for individual $n$. $A$ is the common ancestor of the parents of $n$. $F_A$ is the inbreeding coefficient for $A$. $P$ is a pair of paths that is not overlapping from the common ancestor $A$ to the parents of $n$. The number of generations between $n$ and $A$ on the maternal side is denoted by $r$, and on the paternal side is denoted by $s$.

#### 2.2.3 Kinship Coefficient

The next genetic calculation provided in this thesis is the kinship coefficient. The kinship coefficient for two individuals $a$ and $b$, denoted as $\Phi_{ab}$, is the probability that an
allele selected randomly from \(a\) and an allele selected randomly from the same autosomal locus of \(b\) are IBD [2].

The path-counting formula for kinship coefficient is very similar to inbreeding coefficient formula. More specifically, inbreeding coefficient of an individual is actually the kinship coefficient for the parents of the individual [2]. The path-counting approach for \(\Phi_{ab}\) that is based on Wright’s formula [11] is given as follows:

\[
\Phi_{ab} = \sum_A \sum_P \left( \frac{\sqrt{2}}{2} \right)^{r+s+1} (1 + F_A)
\]

where \(A\) is a common ancestor of \(a\) and \(b\). \(P\) is the set of non-overlapping path-pairs from \(A\) to \(a\) and from \(A\) to \(b\). The length of a path from \(A\) to \(a\) is denoted by \(r\) whereas the length of a path from \(A\) to \(b\) is denoted by \(s\). \(F_A\) is the inbreeding coefficient of \(A\).

In addition to \(\Phi_{ab}\), generalized kinship coefficients such as kinship coefficient for three individuals, four individuals and two pairs are also provided via the tool proposed in this thesis. In the following sections, the definitions and the formulas behind these calculations are reviewed. In order to be concise, the details related to the formulas are not included. However, the further explanations can be found in the corresponding references.

**2.2.4 Three Individuals - \(\Phi_{abc}\)**

The generalized kinship coefficient for three individuals \(a\), \(b\), and \(c\) is denoted as \(\Phi_{abc}\). This coefficient is the probability that three randomly chosen alleles are IBD, where one allele is selected from each individual.
The path-counting formula for $\Phi_{abc}$ is proposed by Cheng et al. in [2]. The formula is defined as follows:

$$\Phi_{abc} = \sum_A \left( \sum_{\text{Type 1}} \left( \frac{1}{2} \right)^{l_{\text{triple}}} \Phi_{AAA} + \sum_{\text{Type 2}} \left( \frac{1}{2} \right)^{l_{\text{triple}}+1} \Phi_{AA} \right)$$

where $A$ is a triple-common ancestor of $a$, $b$, and $c$. $s$ is the last individual of the root 2-overlap path $P_{As}$. Type 1: $<p_{As}, p_{Ab}, p_{Ac}>$ has no root 2-overlap. Type 2: $<p_{As}, p_{Ab}, p_{Ac}>$ has one root 2-overlap. In addition, $L_{\text{triple}}$ is defined as follows:

$$L_{\text{triple}} = \begin{cases} L_{p_{As}} + L_{p_{Ab}} + L_{p_{Ac}} & \text{for Type 1} \\ L_{p_{As}} + L_{p_{Ab}} + L_{p_{Ac}} - L_{p_{Ad}} & \text{for Type 2} \end{cases}$$

2.2.5 Four Individuals - $\Phi_{abcd}$

The generalized kinship coefficient for four individuals $a$, $b$, $c$, and $d$ is denoted as $\Phi_{abcd}$. This coefficient is the probability that four randomly chosen alleles are IBD, where one allele is selected from each individual.

The path-counting formula for $\Phi_{abcd}$ is proposed by Cheng in [9]. The path-counting formula is given as follows:

$$\Phi_{abcd} = \sum_A \left( \sum_{\text{Type 1}} \left( \frac{1}{2} \right)^{l_{\text{quad}}} \Phi_{AAAA} + \sum_{\text{Type 2}} \left( \frac{1}{2} \right)^{l_{\text{quad}}+1} \Phi_{AAA} + \sum_{\text{Type 3}} \left( \frac{1}{2} \right)^{l_{\text{quad}}+2} \Phi_{AA} \right)$$

where $A$ is a quad-common ancestor of $a$, $b$, $c$ and $d$. $s$ is the last individual of a root 2-overlap path $P_{As}$. $t$ is the last individual of a root 3-overlap path $P_{At}$. Type 1: $<p_{As}, p_{Ab}, p_{Ac}, p_{Ad}>$ has neither root 2-overlap nor root 3-overlap path. Type 2: $<p_{As}, p_{Ab}, p_{Ac}, p_{Ad}>$ has only one root 2-overlap path. Type 3 is defined as follows:
In addition, $L_{\text{quad}}$ is calculated based on the following formula:

$$L_{\text{quad}} = \begin{cases} 
L_{Pa_a} + L_{Pa_b} + L_{Pa_c} + L_{Pa_d} & \text{for Type1} \\
L_{Pa_a} + L_{Pa_b} + L_{Pa_c} + L_{Pa_d} - L_{Pa_c} & \text{for Type2} \\
L_{Pa_a} + L_{Pa_b} + L_{Pa_c} + L_{Pa_d} - L_{Pa_c} - L_{Pa_c} & \text{for case1 in Type3} \\
L_{Pa_a} + L_{Pa_b} + L_{Pa_c} + L_{Pa_d} - 2L_{Pa_c} & \text{for case2 in Type3} \\
L_{Pa_a} + L_{Pa_b} + L_{Pa_c} + L_{Pa_d} - L_{Pa_c} - L_{Pa_c} & \text{for case3 in Type3}
\end{cases}$$

### 2.2.6 Two Pairs of Individuals - $\Phi_{ab,cd}$

The last generalized kinship coefficient provided by PQV Tool is the kinship coefficient for two pairs of individuals, which is denoted by $\Phi_{ab,cd}$. This coefficient is the probability that a random allele from $a$ is IBD with a random allele from $b$ and a random allele from $c$ is IBD with a random allele from $d$.

Cheng proposed the path-counting formula for $\Phi_{ab,cd}$ in [9]. The path-counting formula is as follows:

$$\Phi_{ab,cd} = \sum_A \left( \sum_{\text{Type1}} \left( \frac{1}{2} \right)^{L_{2-\text{pair}}} \Phi_{AAA} + \sum_{\text{Type2}} \left( \frac{1}{2} \right)^{L_{2-\text{pair}}+1} \Phi_{AAA} + \sum_{\text{Type3}} \left( \frac{1}{2} \right)^{L_{2-\text{pair}}+2} \Phi_{AA} + \sum_{\text{Type4}} \left( \frac{1}{2} \right)^{L_{2-\text{pair}}+1} \Phi_{AA} \right)$$

$$+ \sum_{(S, T) \in \text{Type5}} \left( \frac{1}{2} \right)^{L_{2-\text{pair}}} \Phi_{BB}$$

where $A$ is a quad-common ancestor of $a$, $b$, $c$, and $d$. $S$ is a common ancestor of $a$ and $b$. $T$ is a common ancestor of $c$ and $d$. $h$ is the last individual of a root homo-overlap path $P_{Ah}$. $r$ is the last individual of a root heter-overlap path $P_{Ar}$. $B$ is defined based on the categorization defined as follows:
Identity Coefficients

The final genetic calculation that will be discussed in this section is named as identity coefficients [12]. This relatedness measure provides more detailed information by considering two alleles at a fixed autosomal locus for two individuals. This approach results in 15 possible identity states based on the IBD relation within (due to usage of two alleles) or between individuals. The probabilities associated with each identity states are called identity coefficients.

The identity coefficients provided in this work are not all the 15 different coefficients. Instead, condensed identity coefficients are provided by PQV Tool. The identity states obtained without the distinction of maternally or paternally derived alleles are named as condensed identity states. The grouping of 15 identity states in 9 condensed identity states is given in Figure 2.1. The probabilities associated with these condensed states are denoted with $\Delta$ symbol and named as condensed identity coefficients.

$$B = \begin{cases} 
S & \text{when } \text{gen}(S) < \text{gen}(T) \\
S & \text{when } \text{gen}(S) = \text{gen}(T), \text{ and } T \text{ has two parents} \\
T & \text{otherwise} 
\end{cases}$$

**Figure 2.1** The grouping of 15 identity states into 9 condensed identity states, original figure from [9]
The condensed identity coefficients of given two individuals, \( a \) and \( b \), are calculated based on the formulas provided in Figure 2.2. In this figure, symbol \( f \) and \( m \) denote the mother and father of the individual \( a \), respectively. In addition, symbol \( \overline{f} \) and \( \overline{m} \) denote the mother and father of the individual \( b \), respectively. As seen in this figure, each identity coefficient is calculated based on generalized kinship coefficients.

\[
\begin{align*}
\delta_1 &= \Phi_{fm\overline{m}} \\
\delta_2 &= \Phi_{fm\overline{m}} - \Phi_{fm\overline{m}} \\
\delta_3 &= \Phi_{\overline{m}m} - \Phi_{fm\overline{m}} \\
\delta_4 &= \Phi_{\overline{m}m} - \Phi_{fm\overline{m}} \\
\delta_5 &= \delta_1 + \delta_2 + \delta_3 + \delta_4 \\
\delta_6 &= \delta_1 + \delta_2 + \delta_3 + \delta_4 \\
\delta_7 &= \Phi_{f,m\overline{m}} - \Phi_{fm\overline{m}} \\
\delta_8 &= \Phi_{f,m\overline{m}} - \Phi_{fm\overline{m}} \\
\delta_9 &= \delta_7 + \delta_8 + \delta_9 \\
\delta_{10} &= \delta_7 + \delta_8 + \delta_9 \\
\delta_{11} &= \Phi_{fm\overline{m}} - \Phi_{fm\overline{m}} \\
\delta_{12} &= \Phi_{fm\overline{m}} - \Phi_{fm\overline{m}} \\
\delta_{13} &= \delta_{11} + \delta_{12} \\
\delta_{14} &= \delta_{11} + \delta_{12} \\
\delta_{15} &= 1 - \sum_{i=1}^{14} \delta_i
\end{align*}
\]

*Figure 2.2 The condensed identity coefficients calculation formulas*

### 2.3 Pedigree labeling

This section covers the graph labeling method, named NodeCodes [7, 13], and the applications of NodeCodes in pedigree processing. Since NodeCodes is a graph labeling method, the graph representation of pedigree data is discussed. Then, the code generation and properties of the NodeCodes are investigated. Finally, the NodeCodes usage during the evaluation of relationship queries and path-based genetic calculations are covered.
2.3.1 Pedigrees as Directed Acyclic Graphs (DAGs)

The pedigree data is modeled as a directed acyclic graph (DAG) composed of nodes and edges in [7]. An individual in a pedigree data is represented by a node and parent-child relationship between nodes is represented by an edge. These edges are directed from parents to children.

There exist some important properties of pedigree graphs. In a pedigree graph each node has at most two incoming edges as an individual can have two biological parents. The nodes without incoming edges are called progenitors. And in a more general way, pedigree graphs are DAGs where each node can have at most two edges.

In this thesis, the pedigree data is represented with a single table in the relational database:

**Individual**(PedigreeID, ID, MotherID, FatherID, Gender, …)

As seen in this table, the Pedigree ID (Family ID), ID (Individual ID), parent IDs and Gender are the mandatory attributes. The ellipsis (…) represents other additional attributes associated with the individual.

