VCF Parser

Northeastern University

Author: Joshua Conte

Class: DA5020 Collect/Store/Retrieve Data

Instructor: Sara Arunagiri, Ph.D.

Date: August 13, 2017

Contents

1 Abstract

The final project for the class DA5020 Collect/Store/Retrieve Data is a program designed to extract all variant call records from a VCF file using R programing language and storing it in a third normal form (3NF) database. This process will make the data more convenient to work with in downstream analyses. The program will also include examples to understand if a variant is a single nucleotide polymorphism (SNP) or indel (insertion, deletion, or mixed), along with examples of filters based on quality, depth of coverage, and chromosome.

2 Introduction

2.1 What is a VCF File?

VCF stands for Variant Call Format. It is a standardized text file format for representing SNPs, indels, and structural variation calls. The VCF specification is managed by the Global Alliance for Genomics and Health Data Working group file format team[1]. The full format spec can be found in the Samtools/Hts-specs repository[2].

VCF is the primary format used by programs that report variant calls. A variant call is a conclusion that there is a nucleotide difference vs. some reference at a given position in an individual genome or transcriptome[3]. It is a preferred format because while it can be a bit verbose, the VCF format is very explicit about the exact type and sequence of variation as well as the genotypes of multiple samples for this variation[4].

2.2 Structure of a VCF File

A valid VCF file is composed of three main parts: meta-information lines (lines beginning with "##"), header line (line beginning with "#CHROM"), and the variant call records.

The header contains information about the dataset and relevant reference sources (e.g. the organism, genome build version etc.), as well as definitions of all the annotations used to qualify and quantify the properties of the variant calls contained in the VCF file[4]. This information is not necessary to parse the data, only the variant call records are relevant.

In the variant call records section, each line represents a single variant, with various properties of that variant represented in the columns. The columns are CHROM, POS, ID, REF, ALT, QUAL, FILTER, INFO, FORMAT, SAMPLE[4]:

- CHROM: the chromosome
- POS: The genome coordinate of the first base in the variant. Within a chromosome, VCF records are sorted in order of increasing position.
- ID: An optional identifier for the variant. Based on the contig (contig refers to overlapping sequence data (reads)) and position of the call and whether a record exists at this site in a reference database such as dbSNP.
- REF: the reference allele expressed as a sequence of one or more A/C/G/T nucleotides (e.g. "A" or "AAC").
- ALT: the alternate allele expressed as a sequence of one or more A/C/G/T nucleotides (e.g. "A" or "AAC"). If there is more than one alternate alleles, the field should be a comma-separated list of alternate alleles.
  - Note that REF and ALT are always given on the forward strand. For insertions, the ALT allele includes the inserted sequence as well as the base preceding the insertion to know where the insertion is compared to the reference sequence. For deletions, the ALT allele is the base before the deletion.

- QUAL: The Phred-scaled probability that a REF/ALT polymorphism exists at this site given sequencing data. Because the Phred scale is -10 * log(1-p), a value of 10 indicates a 1 in 10 chance of error, while a 100 indicates a 1 in 10^10 chance (see the FAQ article for a detailed explanation). These values can grow very large when a large amount of data is used for variant calling, so QUAL is not often a very useful property for evaluating the quality of a variant call.
  - Not to be confused with the sample-level annotation GQ (explained below).
- FILTER: This field contains the name(s) of any filter(s) that the variant fails to pass, or the value PASS if the variant passed all filters.
- INFO: Various site-level annotations. The annotations contained in the INFO field are represented as tag-value pairs, where the tag and value are separated by an equal sign, i.e. =, and pairs are separated by colons, i.e.; as in this example: MQ=99.00;MQ0=0;QD=17.94. They typically summarize context information from the samples, but can also include information from other sources (e.g. population frequencies from a database resource). They are always defined in the VCF header, so that's an easy way to check what an annotation means. The VCF specification is also a good reference for the annotations[5].