2.3.2 NodeCodes Overview

In this work, NodeCodes are employed both in query evaluation and path-based genetic calculations. The operations that require graph traversal can be accomplished by using this labeling system in an efficient way. In this section, we will cover how the NodeCodes are generated and their properties.
NodeCodes are assigned in a depth-first-traversal order starting from the progenitors. Each NodeCode includes characters to represent the sibling order and delimiters to represent generations. These delimiters are also used to identify genders where ‘.’, ‘,’ and ‘;’ represents female, male and unknown, respectively. The NodeCode labeled graph of sample pedigree drawing (from Figure 1.1) is shown in Figure 2.3.

The formal definition of NodeCode assignment requires a virtual node, named as $s$, where all progenitors are children of $s$. Then, starting from $s$ for each node $u$ in graph NodeCodes of $u$, $NC(u)$, are assigned as follows [7]:

- If $u$ is the source node, then $NC(u) = \{\text{the empty string}\}$.
- Let $u$ be a node with NodeCode $x$, and $v_0, v_1, \ldots v_k$ be $u$’s children in sibling order, then $NC(v_i) = x_i^*$, where $0 \leq i \leq k$ and * indicates the gender of $v_i$. 

Figure 2.3 Pedigree graph of the sample pedigree given in Figure 1.1 with NodeCodes labeling, original figure from [2]
The NodeCodes are represented with a single table in the relational database:

**NodeCodes**(*PedigreeID, NodeCode, IndividualID*).

The properties of the NodeCodes are covered in depth by [7, 13, 2, 14]. However, we will summarize the properties as they justify the applications of NodeCodes.

**Property 1**: A NodeCode can be assigned to one and only one node in a pedigree graph.

**Property 2**: In single source DAG, there is one-two-one correspondence between NC of u and the paths from s to u. This simply means that number of paths from s to u equals |NC(u)|.

**Property 3**: Let p be a path of length k. Then, there exists a string  
\[str=a_1*a_2*...a_k*\] (where \(a_i\), \(0 \leq i \leq k\), is an integer, and * is a delimiter) such that for every begin, end NodeCode pair \(<b, e>\) representing p, \(e = b + str\) always holds, where + is the string concatenation [7].

These properties are specifically important since the relationship queries and path calculations are based on these properties. In the next section, the application of NodeCodes in pedigree querying will be covered.

### 2.3.3 Relationship Queries with NodeCodes

NodeCodes enables relationship queries such as finding ancestors, descendants or children via simple string comparison in SQL. Let \(uc_1\) be a single NodeCode in \(NC(u)\). Based on the properties summarized in the previous section, node c is a child of node u if and only if \(NC(c)\) includes \(uc_1+str\) where str is a string with two characters (a gender
delimiter and sibling number) [7]. This definition assumes that sibling order can be represented by single character. In this thesis, 64 different characters are used to represent the sibling order. This is a realistic number for representing children and progenitors in a real pedigree data.

The following example query returns the children of individual with ID \( k \):

\[
Q_A: \text{SELECT } n2.\text{IndividualID} \text{ FROM NodeCodes } n1, \text{ NodeCodes } n2 \text{ WHERE } n1.\text{IndividualID}=k \text{ AND } n1.\text{PedigreeID}=n2.\text{PedigreeID} \text{ AND } n1.\text{NodeCode}+\_\_ \text{ LIKE } n2.\text{NodeCode}
\]

In SQL representation, LIKE expressions uses ‘\_’ to express any single character and ‘\%’ represents zero or more characters. In query \( Q_A \), the parents of the given individual can be obtained by replacing the LIKE expression with the following: AND \( n1.\text{NodeCode} \text{ LIKE } n2.\text{NodeCode} +\_\_ \).

The ancestors can be obtained by the modification of LIKE expression with the following: AND \( n1.\text{NodeCode} +\_\% \text{ LIKE } n2.\text{NodeCode} \). Similarly, descendants can be obtained by a simple modification as follows: AND \( n1.\text{NodeCode} \text{ LIKE } n2.\text{NodeCode} +\_\% \).

As shown in this section, the relationship queries, which require the traversal of the pedigree graph, can be realized using pattern matching operations provided by SQL.

Pedigree query interface proposed in Chapter 4 uses the techniques covered in this section in order support following queries: ‘Ancestor’, ‘Children’, ‘Descendant’,

17

2.3.4 Path-Based Genetic Calculations with NodeCodes

NodeCodes encoding method is employed in genetic calculations to expedite the calculation time and improve scalability. The first example of this effort is inbreeding coefficient calculation by [13]. The similar ideas provided by [2] are used to calculate generalized kinship coefficients. As the identity coefficients are calculated via generalized kinship coefficients, in an implicit way, NodeCodes are used to calculate condensed identity coefficient, as well [9].

Given an individual $I$, inbreeding coefficient calculations require identification of $I$’s parents. Then, the common ancestors of the parents have to be found. Finally, based on the paths from the common ancestor to parents, the inbreeding coefficient can be calculated. As shown in [13], given the NodeCodes of $I$, $NC(I)$, it is possible to find common ancestors by using longest prefix matching sets. These matching sets also represent the paths from the common ancestor. Finally, these paths are processed as pairs in order to find the path-pairs that contribute to inbreeding coefficient calculation.

As reviewed in the Section 2.2, the generalized kinship calculations provided by PQV Tool are realized based on the path-based formulas proposed in [2, 9]. Cheng provides efficient and scalable calculations of these coefficients by employing NodeCodes in [9]. Therefore, PQV Tool employs NodeCodes to calculate these coefficients, as well.
In addition, Cheng provided an improved version of the condensed identity coefficient calculations in [9]. The improvement is achieved by reusing the calculated paths involved in similar $\Phi_{abc}$ and $\Phi_{abcd}$ calculations. In order to provide efficient condensed identity coefficients calculations, PQV Tool employs the improved condensed identity coefficients proposed in [9].

2.4 Related Work

In this section, the existing pedigree querying tools and their functionalities have been discussed. To the best of our knowledge, there exists no pedigree processing tool that can produce the type of pedigree queries proposed in this work. In addition, the relatedness calculations by using path-based formulas, such as generalized kinship (four individuals, pairs) and identity coefficients, are novel to this tool.

The academic work on pedigrees is mostly based on the aesthetic drawing of pedigrees or relatedness calculations. In one of the early works, PedHunter [15] is proposed as a pedigree verifying tool that checks the validity of the relationships between individuals. PedHunter provides some functions to find the relatives of given individuals, as well. Another example to early pedigree processing systems is named as PEDSYS [16]. PEDSYS provides couple of functions over pedigree data including inbreeding and kinship calculation.

PVIn is a pedigree processing tool in the literature that aims to draw large and complicated pedigrees [17]. This tool employs the capabilities of modern graphics to expedite pedigree drawing. Another proposed work, named Peditree [18], provides tree
like visualization of the pedigree data and performs inbreeding calculation. However, these tools lack building complex queries and identity coefficient calculations.

The pedigree data management system, called Cologene [19] provides a tree based query template. This template allows queries based on the data associated with the pedigree. Some structure-based queries are also available. However, the template used in Cologene is not dynamic and it does not provide complicated structure-based queries.

Commercial pedigree processing tools are also available. One example to popular tools is called Progeny [20]. Although this tool has extensive drawing support, the queries defined over the pedigree are basic Boolean queries and statistics over the associated data. Another commercial pedigree tool is Cyrillic 3 [21]. Cyrillic provides inbreeding calculation over pedigree data. However, no structure-based queries are supported in this system. In addition, Cyrillic uses MENDEL [22] system to calculate inbreeding coefficient. And MENDEL system is constrained by small and medium size pedigrees.

So far, the pedigree processing tools, that are standalone applications, are covered in this section. Recently, more web-based pedigree processing tools are proposed due to availability of internet and improvement on web application technologies. Some of these tools, such as [6], are solely for providing family history data in a portable way. Other tools such as [23] and [24] provide only online pedigree drawing support. One web-based tool that can support some relatedness calculations and some querying is called PedServe [25]. In PedServe, the inbreeding and kinship calculations are provided via another system, named FSpeed [26]. However, the queries provided in this tool are based mostly
on the data associated with individuals, and identity coefficients calculation is not provided.

2.5 Discussion

In this section, the necessary background information related to pedigree processing methods is discussed. This discussion covered pedigree labeling, pedigree queries and relatedness calculations. In addition, the previous pedigree processing tools are investigated. The capabilities of these tools are provided, with a discussion of the differences between those tools and the proposed pedigree query, visualization and genetic calculation tool.
Chapter 3 - Pedigree Query, Visualization, and Genetic Calculations Tool (PQV Tool)

So far, the introductory and background information related to pedigree processing methods are discussed. This section will introduce the PQV Tool and pedigree processing capabilities provided by the tool in a general way.

3.1 Overview

The overview of the PQV Tool will cover the following aspects of pedigree data processing:

- The query interface where users can form queries in a dynamic template. The query results will be displayed in different views.
- The genetic calculations interface where users can calculate inbreeding, kinship or identity coefficients of selected individuals.
- The visualization of the pedigree data as an interactive graph drawing.

This section is an overview of functionalities supported by PQV Tool. The details related to the usage of PQV Tool are covered by the user manual provided in Appendix A.

3.1.1 Query Interface

The query interface supported by PQV Tool is a dynamic template that allows user to build complicated queries via a graphical user interface. The overview of the query operations in PQV Tool is shown in Figure 3.1. User can run and analyze the results in an integrated environment as seen in this figure. The query results are updated...
in real-time whenever user issues a new query. PQV Tool allows user to save their frequently used queries, as well.

![PQV Tool: Pedigree query operations overview](image)

**Figure 3.1 PQV Tool: Pedigree query operations overview**

3.1.2 Genetic Calculation Interface

The overview of the genetic calculation operations in PQV Tool is given in Figure 3.2. Genetic calculation interface allows user to select the individuals, who are going to be used in calculations, via different data views. As an example, in this figure user selects the individuals on the pedigree drawing. Results are provided in a list form and they can be saved in a file.
3.1.3 Visualization

The pedigree visualization system provided in this tool has a different motivation than the other pedigree drawing systems. The main aim of the visualization system is to help query interface and genetic calculations interface. As an example, visualization system highlights selected individuals to help users to analyze query results in a more understandable way. In addition, the individuals used in genetic calculations can be selected via visualization. Therefore, visualization provided in PQV Tool is more than a static image.
3.2 Discussion

In this section we provided a brief introduction to PQV Tool. The capabilities of the PQV Tool are summarized based on different aspects of pedigree processing. In the next chapter, the pedigree query aspect of PQV Tool will be investigated in detail. The Chapter 5 will discuss the visualization aspect of the PQV Tool.
Chapter 4 - Pedigree Query Interface

This chapter focuses on the query interface provided by the PQV Tool. Chapter starts with a brief overview of the query interface. Then, the query semantics of the pedigree graph querying is discussed. Afterwards, query building interface is explained by providing example queries via the proposed visual query interface. Chapter concludes with the description of the query evaluation algorithm behind the query interface.

4.1 Overview

Pedigree query interface proposed in this work is inspired by the pedigree query language (PQL) [7]. PQL is an XPath [27] like path query language that enables querying both the structure of the pedigree and the associated data.

The proposed query interface enables the visualization of the query building by mapping query constructs of PQL to visual elements (query boxes). Therefore, knowledge of high level language or schema information is not required from the end-user.

Before diving into the details of the visual query interface, the query semantics related to the pedigree graph querying have to be explained. In the next section, the query semantics over a pedigree graph will be discussed in detail. Then, the visual query constructs and example queries will be provided in the following section.

4.2 Query Semantics over Pedigree Graph

A pedigree is defined as a DAG and denoted by $G = (V, E)$, where set $V$ (vertices or nodes) is the set of individuals and set $E$ (edges) is the set of parent-child relationships
between nodes. As an example, assume that there exists a directed edge \( \{ u, v \} \). Then, \( u \) is the biological parent of \( v \).

In this section, the query semantics over \( G \) are discussed. The discussion explains the semantics over a single pedigree graph. Although the PQV Tool can handle multiple pedigree graphs in the query interface, each pedigree graph is processed independently based on these semantics. Therefore, the rest of the discussion will be over a single pedigree graph.

**4.2.1 Query Input and Output**

The input of the pedigree query interface is the pedigree graph \( G \). In addition to the graph structure, a vertex \( u \) is associated with additional data. Therefore, both the pedigree graph structure and the associated data with each individual are the inputs of the query interface.

The output of the query interface is the subgraph of \( G \), denoted by \( G' \). The set of vertices in this subgraph is denoted by \( V' \), where each vertex \( u \) in \( V' \) has to satisfy some conditions. The semantics defined in this work employs set \( E \) when query is interested in the structure of the pedigree. However, the set \( V' \) is the main interest from the result perspective. In the following subsections, the semantics behind selection of the vertices that forms \( V' \) will be covered.

**4.2.2 Queries with a single query box - Type A Queries**

The concept *query box* is used to represent the vertices set \( V \) or \( V' \). A single query box can only be used to query the data associated with each vertex in the graph. This type
of queries are called Type A queries. However, the structure of the pedigree cannot be employed in the queries via a single query box.

The conditions based on the associated data are called *attribute-based* conditions. A query box can have zero or more attribute-based conditions where the set of attribute-based conditions denoted by $AC$.

As shown in Figure 4.1(A), a query box without attribute based conditions represents $V$. However, Figure 4.1(B) shows that a query box with attribute-based conditions represents $V'$ that consists of the nodes that satisfies the conditions.

![Diagram](image)

**Figure 4.1** (A) Type A query without attribute-based conditions. (B) Type A query with attribute-based conditions

In this subsection, the query box concept is defined and attribute-based conditions are introduced. In the following subsections, the semantics behind the queries that employ the pedigree structure will be covered.

### 4.2.3 Queries with two query boxes - Type B Queries

A single query box, which is denoted by $Q_1$, represents $V$ or $V'$. In this subsection, usage of another query box $Q_2$ and the query semantics that is represented by these two query boxes are discussed. This type of queries is named as Type B query.
The main motivation behind using more than one query box is to employ the structure of the pedigree in querying. More specifically, the existence of a path between a vertex \( u \) and some other vertices is the main interest of this type of queries. The paths in pedigree graph are based on relationships such as parent, child, sibling etc. As a matter of fact, although the edges in pedigree graph are directed, the paths can traverse the graph in both directions. The query semantics discussed in this section are applicable to pedigree queries as NodeCodes allows us to calculate paths by relationships without considering the direction of the path.

Let \( u_i \) be a vertex represented by query box \( Q_1 \). Given a path, which is denoted by \( P \), the existence of at least one vertex that is reachable by \( P \) from \( u_i \) determines the semantic behind structure-based queries. This path existence is employed as a condition and these conditions are named as structure-based conditions.

The vertices that are reachable from \( u_i \) via \( P \) are represented by \( Q_2 \). \( Q_2 \) can have attribute-based conditions, as well. The graphical representation of Type B queries is given in Figure 4.2. As seen in this figure, a tree analogy is used symbolize the set of vertices represented by different query boxes. The root of the tree, which is denoted by \( T_i \), is the vertex \( u_i \). The edges of \( T_i \) are same as \( P \). Therefore, the leaves of \( T_i \) are the vertices reachable from \( u_i \) via \( P \). Assume that there exist \( n \) trees, where each tree has a least one leaf. Then, \( Q_1 \) represents the roots of these trees whereas \( Q_2 \) represents the leaves of all trees.
As mentioned before, the query output has to be $V$ or $V'$. In order to keep the results as a set, the duplicate vertices that might be represented by $Q_2$ are eliminated. In addition, the set of vertices represented by $Q_1$, $Q_2$ or both can be the result of the query.

4.2.4 Queries with three or more query boxes - Type C and D Queries

In this section, the queries that require three or more query boxes are investigated. In the simplest case, assume that another query box, which is denoted by $Q_3$, is added over a Type B query. There are two possible ways to associate $Q_3$ with the other two query boxes in the proposed system. Based on this association, two different semantics can be represented by using $Q_3$. These types of queries are named as Type C and Type D queries. First, the type C queries are investigated.

**Type C queries**

As mentioned in the previous section, Type B queries can check the existence of a path between two query boxes. In Type C queries, $Q_3$ is chained with $Q_2$ to provide a second level for structure-based conditions.

The graphical representation and the tree analogy presented in the previous subsection are employed to describe the Type C queries, as well. In Figure 4.3, the graphical representation of Type C queries is given. As shown in this figure, $Q_2$ is
connected to $Q_1$ via $P_1$ whereas $Q_3$ is connected to $Q_2$ via $P_2$. Therefore, in this type of queries $T_i$ is a two level tree. Again, $T_i$ can exist as long as there is at least one node on each level of $T_i$. Then, $Q_1$ represents the roots of the trees. The nodes in the first level of all trees are represented by $Q_2$. And the leaves of all trees are represented by $Q_3$.

The set of vertices represented by any query box can be the output of Type C queries. In addition, two or three query boxes can be combined together to form the query result.

The main motivation behind the Type C queries is to chain $n$ query boxes together to form $n-1$ level trees. Any vertex set, that is represented a level in the tree, is accessible via the corresponding query box. This simply means that same output semantics would apply for any number of chained query boxes. The graphical representation of Type C queries, where $n$ query boxes are used, is given in Figure 4.4. In this figure, the dashed paths represent the connections provided by the other query boxes that are not included in the figure.

Figure 4.3 The graphical representation of Type C query semantics

31
Type D queries

In the previous section, the query boxes that are added to the systems are chained together. Instead of chaining the query boxes, Type D queries enables associating query more than one query boxes with the same query box by employing different paths. First, the simple case will be investigated where only $Q_3$ is employed on top of Type B queries. Then, the general form of the Type D queries will be discussed.

In this type of queries $Q_3$ is not chained to $Q_2$ via $P_2$. Instead, $Q_3$ is connected to $Q_1$ via $P_2$ whereas $Q_2$ is still connected to $Q_1$ via $P_1$. The graphical representation of Type D queries is shown in Figure 4.5. As shown in this figure, tree analogy applies to this type of queries, as well. In this type of query, $T_i$ is always one level tree where there is at least one leaf connected to root via $P_1$ and one leaf connected to root via $P_2$. Then, $Q_1$ represents the roots of such trees. $Q_2$ represents the leaves, which are connected to root via $P_1$, of all trees. And $Q_3$ represents the leaves, which are connected to root via $P_2$, of
all trees. Again, query results can be set of vertices represented by one query box, combination of two query boxes or three query boxes.

![Figure 4.5 Type D query semantics graphical representation](image)

An important point has to be cleared about Type D queries. When three query boxes are employed, it is possible to have equivalent query semantics with Type C and Type D queries. Type C query employs two paths $P_1$ and $P_2$. As long as it is possible to reverse the direction of a $P_1$, Type C query that uses three query boxes can be converted to a Type D query with three query boxes. An example is given in Figure 4.6. As shown in this figure, left hand side of the figure is a Type C query and on the right hand side the equivalent Type D query is provided. In the Type D query, the reversed version of the path $P_1$, which is denoted by $P_1'$, provides the same query semantics as Type C query does. However, during the conversion of Type C query to Type D query it is not possible to reverse only $P_2$ as it causes two incoming edges for $Q_2$. In addition, when both paths are reversed, another Type C query is obtained.

The Type D query can be converted into Type C queries as long as any of the two paths can be reversed. Each of the path reversals will lead to a different Type D query. However, reversing both paths is not allowed for Type D queries, as well.
Figure 4.6 The equivalence of Type C and Type D queries with three query boxes

However, this is a special case for three query boxes. As it is possible to generalize both query types with more than three query boxes, Type D queries are still necessary. In addition, in pedigree querying some paths are defined with the non-reversible relationships. There are two non-reversible relationships FDR and SDR. Sibling relationship is symmetric. And the reversible relationships are as follows: {Parent, Children}, {Grandparent, Grandchildren}, {Ancestor, Descendant}.

The Type D queries can be generalized to employ more query boxes. In a more clear way, $n$ query boxes can be connected to $Q_1$ by using $n$ paths. The generalized form of Type D queries does not have a corresponding equivalence in Type C queries or vice versa. The graphical representation of such generalization is given in Figure 4.7. As seen in this figure, $T_i$ is a one level tree with $n$ different edges (paths) and there should be at least one leaf node that can be reachable via the corresponding path.
Type C and Type D queries are provided to describe the general ways to utilize more query boxes in the system. However, the proposed system also provides a combination of these two query types. In a more general way, a query box $Q_k$ can have at most one incoming path and zero or more outgoing paths. An example to such query semantics is given in Figure 4.8. As seen in this figure, $Q_3$ and $Q_4$ are chained together as in Type C queries whereas $Q_2$ and $Q_3$ are connected to $Q_1$ as in Type D queries.
So far, the semantics of pedigree graph querying are described. In the next section, the visual query interface that follows the semantics defined in this section is proposed.

### 4.3 Visual Query Building Interface

In the previous section, the query semantics over pedigree graph are described. This section is the realization of those query semantics via a visual query interface. In this section, we will cover the query constructs that are provided by the proposed query interface. We will introduce the query constructs such as starting step, axis steps, conditions etc. in each subsection.

We will provide PQL constructs and type of the queries (Type A, B, C, D) along with the query boxes in order to provide better understanding on the query semantics. In addition, example queries over pedigree data will be provided along with introduced constructs.

#### 4.3.1 Query starting step

In the proposed query interface, all queries start with a set of individuals where all individuals in all pedigrees are included. This is represented by a single query box without any conditions. This representation simulates the ‘Individual’ keyword in PQL which is a query starting step where all individuals are included. The visual representation is shown in Figure 4.9. This is also an example to Type A query defined in the previous section.
PQL allows some other query starting steps such as ID*, where * represents a unique ID number. This construct enables starting with a specific individual. Query interface does not provide such steps. However, such steps can be obtained easily by defining a condition (based on ID attribute) on the Individual starting step.

Next, the conditions provided by query interface will be introduced. Query interface enables two types of conditions: attribute-based conditions and structure-based conditions. Attribute-based conditions utilize the data associated with pedigree whereas the structure-based conditions are related to the structure of the pedigree.

4.3.2 Attribute-based conditions

The attribute-based conditions are related to the data associated with each individual in a pedigree. Query interface uses a special data typing system over the underlying database schema. The data associated with the pedigree has to comply with the defined data types. Therefore, before diving into the details of attribute-based conditions, the data typing system is explained in the next subsection.

Query Interface Data Types

There are six different data types defined in the PQV Tool: ‘identity’, ‘string’, ‘integer’, ‘double’, ‘date-time’, and ‘categorical’. The data types defined by query interface are important since the queries based on the associated data rely on these types.
The main motivation behind these data types is to provide proper comparison operators and input fields during query building. In addition, the query interface validates user-input based on these data types.

The identity type is a numeric value which uniquely identifies an individual. Identity data type is treated differently than other numeric values in the system. Query interface provides a list of non-null identity values belonging to an attribute. This list prevents invalid user-input and helps to look up an identity value.

The data type string is used to present text based attributes. Numeric values are represented either with integer or double data type. The date-time type provides both the date (year-month-day) and time (hour-minute-second) information.

The last defined data type is categorical. Categorical is a special string-based type provided by the query interface. Categorical attributes can have one or more choices which are stored in the underlying database. Categorical attributes can be compared only with the choices pre-defined for that attribute. As an example, Gender is a categorical attribute, where female, male and unknown are the predefined choices. Some data types such as image, binary, byte or currency are not supported by query interface. However, query interface still covers the most commonly used data types.