FORMAT and SAMPLE have valuable information. Below, in Table 1, is an example of how this information is currently presented in a VCF file:

Table 1: Example VCF Data

REF	ALT	FORMAT	SAMPLE
G	C	GT:AD:DP:GQ:PL	0/0:0,2:2:40:86,6,0
G	C	GT:AD:DP:GQ:PL	0/1:0,2:2:40:86,6,0
G	C	GT:AD:DP:GQ:PL	1/1:0,2:2:40:86,6,0

Where the common formats are listed below, for a full list refer to the VCF specification[5], the information below was referenced from GATK[4]:

- GT: The genotype of this sample at this site. The GT field indicates the two alleles carried by the sample, encoded by a 0 for the REF allele, 1 for the first ALT allele, 2 for the second ALT allele, etc. When there's a single ALT allele (by far the more common case), GT will be either:
  - 0/0 - the sample is homozygous reference
  - 0/1 - the sample is heterozygous, carrying 1 copy of each of the REF and ALT alleles
  - 1/1 - the sample is homozygous alternate
  - In the three sites shown in the example above, the sample is observed with the allele combinations T/T, G/C, and C/C respectively.
- AD and DP: Allele depth and depth of coverage. These are complementary fields that represent two important ways of thinking about the depth of the data for this sample at this site. AD is the unfiltered allele depth, i.e. the number of reads that support each of the reported alleles. All reads at the position (including reads that did not pass the variant caller's filters) are included in this number, except reads that were considered uninformative. Reads are considered uninformative when they do not provide enough statistical evidence to support one allele over another. DP is the filtered depth, at the sample level. This returns the number of filtered reads that support each of the reported alleles. Only reads that passed the variant caller's filters are included in this number. However, unlike the AD calculation, uninformative reads are included in DP. For example, if REF is 'A' and ALT is 'G' and AD is '6,9' there are 6 A reads and 9 G reads, therefore, DP will be 15 (6+9).
- PL: "Normalized" Phred-scaled likelihoods of the possible genotypes. The PL field will contain three numbers, corresponding to the three possible genotypes (0/0, 0/1, and 1/1). The PL values are "normalized" so that the PL of the most likely genotype (assigned in the GT field) is 0 in the Phred scale. The higher the better for phred scores.
- GQ: Quality of the assigned genotype. The Genotype Quality represents the Phred-scaled confidence that the genotype assignment (GT) is correct, derived from the genotype PLs. Specifically, the GQ is the difference between the PL of the second most likely genotype, and the PL of the most likely

genotype. As noted above, the values of the PLs are normalized so that the most likely PL is always 0, so the GQ ends up being equal to the second smallest PL. If it is low, there is not much confidence in the genotype, i.e. there was not enough evidence to confidently choose one genotype over another.

2.3 The Problem with VCF Files

The problem with VCF files is that they are challenging to understand and the information provided is especially complex. The files have hundreds of lines of metadata which is hard to navigate through and the variant call records are difficult to analyze with data grouped together (i.e. the data is not normalized).

3 Methodology

3.1 Parsing and Filtering the VCF File

The program designed for this project uses R to clean up the data to make it easy to analyze and parse by storing it in a normalized 3NF database. The R packages used to organize the data are: tidyr, dplyr, and reshape2. For additional analysis, this program uses SQL language to make quality filters based on depth of coverage, phred score, and removing unsupported chromosomes. The R packages used to put the data into a database and analyze it are: RSQLite and sqldf.

Utilizing the 3NF database and SQL, the program has three different functions to get SNPs, insertions, deletions, mixed records, or all variant call records with adjustable parameters. These processes for organizing and analyzing the data are described in detail in the coding with R section.

3.2 Acquiring a VCF File

The best way to get a VCF file is to make it. A VCF file can be made from different FASTA and FASTQ files using software such as BWA, Picard, and GATK. This information can be referenced from the document, "PDF workflow of a pipeline using GATK Best Practices"[6].

VCF files can also be downloaded from different websites, for example, the NCBI: ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/pilot_data/release/2010_07/ and from the 1000 genome project: ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/pilot_data/release/2010_07/.

The VCF file for this example has been provided and is titled, "recalibrated_variants_SRR4032408.vcf". It was made using a GATK pipeline from the SRX2023860 exome sequencing of PBCF cell line derived from a metastatic liver, run SRR4032408, using the human genome reference GRCh38.