The details of the relational tables, which are regulating the defined data types, are discussed in the Section 6.1.1, where the database schema behind PQV Tool is explained.
**Queries with attribute-based conditions**

An example query using attribute-based conditions is provided at the bottom of Figure 4.10. This type of queries are Type A queries with attribute-based conditions defined on the query box. This figure also shows how to add attribute-based conditions. As seen in this figure, PQL uses left ('[') and right (']') brackets to represent conditions. Conditions are placed just after the step name, in this example ‘Individual’. Each condition is defined as `<attribute name operator user-input>`.

All attributes belonging to *Individual* relation can be used in an attribute-based condition expression. Since user does not have direct access to the database schema, the query interface provides a list of attributes associated with Individual relation. As mentioned in previous subsection, any attribute that belongs to Individual relation has to
match with one of the data types defined in the system. Based on the data type, query
interface automatically provides comparison operators and input fields.

Query interface provides several operators that can be used to define an attribute-
based condition. All data types defined in the system supports the following operators:

Query interface provides pattern matching operations only for string data type.
Three different pattern matching operators are provided: ‘prefix’, ‘suffix’ and ‘contains’.
These operators are similar to the ‘LIKE’ operator in SQL. Actually, prefix represents
‘LIKE <user_input>%’, suffix can be expressed by ‘LIKE %<user_input>’ and contains
operator is a combination of both ‘LIKE %<user_input>’ where ‘%’ is a wildcard that
represents any number of unmatched characters.

One other special condition provided by the query interface is to query missing
data. Using ‘=’ operator with an empty user-input represents ‘(IS NULL or IS EMPTY)’
condition as in SQL. Similarly, ‘<>’ operator without user-input translates into ‘(IS NOT
NULL or IS NOT EMPTY)’ condition.

Based on these definitions, each attribute-based condition is a Boolean value.
Two or more attribute-based conditions can be combined to form a Boolean expression.
Logical conjunction operator (‘AND’) and disjunction operator (‘OR’) are the logical
operators supported among attribute-conditions. However, only one of the operators can
be activated at a time. This means that the queries such as ‘Find all individuals who has
cancer or older than 70 and male’ are not available.
Some operators are not supported for attribute-based conditions by the query interface. The operators such as ‘IN’, ‘BETWEEN’, or logical operator negation (‘NOT’) are not supported among attribute-based conditions. In addition, aggregation functions or arithmetic operators such as ‘+’, ‘*’ and so on cannot be used with the attribute-based conditions. Query interface is kept simple and concise as an implication of excluding these operators.

4.3.3 Axis steps

In order to navigate through the structure of the pedigree, nine axis steps (relationships) are defined in the system: ‘Ancestor’, ‘Children’, ‘Descendant’, ‘Grandchildren’, ‘Grandparent’, ‘First Degree Relative (FDR)’, ‘Parent’, ‘Sibling’, and ‘Second Degree Relative (SDR). These are fundamental and frequently used relationships in pedigree domain.

These steps can be combined to form paths that cannot be represented by a single step. Some steps such as FDR or SDR represent the union of other fundamental steps. FDR includes parents, children and sibling sets. SDR consists of grandparent, grandchildren, uncle, aunt, nephew and niece. These two steps are provided as it is not convenient for user to form them and it is error prone to do so due to number of unions needed.

On the other hand, PQL defines five basic steps: Parent, Child, Ancestor, Descendant and Self. A slash sign ‘/’ is used before each step. As an example ‘Individual[ ID = 2474 ]/Ancestor’ returns all ancestors of the individual with ID 2474. The representation of this query is given in Figure 4.11.
The first query box (with number 1) represents the starting step, which is the Individual step. On this query box, user defines a condition with \textit{ID} attribute. Then, user creates another query box (denoted by 1/1) by selecting ‘Ancestor’ step as shown in Figure 4.11. As seen in this figure, this is accomplished by using adding a structure-based condition on the first query box. More clearly, the query box 1 is being conditioned by query box 1/1. However, query box 1/1 solely represents the traversal of the pedigree. This type of queries is the Type B query which is defined in the previous section.

![Figure 4.11 Navigate through the pedigree via a single step (Type B)](image)

As mentioned above, complex \textit{paths} are formed by employing more than one step in some queries. One example to this type of query is ‘Find cousins of the individual with ID 2474’. The representation of this query via proposed interface is given in Figure 4.12. As seen in this figure, query box 1/1 is the main interest of this query. Again, query box 1/1 is used as a structure-based condition on query box 1. This type of queries is still
Type B query as two query boxes are employed. However, the path has more than one step.

4.3.4 Structure-based conditions

So far, we have covered the starting step, attribute-based conditions and axis steps. The paths that are formed by using multiple steps have been shown. In this section, we will focus on the structure-based conditions.

In PQL, this type of conditions are defined by using path expressions inside left (‘[‘) and right (‘]’) brackets. Path expressions start with one of the defined steps (except starting step) and can have conditions on them. An example to this type of query is ‘Find all individuals who have at least one ancestor with cancer’. The PQL representation of this query is: ‘Individual[/Ancestor[ HasCancer = True ]]’. As seen in this example, the
path expression used inside the condition is relative to the previous step, which is
*Individual* in this example.

The proposed query interface follows the same notation. Every query box is a
relative of another query box. This is the main motivation behind numbering the query
boxes with 1, 1/1 etc. In more general way, every query box in the interface has one
*parent* query box (except the query box 1 (starting step)) and every query box can have
zero or more *child* query boxes. Child query boxes passes their path expression to parents
as a condition and this type of condition is called structure-based conditions.

The realization of the example query ‘Find all individuals who have at least one
ancestor with cancer’ is given in Figure 4.13. As seen in this figure, query box 1/1 passes
its path expression ‘/Ancestor[ HasCancer = True]’ to parent query box. The query box 1
and query box 1/1 have different query semantics. As mentioned in previous section,
query box 1/1 uses query box 1 as a step in pedigree traversal. And query box 1 uses
query box 1/1 as a condition. Therefore, each query box represents a different query as
shown in this figure. The query interface allows user to select any query box as a result.

![Diagram](image.png)

*Figure 4.13 Structure-based condition example where a single query box used as a condition
(Type B)*
The path expressions can be used as a condition inside another structure-based condition in a nested way and this corresponds to the Type C queries. The following is an example to such queries: “Find all individuals who have at least one person with cancer in the ancestors of father-side”. This query is represented in PQL as follows: “Individual[/Parent[(Gender = Male) and (/Ancestor[HasCancer = True])]]”. As seen in this PQL representation, ‘/Ancestor[HasCancer = True]’ is used as a condition for ‘/Parent’ step, which is also a condition for starting step (Individual). Again, every path expression is relative to the previous step. This type of queries is provided by the proposed interface, as well. The example query is realized by query interface in Figure 4.14. As seen in this figure, the path expression of the query box 1/1/1 (marked with blue ellipse) is passed to query box 1/1. The path expression of query box 1/1 (marked with pink ellipse) is passed to query box 1 based on the same logic. And each query box in this figure represents a different set of individuals, as usual.

![Figure 4.14 Nested path expressions example in the query interface (Type C)](image-url)
The structure-based conditions are not limited with a single path expression. In query interface more than one path expression can be used in a structure-based condition. The conjunction logic (‘and’) is used between the path expressions.

The following query is an example of multiple path expressions in a condition: “Find all individuals who have at least one parent with cancer and one child with cancer”. This type of query is an example of Type D queries. In PQL, this query is represented as “Individual[/Parent[ HasCancer = True ] and /Child[ HasCancer = True ]]”. The representation of this query in the proposed interface is given in Figure 4.15.

An important point related to the PQL representation has to be mentioned here. In Figure 4.15, the PQL representation of query box 1/1 is given as follows: “Individual[/Child[ HasCancer = True ]/Parent[ HasCancer = True ]]”. In this expression, the starting step is conditioned with the path expression of query box 1/2 (shown with

![Diagram of structure-based condition with multiple path expressions (Type D)](image-url)
blue arrow in the figure). Then, the second step (marked with pink ellipse in the figure) is added. The main reason behind this semantic is the conjunction logic used between the path expressions in query box 1. Based on this, the PQL representation of query box 1/1 can also be described as follows: Individual[/Parent[ HasCancer = True ] and /Child[ HasCancer = True ]]/Parent[ HasCancer = True ]. However, query interface removes the underlined section to make representation more concise. Same logic applies to query box 1/2, as well.

**Equivalence of Type C and Type D queries**

As mentioned in section 4.2.4, Type C and Type D queries can be equivalent when three query boxes are used. A Type D query can be converted into Type C form as long as one of the paths is reversible. An example is provided in Figure 4.16. In this figure, the example query from Figure 4.15 (a Type D query) is expressed by using a Type C query. The blue arrows point out the semantically equal query boxes. In this
example, the relationship *parent* is reversed as *children* in order to obtain Type C equivalent query. The query boxes selected as ‘output’ in both type of queries return the same set of individuals. However, it is easier to keep track of the query in Type D form as Type D query centers the ‘output’ query box as a starting step.

**Example Queries where Type C and Type D are not equivalent**

In this sub-section, an example query that can be represented with only Type C form and has no correspondence in Type D form will be given. Then, a similar non-convertible example query will be provided for Type D queries, as well.

As long as the first relationship that is used after the starting step is not reversible, Type C queries do not have a correspondence in Type D form. An example query is as follows: “Find all individuals who have at least one female FDR who has at least one child with cancer”. The graphical representation of this query is provided in Figure 4.17(A). As the relationship ‘FDR’ is not reversible this example query cannot be represented with a Type D query. In Figure 4.17(B), the following example query is provided: “Find all individuals who has at least one FDR with cancer and one SRD with cancer”. This example is a Type D query and Type D queries can be represented in Type C as long as one of the relationships is reversible. As given in this example, both FDR and SDR relationships are not reversible. Therefore, this Type D query cannot be represented in a Type C form.
Complex Queries with more than three query boxes

Path expressions can be combined to provide more complicated queries. Up to now, we have discussed nested path expressions and the multiple path expressions in conditions. Next, an example query that employs the combination of these two types of conditions is explained.

Assume that a query as follows is given: “Find the daughters with cancer of female individuals who has at least one person with cancer in the female ancestors on mother side”. The representation of this query by using query interface is given in Figure 4.18. As seen in this figure, the query box 1/1 describes the example query. In addition, every other query box represents another meaningful query. The pink marked path
expressions are passed by query boxes 1/1 and 1/2 to query box 1. The blue marked path expression is passed from 1/2/1 to 1/2 as a path expression and this result in a nested path expression on query box 1/2. The yellow marked path expressions are passed from query box 1 due to multiple path expression usage on this query box.

Figure 4.18 Example query that combines multi-path expressions and nested path expressions in a query (Type C and Type D combined)

4.3.5 Selecting more than one query box as output

As shown in the previous examples, each query box has a checkbox (‘output’). Query interface unions the query boxes that are marked as output to produce the final set. This functionality enables queries such as “Find all female individuals with cancer who have at least one female ancestor with cancer along with her ancestors”. This query is represented by the query interface as shown in Figure 4.19. As seen in this figure, both query boxes are marked as output. Therefore, the set of individuals represented by each query box are merged into a single set of individuals, where only distinct individuals are kept.
The query building constructs that are introduced so far can be combined to represent more complicated queries. Formal definition of the queries that can be represented by a query box using Extended Backus-Naur Form (EBNF) notation is given in the next section.

4.3.6 EBNF notation of the provided pedigree queries

So far, the query building constructs are described using PQL representation and example queries. This effort provides reader a good understanding of the different types of queries that can be answered via the query interface. However, it is not possible to cover all query cases with this approach.

In this section, we employed EBNF notation to define PQL queries that can be represented by a single query box. There are two important points that have to be mentioned. Firstly, EBNF notation does not cover queries where more than one query
box is employed. The PQL representations of the query boxes that are used together are dependent on each other and EBNF supports only context-free problems. Second point is related to the attribute-based conditions. EBNF also cannot cover this type of condition since operators and user-inputs are dependent on data type of an attribute. Therefore, we abused EBNF notation to cover attribute-based conditions. EBNF notation of the PQL queries that can be represented by a single query box is as follows:

```plaintext
query = start_step [ condition ] {step}

condition = “[” attr_cond “]” | “[”structure_cond “]”

| “[” attr_cond “and” structure_cond “]”

step = step_type [condition]

path_exp = step_type { step_type } [condition]

attr_cond = boolean_val { “and” boolean_val}

structure_cond = path_exp { “and” path_exp }

boolean_val = column_name operator user_input

step_type = “/Ancestor” | “/Children” | “/Descendant” | “/Grandchildren” |

“/Grandparent” | “/FDR” | “/Parent” | “/Sibling” | “/SDR”

start_step = “Individual”
```

In this EBNF representation, terminals are described in between two double quotation symbols. ‘[ ]’ notation represents an option that an element can be used once or
not used at all. The ‘{ }’ notation describes repetition, where an element can be repeated zero or more times.

As seen in this notation, non-terminal boolean_val, which is used by non-terminal att_cond, has three undefined non-terminals: column_name, operator and user_input. However, the representation still can express queries in a meaningful way.

4.3.7 Operations over query result

There are some operations that are not supported on each query box but can be used over the query result. As mentioned above, the query result consists of a single query box (representing a set of individuals) or two or more query boxes that are merged into a single set of individuals.

**Projection:** Each selected individual represents a row in Individual relation. This relation might contain several fields and only a subset of these fields might be the interest of the end-user. Therefore, query interface allows user to select which attributes will appear in the query result table. There is one exception to this selection. Query interface always projects ID attribute to match the rows with missing data.

**Aggregation:** The available aggregate functions defined in the query interface are AVG(), Count(), MAX(), MIN(), and SUM(). These functions are used to gather simple statistics about the query results. Any aggregate function can be used over any attribute belonging to the Individual relation. An example to this usage is ‘Find the average age of individuals with cancer’. 
4.3.8 Unavailable Query Constructs

In this section, the unsupported query constructs provided by PQL are discussed. During the discussion, the limitations behind the unavailable constructs are investigated. Moreover, alternative ways to represent some of the unavailable constructs are explained.

The ‘Self’ step is not available in the query interface. This type of step replicates the set of individuals represented by the previous step. On the other hand, any step provided in the proposed query interface is a relationship. In order to keep the query semantics simple this type of step is not included in the query interface.

In PQL, the ‘|’ symbol denotes alternator operator. This operator allows queries that have to follow multiple steps on a single step. As an example: "/Individual[ ID = 2474]/(Children | Ancestor)" returns all the children and ancestors of the individual with ID 2474. This operator is not available in the interface due to the following reasons. In the query interface, steps are represented via query boxes. Any step created in the query interface has to be a condition to previous step. Therefore, it is not possible to follow a previous step via two different steps without conditioning the previous step. In addition, the steps that are used with alternator operator can be followed by other steps such as “/Step1/(Step 2 | Step 3)/Step4”. This requires a visual query interface where query boxes can be merged into a single query box. As the proposed query interface has a tree form, this is not possible.

In PQL, gendered steps are provided as a postfix on a step. The Female, Male, or Unknown postixes are denoted as "{F}". "{M}" and "{U}" respectively. As an example: “Individual[ ID = 2474 ]/Parent{F}” returns the mother of individual with ID
Query interface treats gender steps as regular attribute-based conditions as *Gender* is a mandatory field in the *Individual* relation. Therefore, same query can be represented as “Individual[ ID = 2474 ]/Parent[ Gender = Female ]”.

PQL provides the same aggregate functions discussed in the previous section. However, the aggregate functions provided by PQL can be used in path expressions, as well. For example: “Individual{M}[ COUNT(/Ancestor[ HasCancer=True ]>3)]” returns the male individuals who have more than 3 ancestors with cancer. In this example, aggregate function used with a path expression as a condition. In the proposed query interface, this type of queries is not supported in order to keep each query box as simple as possible.

### 4.4 Query Interface Evaluation

The query template has to be parsed and evaluated by the system to form the actual SQL query, which can be executed against the database. In order to achieve this final result, first the selectivity conditions are defined for the query boxes. Then, the necessary high-level query parsing algorithm which satisfies the selectivity conditions is given. Finally, the mapping between high-level data structures used in the algorithm and relational algebra is discussed in detail.

#### 4.4.1 Terminology

The following definitions will be used while explaining the selectivity conditions, query evaluation algorithm, and mapping to the relational algebra. Additional definitions are introduced under specific sections, as needed.

\[ Q_k \] : A query box in the query interface.
\( PI_k \): Initial individual set from parent query box of \( Q_k \)

\( I_k \): Initial individual set of query box of \( Q_k \)

\( P_k \): The path between each individual in \( PI_k \) and \( I_k \)

\( AC_k \): The set of attribute-based conditions of \( Q_k \)

\( Q_j \): A child query box of \( Q_k \)

### 4.4.2 Selectivity Conditions

Two selectivity conditions are defined based on the semantics of the query interface. Both conditions have to be satisfied by a selected individual. Therefore, the conjunction logic is used between two types of conditions.

Condition one: All attribute-based conditions must be satisfied by \( I_k \).

Condition two (structure-based): The path \( P_j \) has to exist between a selected individual in \( I_k \) and a selected individual in \( I_j \).

The structure-based condition has a recursive nature where the selectivity of the individual in \( Q_k \) depends both on the \( Q_j \) and the parent query box. This issue is addressed by using a depth-first traversal algorithm which satisfies both conditions for \( Q_k \). The details of this algorithm will be covered in the next section.

### 4.4.3 Query Template Parsing Algorithm

Some additional definitions have to be provided before explaining the algorithm. These additional definitions are based on the previous terminology defined in the Section 4.4.1.

**Intermediate Sets:** These are the constructs that are used in the intermediate steps of the algorithm in order to achieve the final output.
$I_k'$: Individual set of query box $Q_k$ after the attribute-based conditions applied.

$I_k''$: Individual set of query box $Q_k$ after the attribute and structure-based conditions applied.

**Output:** These constructs are the output of the algorithm.

$PI_k'$: The set of individuals in $PI_k$ where $P_k$ exists between $PI_k$ and $I_k''$.

$JR_I_k$: A joined relation where each column represents the selected set of individuals, $I'$, in query boxes visited in depth-first order starting from $Q_k$.

**Algorithm:** Apply attribute and structure-based conditions in a depth-first order.

**Input:** $Q_k$.

**Output:** $PI_k'$, $JR_I_k$.

1- Parse_Query_Template ($Q_k$)
2- $I_k' = $ Apply_Attribute_Conditions($I_k, AC_k$)
3- $I_k'' = I_k'$ // Initialize the structure-based condition applied set
4- $JR_I_k = I_k''$ // Initialize the relation with a single column
5- For each child query box of $Q_k$
6- // $Q_j$: A child query box of $Q_k$
7- Initialize_Q ($I_k'$)
8- $PI_j', JR_I_j := Parse_Query_Template ($Q_j$) //recursive call
9- //structure-based conditions via join
10- $I_k''$, $JR_I_k := Join_Relations (I_k'', JR_I_k, PI_j', JR_I_j)$
11- $PI_k' = $ Set of individuals in $PI_k$ where $P_k$ exist between $PI_k$ and $I_k''$
12- return $PI_k'$, $JR_I_k$
13- Apply_Attribute_Conditions($I_k, AC_k$)
14- $I_k' = $ apply all attribute-based conditions in $AC_k$ on $I_k$
15- return $I_k'$
16- Initialize_Q ($I_k''$)
17- $I_k = $ set of individuals that are reachable by following $P_j$ from $I_k''$
18- $PI_j = $ The individuals in $I_k''$ where $P_j$ exists between $I_k''$ and $I_j$
19- Join_Relations ($I_k'', JR_I_k, PI_j', JR_I_j$)
20- $JR_I_k = JR_I_k$ join $JR_I_j$ where $I_k'' = PI_j'$
21- $I_k'' = $ The individuals in $I_k''$ where $I_k'' = PI_j'$
22- return $I_k''$, $JR_I_k$

**Initial call:** The $Parse_Query_Template$ function has to be called initially using the root query box $Q_r$, as a parameter. There are some special cases that have to be pointed
out for the $Q_r$. The $Q_r$ does not have a parent query box. Therefore, $PI_r$ and $P_r$ are null. In addition to this, $I_r$ is the set of all individuals in the system.

**Result:** As $Q_r$ is used for initial call, the final result of the Parse_Query_Template is $PI_r'$ and $JR_I_r$. The $PI_r$ is null, so $PI_r'$ is null, as well. $JR_I_r$ is the relation where each column represents the selected individuals set, $I''$, of query boxes visited in depth-first order starting from $Q_r$. Therefore, the $JR_I_r$ is the only necessary structure to find selected individuals.

**Explanations:** In the first step of the depth-first parsing algorithm, the attribute-based conditions are used to eliminate some of the individuals from $I_k$. The set where attribute-based conditions are applied is denoted as $I_k'$. $I_k'$ is used to initialize $I_k''$, which is the attribute and structure-based conditions applied initial set.

There has to be the path $P_j$ between a selected individual in $I_k$ and a selected individual in $I_j$ as the second condition requires. Initially, this is assured since $I_j$ is initialized by $I_k''$ via $P_j$. However, after initialization, both $I_j$ (converted to $I_j''$) and $I_k''$ (further eliminated by other child query boxes) might be updated. This simply means that they are not finalized, yet. In order to keep track of the updated sets, algorithm employs different constructs. When $Q_j$ is processed by the recursive call, $I_j$ is converted to $I_j''$. And $Q_k$ uses $PI_j'$ set to eliminate $I_k''$ due to the path existence constraint. Algorithm uses a relation denoted by $JR_I_k$ in a join operation to satisfy the second selectivity condition. In this operation, $PI_j'$ is used as join predicate and $JR_I_j$ is the other relation used in the join. Each join operation over $JR_I_k$ guarantees that any elimination on $I_k''$ is propagated to each $I''$ column kept in $JR_I_k$ relation.
The *join_relation* function on the 19th line of the algorithm is responsible for keeping $JR_I_k$ and $I_k''$ up to date. In this function, the join operator in relational algebra is abused to simplify the algorithm. On the 20th line of the algorithm, the $JR_I_j$ relation does not have the column $PI_j'$. However, this operation is still valid as $PI_j'$ is connected to $I_j''$ via $P_j$. One other important point is about the 21st line of the algorithm. The regular relational join operator naturally eliminates the $I_k''$, which is the first column of $JR_I_k$. As the join operation is abused, 21st line of the algorithm is added to make the intention of the operation clear, not to add redundancy.

Once the depth-first traversal of the query template is over, the $JR_I_r$ will contain $m$ columns, where $m$ is the number of query boxes in the query interface. The columns in $JR_I_r$ correspond to the $I'$s of all query boxes. The order of the columns follows the order of depth-first traversal of the query template. Based on this order, the query boxes that are marked as output are retrieved and union operation is used to form the final result.

So far, query interface semantics, the formal selectivity conditions and the high-level query template parsing algorithm have been provided. However, some of these high-level structures and operations are not suitable for relational databases. Hence, in the next section by using relational algebra, the high-level constructs, that have been employed so far, are realized.

### 4.4.4 Mapping to Relational Algebra

First, some specific definitions that are used in this section are provided. These definitions are based on the terminology defined in the Section 4.4.1.
$R_k$: is a relation, where first column represents $PI_k$ and second column corresponds to $I_k$.

$R_k'$: is the relation, where attribute-based conditions are applied on $R_k$.

$P_k$: is the path that consists of $n$ relationships, where $n > 0$.

$R_k''$: is the relation, where first column represents $PI_k''$ and second column corresponds to $I_k''$. The rest of the columns represent the selected set of individuals, $I''$, in query boxes visited in depth-first order starting from $Q_k$.

In the Section 4.4.1, two sets of individuals $I_k$ and $PI_k$ are defined for $Q_k$. Each individual in $I_k$ has a corresponding individual in $PI_k$ where $P_k$ exists between these individuals. All individuals in $I_k$ and $PI_k$ are elements of the $ID$ column in Individual $(IND)$ relation. In order to realize $I_k$ and $PI_k$ in the relational algebra, we will define a relation $R_k$, where first column represents $PI_k$ and second column corresponds to $I_k$. Each row of this relation will represent the connected individuals by $P_k$. The first and second column of $R_k$ is named as $PI_k$ and $I_k$, respectively.

Attribute-based conditions have to be applied on $R_k$, if there exists any for $Q_k$. In order to apply the attribute-based conditions, the relation $R_k$ has to be joined with relation $IND$. After the join operation, the attribute-based conditions can be applied on the joined relation. Finally, $I_k$ and $PI_k$ columns are projected since the other columns are not needed any more. The relation formed after applying the attribute-based conditions is denoted as $R_k'$, where $R_k' = \pi_{PI_k,I_k}(\sigma_{AC_1 \text{and} AC_2 \text{...and} AC_j}(R \bowtie_{I_k = ID} IND))$. One important point is the first and the second columns of $R_k'$ are still named as $PI_k$ and $I_k$, respectively.
$Q_k$ uses $R_k''$ to create $R_j$ for a child query box $Q_j$ by using $P_j$. The initial $R_k''$ equals to $R_k'$. $R_k''$ is transformed into $R_j$ by using the methods discussed in the Section 2.3.3. As an example, assume that $P_j$ has only one relationship, such as children. For this example, the following SQL query returns $R_j$:

$$Q_{\text{children}}: \text{SELECT } n1.\text{IndividualID} \text{ AS } PI_j, \text{ n2.IndividualID } I_j \text{ AS FROM NodeCodes n1, NodeCodes n2 WHERE n1.IndividualID IN (SELECT I_k FROM R_k'')} \text{ AND n1.PedigreeID} = \text{n2.PedigreeID AND n1.NodeCode}+\_\_\_ \text{LIKE n2.NodeCode}$$

However, there might be more than one relationship in $P_j$. In this case, $R_j$ is not finalized until the last relationship is processed. Again, $R_k''$ is employed to initialize $R_j$ by processing the first relationship on the path as shown in example $Q_{\text{children}}$. The next relationship is processed by using $R_j$ (not $R_k''$) and result is saved on a temporary table $R_{\text{temp}}$. Then, the following join operation is applied:

$$R_j = \rho_{I_j/R_{\text{temp}}} (\pi_{PI_j,R_{\text{temp}}} (R_j \bowtie_{I_j=PI_{\text{temp}}} R_{\text{temp}})).$$

As shown in Figure 4.20, $R_{\text{temp}}$ is used in order to keep the original $PI_j$ and to replace the $I_j$ with $I_{\text{temp}}$, where $I_{\text{temp}}$ is the individual set, created after processing the relationship. Same logic is applied for the rest of the relationships in the $P_j$. 

61
In order to apply the structure-based conditions, high-level algorithm uses two different constructs on $Q_k$: $I_k''$ and $JR_I_k$. However, in relational algebra notation only the relation $R_k''$ is sufficient to emulate these constructs. As shown in the high-level algorithm described in the previous section, structure-based conditions are defined via joins. The following join operation emulates the same logic used in the high-level algorithm: $R_k'' = R_k'' \Join \rho_{P_{I_j}/I_k}(R_j)'$. In this operation, the second relation that is need for join ($R_j''$) is obtained after processing $Q_j$. The same logic is used with all other child query boxes of $Q_k$. Based on this, once all child query boxes are processed, $R_k''$ represents all the high-level structures $PI_k'$, $I_k''$ and $JR_I_k$.

As the initial call is established by using $Q_r$, the final output of the algorithm is $R_q''$. $R_q''$ holds $m+1$ columns, where $m$ is the number of query boxes in the query interface. The first column is ignored, which is $PI_q'$. As shown in the previous section, the remaining columns in this relation have a particular ordering, which is order of depth-first traversal. Therefore, for each query box the column representing the IDs of selected individuals can be projected or can be used in join operations based on this order.
4.5 Discussion

In this section, the semantics of the pedigree graph queries that are available via the query interface are described. Then, the actual visual query interface that follows these semantics is introduced. The pedigree query language and EBNF notation are employed to provide a solid understanding of the queries supported by the visual query interface. The chapter is concluded with the query evaluation algorithm behind the query interface.
Chapter 5- Visualization

This chapter focuses on the visualization aspect of pedigree data processing. First, the motivation behind the pedigree data visualization interface is described. Then, the provided user interactions with the visualization are covered. Finally, the graph layout behind the pedigree visualization interface is explained.

5.1 Pedigree Data Visualization

In this thesis, the pedigree data is modeled as DAG, where each node can have at most two incoming edges and zero or more outgoing edges. The main motivation behind the visualization interface provided by PQV Tool is to display the pedigree graph in an interactive and readable way. In Figure 5.1, the visualization of a sample pedigree is shown. The structure of the sample pedigree is obtained from real data [28].

![Figure 5.1 The visualization of the pedigree with ID 327](image)

As reviewed in the Section 2.4, the pedigree visualization systems are following some aesthetic drawing rules. These rules are helping genetic counselors to conceive the information behind the drawing in an intuitive way. These aesthetic pedigree drawing rules are defined in [29]. However, a pedigree drawing system that is following these rules is out of the scope of this thesis.
Although the pedigree visualization tool does not follow the aesthetic drawing rules, other functionalities are introduced to provide a visualization system that is more than a drawing. In the next section, the functionalities provided by the interface are covered.

### 5.2 Visualization Interactions

In this section, the interactive features of the pedigree visualization interface are covered. In each subsection, an interaction provided by the visualization interface is discussed. First, the highlighting mechanism for query results is explained. Then, edge highlighting for parents and children of an individual is introduced. Finally, the operations provided for each individual on the visualization interface are described.

#### 5.2.1 Query Result Highlighting

In PQV Tool, query interface returns a set of individuals. These individuals can belong to one or more different pedigrees. The pedigree visualization interface highlights the selected individuals in a pedigree. An example query result highlighting is shown in Figure 5.2. As seen in this figure, the selected individuals in pedigree with ID 327 are highlighted with red borders.
Figure 5.2 Query result highlighting mechanism in pedigree visualization interface

This mechanism allows analyzing the query results within the structure of the pedigree. Therefore, the structure of the pedigree and the query results are viewed in the same interface.

5.2.2 Parent and Children Highlighting

Pedigree visualization interface provides edge highlighting for each individual. The edges between an individual and the corresponding parents are highlighted with red color whereas the edges between an individual and the corresponding children are highlighted with blue color. Pedigree visualization with edge highlighting for individual with ID 2474 is shown in Figure 5.3.
This mechanism improves the readability of the pedigree visualization since there are crossing edges on the graph. In addition, the mechanism is easy to use since it is triggered when user hovers over the individual.

### 5.2.3 Operations over the Visualization

In PQV Tool, any operation that can be used over a single individual is accessible via the visualization interface. These operations are as follows: inbreeding coefficient calculation and relationship queries. The relationship queries, which are provided via the visualization interface, are the same relationships defined in the Section 4.3.3. The operations that can be used with an individual are shown in Figure 5.4.
5.3 Graph Layout

In this section, the desirable properties of a pedigree drawing system are introduced. Then, the Sugiyama layout [30] is reviewed. The properties of this layout are described in order to present the motivation behind preferring this layout for pedigree visualization interface.

5.3.1 Pedigree drawing properties

As discussed above, the visualization interface provided with PQV Tool is not supporting all the aesthetic pedigree drawing rules that are defined in [29]. However, visualization interface provides readable pedigree drawings by following some of aesthetic drawing rules reviewed in [19]. Therefore, the aesthetic rules that are employed in the query visualization interface are discussed in this section.

Visualization interface employs the following rules during pedigree drawing:

- Nodes on the graph should not overlap.
- Nodes in the same generation should be positioned on the same level.
In order to accomplish these goals, visualization interface employs a graph layout over the drawing. In the next section, this graph layout is discussed in detail.

5.3.2 Sugiyama Layout

Sugiyama layout is proposed as a heuristic method in order to display hierarchy, which is defined as a multi-level directed graph [30]. This layout is applicable only on acyclic graphs.

As pedigrees are defined as DAGs and the generations can be considered as levels, Sugiyama layout is suitable for pedigree visualization. Moreover, this layout has the following properties:

- Vertices on the same level are positioned on a straight horizontal line.
- Vertices are permuted in order to decrease the number of crossing edges.
- The edges are kept straight.
- The shortest paths are provided as long as there is no compromise on the vertex positioning.

Based on these properties, Sugiyama layout is preferred in this work as it provides the pedigree drawing properties discussed in the previous section.

5.4 Discussion

In this chapter, the pedigree visualization aspect of the PQV Tool is covered. First, the motivation behind the visualization interface is explained. Then, the interactions between the visualization and other PQV Tool components are discussed. Chapter is concluded with the graph layout method behind the visualization interface.
Chapter 6- Implementation Details

In this chapter, the technologies behind the PQV Tool will be explained. In each section, a different aspect of the implementation will be discussed. Firstly, the database engine that is used by PQV Tool is explained. Then, the programming framework and the design pattern, that are used to develop the PQV Tool, are discussed. Chapter concludes with the overview of the libraries that are employed in PQV Tool.

6.1 SQLite

SQLite [31] is a lightweight, serverless database engine that serves behind the PQV Tool. The source code of this database engine is in the public domain. In addition, SQLite supports transactional SQL. In this work, SQLite version 3.7.12 is used.

SQLite is chosen since it requires no configuration at all. Therefore, users can download the PQV Tool and start using it without any database configuration.

6.1.1 Database Schema

The database schema used with PQV Tool is provided in Figure 6.1. Any SQLite database file that complies with this database schema can be used with PQV Tool.

PQV Tool creates an empty database by following the database schema if user does not provide any database file. In addition, when user supplies a database file the NodeCodes relation is populated by the tool. However, PQV Tool does not overwrite the NodeCodes relation, when there exist some tuples in the relation.
The data types used in the Section 4.3.2 is regulated by the Column relation that is shown in Figure 6.1. The *DataType* column in this relation determines the associated data type with the column. In addition, the *categorical* columns have their corresponding *categorical values* in the *CategoricalValue* relation. Any additional attribute defined on Individual relation must have a corresponding column definition in the Column relation. Otherwise, PQV Tool does not provide access to the attribute in query interface.

### 6.2 Windows Presentation Foundation (WPF) Framework

Windows Presentation Foundation (WPF) framework is one of the latest technologies provided by Microsoft© [32]. WPF enables the development of standalone (desktop) or browser-hosted applications. By taking the advantage of modern graphic
hardware, fast and highly-visual user experiences can be developed in this framework. WPF is provided as a part of .Net 4.0 framework [33]. Therefore, it is possible to integrate other class libraries provided in the .Net framework with WPF applications.

In WPF development, user interfaces implemented via the mark-up language, named Extensible Application Markup Language (XAML) [34]. The business logic behind the user interfaces are implemented in code-behind files. In code-behind files, programming language C# is used. In the next section, XAML is reviewed in a briefly.

6.2.1 Extensible Application Markup Language (XAML)

In WPF applications, XAML allows developers to create user interfaces in a simplified way. XAML provides the separation of user interface design and the business logic, where business logic is handled in code-behind files. In Figure 6.2, a XAML code snippet is provided. As seen in this figure, user interface constructs such as button, stackpanel can be implemented in an intuitive way.

```xml
<GroupBox Header="Query Builder" Grid.Row="1">
  <TreeView ItemsSource="[Binding Path=TreeRoot]" VirtualizingStackPanel.IsVirtualizing="True"
  VirtualizingStackPanel.VirtualizationMode="Recycling">
  
  </TreeView>
</GroupBox>
<StackPanel Orientation="Horizontal" Margin="6,3,6,3" Grid.Row="2">
  <Button Command="[Binding Path=QuerySaveCommand]" Margin="0,0,0,0" CommandParameters="Query">Save Query</Button>
  <Button Command="[Binding Path=ShowSavedQueriesCommand]" Margin="0,0,0,0" CommandParameters="Saved Queries">Saved Queries</Button>
</StackPanel>
```

Figure 6.2 The XAML example from PQV Tool implementation

6.3 Model - View - ViewModel (MVVM) Pattern

In order to have a fast and robust development cycle, it is desired to separate business-logic and database operations from user-interface. So, it would be possible to focus on these different aspects of the system separately. In PQV Tool implementation,
the architectural pattern called Model-View-ViewModel (MVVM) is employed to accomplish this goal [35].

The main motivation behind the MVVM pattern is to represent the database objects as a Model and user interface components as a View. The bridge between these two parts is named as the ViewModel. The main responsibilities of a ViewModel are processing commands from user interface to modify data and retrieving necessary data from the database. The ViewModel is also responsible of executing the business logic behind the View. The interaction between these three components is shown in Figure 6.3 in a graphical fashion.

![Figure 6.3 Interactions between the elements in MVVM Pattern](image)

### 6.3.1 Data Binding

In order to realize the MVVM pattern, the View, ViewModel and the Model has to be uncoupled. WPF enables a mechanism, called data binding [36], where user interface elements can bind to business logic objects. By using this mechanism, it is possible to realize the same ViewModel in a variety of different Views. Therefore,
The MVVM pattern is one of the design patterns that are highly suitable for WPF application developing.

The data binding mechanism has a wide application area in the software development. This mechanism not only binds class attributes to user interface constructs, but also provides support for data validation or data conversion (casting). More information about the data binding provided with WPF is available in [36].

6.4 Pedigree Query Language Library (PQLLib)

Pedigree query language library (PQLLib) has an extensive pedigree processing operation support [7, 13, 2, 14, 9]. This library has been developed in C#. However, any graphical interface for pedigree processing is not available in this library.

Some of the functionalities provided by PQLLib are as follows: synthetic data generation, NodeCode graph labeling, identity coefficient calculations, generalized kinship calculations, inbreeding calculation, and NodeCode based relationship querying. PQV Tool employs some of these functions that are already provided in the PQLLib.

In this thesis, generalized kinship calculations $\Phi_{abcd}$ and $\Phi_{ab,cd}$ are added to PQLLib. The following NodeCode based relationship calculations are contributed to the library: grandchildren, grandparent, FDR, SDR.

6.5 Third Party Libraries

Some third-party libraries are employed during the implementation of PQV Tool. One of these libraries, called WPF Extensions [37], enables the zoom in/out feature in the visualization. The other third-party library used in PQV Tool is Graph# [38].
6.5.1 Graph Sharp (Graph#)

Graph drawing operations have to be handled by a separate library, since this topic has a very large scope and can be a separate thesis work. A graph drawing library called Graph# is integrated with the PQV Tool. This integration has been possible as Graph# library is based on WPF.

The layout algorithms, edge/vertex overlap algorithms, and the highlighting mechanism are provided by Graph#. In addition, this library enables PQV Tool to create interactive graphs, instead of static images.

6.6 Discussion

In this chapter, the technologies behind the PQV Tool are covered briefly. The database engine and database schema that have been used with PQV Tool implementation is reviewed. Then, the programming framework and design pattern that are employed in PQV Tool are covered. Finally, the libraries that are integrated with PQV Tool are reviewed.
Chapter 7 - Future Work and Conclusion

7.1 Future Work

In this section, future directions for PQV Tool are provided. In each subsection, an aspect of PQV Tool with possible future improvements is described.

7.1.1 PQV Tool improvements

The PQV Tool is a prototype that works on a sample database. The followings are the future directions that can improve the usage of PQV Tool:

- Data import and export support
- Updates on the underlying database schema via the tool
- Support auxiliary relations for pedigree data
- Create/Read/Delete/Update operations over pedigree data

7.1.2 Query Interface

The query interface proposed in this work does not provide queries over multiple relation tables. However, such queries can be meaningful especially when the data related to individuals is stored in multiple tables.

Query interface can be extended visually in order to include more query constructs provided by PQL. Query constructs such as aggregation or associated data would improve the variety of the queries provided by the interface.

7.1.3 Genetic calculations

Current genetic calculation interface allows user to perform computations over selected individuals. These individuals have to be selected manually. However, a batch
calculation interface would help users to run several calculations on their data in an easy way.

7.1.4 Visualization

The visualization interface provided by PQV Tool does not produce pedigrees that are following the aesthetic pedigree drawing rules. Therefore, an improved version of the current visualization interface, which supports these rules, would provide a better understanding on the pedigree data.

7.2 Conclusion

In this thesis, a pedigree data processing tool is proposed. This tool is named pedigree query, visualization, and genetic calculations tool (PQV Tool). The pedigree processing aspects that are investigated are as follows: pedigree querying, genetic calculations and visualization of pedigree data.

A dynamic and visual pedigree query interface is proposed. The proposed query interface allows complex pedigree queries where both pedigree structure and the data associated with pedigree can be used. The query semantics of the interface are described in detailed examples. In addition, the formal description of the queries is provided via PQL and EBNF notation. Finally, the algorithm that evaluates the query interface is described in detail. In order to provide an efficient query evaluation, NodeCode graph encoding method is employed to compute structure based queries.

In PQV Tool, several genetic calculations are provided via the user interface, including inbreeding coefficient, kinship coefficient, generalized kinship coefficients and
condensed identity coefficients. PQV Tool employs the NodeCode based path counting formulas to perform these calculations in an efficient and scalable way.

An interactive data visualization interface is provided with the PQV Tool. This interface allows displaying query results in a more clear way by providing a highlighting mechanism. The data visualization interface also provides interactions between the pedigree drawing and some other components of PQV Tool, such as querying and genetic calculations.

Finally, the technologies behind the PQV Tool are reviewed. These technologies include Windows Presentation Foundation (WPF) Framework, Model-View-View-Model (MVVM) and Graph# library.
Appendix A - User Manual

In this manual, all the aspects related to the usage of the PQV Tool will be covered. Firstly, the compilation and installation will be explained. Then, all the functions provided by the tool and their usage will be covered in the following sections.

PQV Tool works only on 64-bit Windows operating system.

A.1 Source Code and Compilation

The source code of the project is available under Subversion (SVN) server: https://efes.case.edu/svn/Pedigree. This project can be checked out by using any SVN client program.

Once the project is checked out, the solution file must be opened in Microsoft Visual Studio 2010. The Visual Studio solution file, named PQVTool.sln, is under directory ‘trunk\src\PQVTool’. After these steps, project is ready for compilation.

PQV Tool can work with any SQLite database file that complies with the previously defined database schema. If source code is not modified, the database file name has to be ‘sample.db3’. The project comes with its own sample database with the same name.

A.2 Download and Installation

The setup file of the project can be downloaded from the web address: ‘https://filer.case.edu/mxk606/PQVTool/setup.exe’. This installation file will check the current version of the .Net libraries and based on that, it will download all necessary libraries to complete installation.
Once the PQV Tool is installed, user can run it by using the short-cut created on the Desktop. In each run, PQV Tool will communicate with download server to check if an updated version is available. The updated version can be downloaded and installed automatically or user can skip the update step to continue with the current version.

### A.3 Overview

The overview of the query interface system is given in Figure A.1. As seen on the left-hand side of this figure, the query forming template is under ‘Query Builder’ section. On the right-hand side of the same figure, query results can be displayed either in table view or in a graphical view using the corresponding tabs. The genetic calculations are performed by the corresponding buttons in ‘Calculate’ section. Then, the results of the calculations are given in ‘Calculation Result’ section.

![Figure A.1 General view of the PQV Tool](image)
A.4 Query Interface

A.4.1 Query Box

The basic structure of query building interface is a query box which is shown in Figure A.2. This box represents a set of individuals that has to satisfy the conditions defined on the query box. The selected set of individuals appears in the query result only when ‘Output’ check box is checked.

![Query Box in PQV Tool](image)

**Figure A.2 The query box in PQV Tool**

A.4.2 Adding Conditions

There are two types of conditions that can be defined on a query box: attribute-based and structure-based conditions. Attribute-based condition is added to the query box by using ‘Attribute Condition’ button. As shown on the left-hand side of Figure A.3, user can select one or more conditions on the window. Green ‘+’ button adds a new condition and red ‘X’ button removes the condition. System brings the list of attributes from database that can be used as an attribute-based condition.

The second type of conditions is the structure-based condition. Once the corresponding button is clicked, system provides the window shown in the right-hand side of Figure A.3. User has to pick the relationships from combo-box provided on the same window. System provides nine predefined relationships. More than one relationship can be used to form a complex path.
For each attribute-based condition, interface provides the name of the attribute, comparison operators, and an input box. As shown in Figure A.4, interface provides different comparison operators and input boxes based on the data type of the attribute. Any attribute-based condition can be removed by using red ‘X’ button.

Figure A.4 The attribute-based conditions in PQV Tool
A.4.3 Query Execution

Once the user completes her query with the builder, the ‘Run Query’ button shown in Figure A.5 executes the query to produce results. ‘Clear Query’ button is a short-cut for removing all the query boxes and conditions in the interface.

![Query Operations](image)

<table>
<thead>
<tr>
<th>IsOutput</th>
<th>TableName</th>
<th>ColumnName</th>
<th>DataType</th>
<th>AggFunc</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Individual</td>
<td>AdmissionDate</td>
<td>date</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Individual</td>
<td>Age</td>
<td>int</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Individual</td>
<td>FatherID</td>
<td>guid</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Individual</td>
<td>FullName</td>
<td>string</td>
<td>Avg</td>
</tr>
<tr>
<td></td>
<td>Individual</td>
<td>Gender</td>
<td>string</td>
<td>Max</td>
</tr>
<tr>
<td></td>
<td>Individual</td>
<td>HesCancer</td>
<td>string</td>
<td>Min</td>
</tr>
<tr>
<td></td>
<td>Individual</td>
<td>MotherID</td>
<td>guid</td>
<td>Sum</td>
</tr>
<tr>
<td></td>
<td>Individual</td>
<td>PedigreeID</td>
<td>guid</td>
<td>None</td>
</tr>
</tbody>
</table>

Figure A.5 Query Execution buttons in PQV Tool

The ‘Set Output’ button shown in Figure A.5 provides the window shown at the bottom of the same figure. User can select the output columns, which are used in the table view, by checking the boxes given on each row. ‘Deselect All’ button clears the all the check boxes related to the output selection. In addition to this, the query statistics are also obtained by using this window. User has to pick an aggregate function from the combo box provided in the window.

A.4.4 Query Save and Load

PQV Tool enables user to save the frequently used queries to the database. Once user completes the query building, the ‘Save Query’ button shown in Figure A.6 is used
for saving the query. System will ask a comment from user to associate some information with the query to be saved. The saved queries in the database can be retrieved by clicking the ‘Saved Queries’ button. This button will bring up the saved queries management window which is shown in the same figure. The ‘Load’ button on each row will retrieve the query associated with that row. User can also delete any queries by using the red ‘X’ button placed at the beginning of each row.

<table>
<thead>
<tr>
<th>ID</th>
<th>Comment</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Find all individuals who are older than 70 and has cancer</td>
<td>5/6/2012</td>
</tr>
<tr>
<td>2</td>
<td>Find ancestors and descendants of individual with ID 2504</td>
<td>5/6/2012</td>
</tr>
<tr>
<td>3</td>
<td>Find individuals who has at least one ancestor with cancer and one descendant with cancer</td>
<td>5/6/2012</td>
</tr>
<tr>
<td>4</td>
<td>Find individuals who has grand-grand children</td>
<td>5/6/2012</td>
</tr>
<tr>
<td>5</td>
<td>Find individuals who are older than 50 and male</td>
<td>5/6/2012</td>
</tr>
<tr>
<td>6</td>
<td>Find male individuals who are older than 80 and has at least one ancestor and one descendant</td>
<td>5/6/2012</td>
</tr>
</tbody>
</table>

**Figure A.6 Query save and saved queries management window**

**A.4.5 Query Results**

Every query executed via query interface will return a set of individuals. This selected set of individuals can be represented in two forms: table view and graphical view. As seen in Figure A.7, the tabs under ‘Query Results’ section will display the query results in different views, based on the tab used.

The table view tab is a spreadsheet like representation of the results, where each row represents an individual and columns are the corresponding attribute values in the database. This view provides information about the data associated with the individual.
The visualization tab provides a graphical visualization of the pedigree graph. In this graph, the structural information related to the pedigree is represented. In addition to this, the selected individuals are highlighted on this graph. System can visualize one pedigree at a time. Therefore, user has to pick another pedigree ID from the combo box in order to switch between different pedigrees.

![Query Results](image)

**Figure A.7 The query result interfaces provided by the PQV Tool**

### A.5 Visualization

The visualization interface helps user to view the query result in a more understandable way. In addition, there are other functionalities that have to be discussed about the visualization. The zoom in and out pane that is shown on the left-hand side of Figure A.8 allows user to focus on the specific parts of the pedigree graph.

The visual drawing is completely interactive. Hovering over the vertices highlights the edges belonging to parent (red color) and children (blue color). And user can move the vertices if necessary.
Figure A.8 The zoom in & out and highlighting in visualization interface

The graphical view can interact with the table view. User can select individuals by using the check boxes associated with each row on table view. Once the rows are selected, user can open up a context menu by right-clicking in table view. In this context menu, user can click on ‘Show On Graph’. As a result, the selected individuals will be highlighted with red color on the visualization pane, as shown in Figure A.9.
Figure A.9 The interaction between different data views

User can accomplish some basic queries or inbreeding coefficient calculation by right-clicking on a node in the visualization. A context menu will pop-up and several operations will be displayed as shown in Figure A.10. When user clicks on any operation, system will perform that operation for only that particular node.
A.6 Genetic Calculations

PQV Tool provides several genetic calculations. User can calculate inbreeding, kinship (two, three, four or pair) or identity coefficient by using the corresponding buttons shown in Figure A.11.

User can select individuals via two different data view. Table view provides check boxes for each row and drawing view provides check boxes for each node. Based on the number of selected individuals, system associates the kinship request with the correct type of kinship calculation or checks the validity of the calculation. Identity coefficients can be calculated for only two individuals, and inbreeding can be calculated for one individual. User can check ‘Pair’ box to calculate kinship pair value. The order of
the selected individuals will determine the order of the pairs. First two individuals will be the first pair. The other two selected individuals will be the second pair.

The results of the genetic calculations will accumulate on ‘Calculation Results’ section as shown in Figure A.11. User can clear all calculations by using ‘Clear Results’ button shown in the same figure. The ‘Save File’ button will ask for a file name and all calculations will be saved as a comma separated values (.csv) file. The saved file can be opened in spreadsheet editors as shown in Figure A.12.

![Calculation Results](image)

Figure A.12 Saving genetic calculation results to a CSV file

A.7 Settings

PQV Tool provides a menu where user can interact with query interface settings. In the current implementation there exists only one setting related to the query interface. This setting allows the usage of disjunction (or) logic with attribute based conditions. This setting uses disjunction logic instead of conjunction (and) logic among attribute based conditions in every query box. So, queries such as ‘Find all individuals who are older than 70 or has cancer’ become available. However, combination of different logical operators is still not available.
Appendix B- Namespaces and Class Descriptions for PQV Tool

This section only covers the namespaces and class descriptions. The complete documentation of the code base can be obtained by checking out the project from SVN as explained in the Source Code and Compilation section. Once the project is checked out, the help files are under ‘PQVTool\trunk\help’. The help files are available either in pure HTML (view by web-browser) or Compiled HTML Help File (.chm).

### PQVTool Namespace

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="App" /></td>
<td>Interaction logic for App.xaml</td>
</tr>
<tr>
<td><img src="image" alt="MainWindow" /></td>
<td>Interaction logic for MainWindow.xaml</td>
</tr>
<tr>
<td><img src="image" alt="RelayCommand" /></td>
<td>A command whose sole purpose is to relay its functionality to other objects by invoking delegates. The default return value for the CanExecute method is 'true'.</td>
</tr>
</tbody>
</table>

### PQVTool.Converters Namespace

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="BoolToVisibleEnabledConverter" /></td>
<td>Each box can add new relationships until they are expanded therefore we need a conversion to check if a box is extended the expansion removal button will be collapsed</td>
</tr>
<tr>
<td><img src="image" alt="HasParentToVisibilityConverter" /></td>
<td>This converter used in query boxes to check if the box the can</td>
</tr>
<tr>
<td>Converter Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>LinePositionToVisibilityConverter</td>
<td>Based on the position of a line in the list shows add/remove buttons</td>
</tr>
<tr>
<td>MultiBoolVisibilityConverter</td>
<td>This converter used in constraint boxes to check if the box can be visible or not (when it has parent and existence condition is enabled) or not</td>
</tr>
<tr>
<td>NumLinesToVisibilityConverter</td>
<td>This class converts the number of attribute-based conditions to visibility. If number of attribute-conditions is larger than 0 then we show the label.</td>
</tr>
<tr>
<td>NumOfIndividualToVisibilityConverter</td>
<td>This class converts the number of selected individuals to visibility. If number of selected individuals are larger than 0 then we show the text box.</td>
</tr>
<tr>
<td>NumStatisticsToVisibilityConverter</td>
<td>This class converts the number of statistics calculated at the end of query to visibility. If number of statistics calculated is larger than 0 then we show the text box.</td>
</tr>
</tbody>
</table>

### PQVTool.GraphElements Namespace

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphGeneration</td>
<td>This class is responsible of providing graph objects such as vertices and edges given the individuals to the class</td>
</tr>
<tr>
<td>PedEdge</td>
<td>A simple identifiable edge extended from Edge class of Quick Graph.</td>
</tr>
<tr>
<td>PedGraph</td>
<td>Graph class which is derived from Bidirectional Graph class in QuickGraph</td>
</tr>
<tr>
<td>Class</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PedGraphLayout</td>
<td>This is just a derived class from GraphSharp controls for layout</td>
</tr>
<tr>
<td>PedVertex</td>
<td>A simple identifiable vertex for graph representation.</td>
</tr>
</tbody>
</table>

**PQVTool.Model Namespace**

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CalculationResult</td>
<td>Keeps the result of kinship, identity or inbreeding calculation</td>
</tr>
<tr>
<td>QueryTemplate</td>
<td>This class will be needed for the serialization of the Query templates for saving the query templates to the database</td>
</tr>
</tbody>
</table>

**PQVTool.Util Namespace**

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enumerations</td>
<td>This is a utility class needed for public enumerations</td>
</tr>
<tr>
<td>OutputSetColumn</td>
<td>This class represents each element on the set output window as a row</td>
</tr>
<tr>
<td>QueryResultHelper</td>
<td>This class helps the query processor to return both the query (SQL) and the string representation of the query for display purposes</td>
</tr>
<tr>
<td>RelationFindCommandArgs</td>
<td>Driven from EventArgs needed to pass parameters for context menu find relatives on vertices</td>
</tr>
<tr>
<td>StatisticsResult</td>
<td>This class represents each element on the statistics table</td>
</tr>
</tbody>
</table>
This class needed for attaching a property "Close" to close a window by using MVVM

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enumerations.AggregateFunctions</td>
<td>The aggregate functions that can be used by the system</td>
</tr>
<tr>
<td>Enumerations.HeaderTextTypes</td>
<td>An enumeration that is used in add attribute-based and structure-based conditions windows</td>
</tr>
<tr>
<td>Enumerations.RelationTypes</td>
<td>An enumeration that will be used in query box structure-based conditions.</td>
</tr>
</tbody>
</table>

PQVTool.View Namespace

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CommentDialogBox</td>
<td>Interaction logic for DialogBoxWithListPicker.xaml</td>
</tr>
<tr>
<td>IndividualBasedBox</td>
<td>Interaction logic for IndividualBasedBox.xaml</td>
</tr>
<tr>
<td>MultiPickDialogBox</td>
<td>Interaction logic for DialogBoxWithListPicker.xaml</td>
</tr>
<tr>
<td>SavedItemsDialogBox</td>
<td>Interaction logic for SavedItemsDialogBox.xaml</td>
</tr>
<tr>
<td>SetQueryOutputDialogBox</td>
<td>Interaction logic for SetQueryOutputDialogBox.xaml</td>
</tr>
<tr>
<td>Visualization</td>
<td>Interaction logic for Visualization.xaml</td>
</tr>
</tbody>
</table>
## PQVTool.ViewModel Namespace

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComboBoxLineViewModel</td>
<td>This class will represent each line in the dialog boxes to select an element from combobox</td>
</tr>
<tr>
<td>CommentDialogBoxViewModel</td>
<td>This view model represents comment dialog box, where user enters the comments about the query that she would like to save.</td>
</tr>
<tr>
<td>IndividualBasedBoxViewModel</td>
<td>This class represents the view model of a query box in the query interface.</td>
</tr>
<tr>
<td>IndividualBasedLineViewModel</td>
<td>This class represents the view model behind each attribute-based condition on a query box.</td>
</tr>
<tr>
<td>MainViewModel</td>
<td>This class contains properties that the main View can data bind to. Use the <strong>mvvminpc</strong> snippet to add bindable properties to this ViewModel.</td>
</tr>
<tr>
<td>MainWindowViewModel</td>
<td>This is the class where the view model represents the whole interface. The query, visualization and genetic calculations are a part of this view model.</td>
</tr>
<tr>
<td>MultiPickDialogBoxViewModel</td>
<td>This class represents a dialog box where there can be more than one combobox to select items.</td>
</tr>
<tr>
<td>SavedItemsDialogBoxViewModel</td>
<td>This is the view model behind the saved queries management page. By using this page user can load or delete a previously saved query.</td>
</tr>
</tbody>
</table>
query from the database.

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetQueryOutputDialogBoxViewModel</td>
<td>This class is for the view model where the query output columns and query statistics are being set.</td>
</tr>
<tr>
<td>ViewModelBase</td>
<td>Base class for all ViewModel classes in the application. It provides support for property change notifications and has a DisplayName property. This class is abstract.</td>
</tr>
<tr>
<td>ViewModelLocator</td>
<td>This class contains static references to all the view models in the application and provides an entry point for the bindings.</td>
</tr>
</tbody>
</table>
Bibliography