4 Coding with R

4.1 Making the Database

R studio was configured with the following parameters before beginning the project:

```
# clears the console in RStudio
cat("\014")
```

```
# clears environment
rm(list = ls())

# Set working directory
```

```
setwd("C:/R/DA5020/Final_Project/")

# Load required packages
library(RSQLite)
library(sqldf)
library(tidyr)
library(dplyr)
library(reshape2)
library(knitr) # Used for making tables in this report
```

The program opens a specified VCF file and puts it into a data frame. This command also removes all of the meta-information and header lines, leaving the variant call records:

```
# Some VCF files are really big and take a while to open.  This command checks to
# see if it is already opened, if it is, it does not open it again.
if (!exists("vcf.df")) {
  vcf.df <-
    read.table(
      'recalibrated_variants_SRR4032408.vcf',
      sep = "\t",
      stringsAsFactors = FALSE,
      row.names = NULL,
      na.strings = c(".", "NA")
    )
}
```

Then the program "cleans" the data a little. It begins by removing any variant that has not passed in the FILTER column of the VCF file:

```
# This keeps only the variants that passed the filters under the FILTER column
# Changes anything that is not "PASS" to NA
vcf.df$V7[vcf.df$V7 != "PASS"] <- NA
# Removes all rows with NA in the FILTER column
vcf.df <- vcf.df[!is.na(vcf.df$V7),]
```

Once the data only contains variants that have passed, the program adds a UNIQUE_ID column to properly format a database, this will allow the construction of different tables to ensure the database is normalized to 3NF:

```
# This adds a unique ID to the data
vcf.df$UNIQUE_ID <- seq.int(nrow(vcf.df))
```

4.1.1 The Main Table

Next, the program creates and begins to build the database by adding the necessary information in the first table. This table will contain the information: CHROM, POS, ID, REF, ALT, QUAL, and FILTER. In some cases there could be more than one ID and ALT value seperated by a comma, which is how the data needs to be displayed and does not require normalization:

```
#create database
vcfData <- dbConnect(SQLite(),'vcfData.db')

# Create a table.  The table is created as described above in the introduction to
# database section.  Primary key and specified the data types have been defined in
# the code below.
dbSendQuery(
```

```
  vcfData,
  "CREATE TABLE main_table(
  UNIQUE_ID INT,
  CHROM CHAR,
  POS INT,
  ID CHAR,
  REF CHAR,
  ALT CHAR,
  QUAL CHAR,
  FILTER CHAR,
  PRIMARY KEY (UNIQUE_ID))"
  )
```

```
## <SQLiteResult>
##   SQL  CREATE TABLE main_table(
##   UNIQUE_ID INT,
##   CHROM CHAR,
##   POS INT,
##   ID CHAR,
##   REF CHAR,
##   ALT CHAR,
##   QUAL CHAR,
##   FILTER CHAR,
##   PRIMARY KEY (UNIQUE_ID))
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

```
# The code below works by first listing all column data listed below, then
# processing each column at a time to be written to the database in the required
# format.
UNIQUE_ID <- vcf.df$UNIQUE_ID
CHROM <- vcf.df$V1
POS <- vcf.df$V2
ID <- vcf.df$V3
REF <- vcf.df$V4
ALT <- vcf.df$V5
QUAL <- vcf.df$V6
FILTER <- vcf.df$V7

# The data above is put into a data frame called data
data = cbind.data.frame(
  UNIQUE_ID,
  CHROM,
  POS,
  ID,
  REF,
  ALT,
  QUAL,
  FILTER
)

# This puts the data frame into the table and into the database
dbWriteTable(
  conn = vcfData,
```

```
  name = "main_table",
  data,
  append = T,
  row.names = F
)
```

4.1.2 The Info Table

The INFO column has a lot of information that is divided up with a semicolon. Table 2 shows an example of how this information is stored in a VCF file:

Table 2: INFO Column Example of a VCF File

INFO
AC=2;AF=1.00;AN=2;DB;DP=2;ExcessHet=3.0103

The table in the database splits the INFO column up and keeps the corresponding UNIQUE_ID from the original data frame. This uses the cast function from the reshape package to make INFO column headers with the corresponding value, an example is shown in Table 3.

Table 3: INFO Column Data in Databse Example

UNIQUE_ID	AC	AF	AN	BaseQRankSum	ClippingRankSum	culprit	DB	DP	ExcessHet
1	2	1	2	NA	NA	FS	NA	2	3.0103

Depending on the size of the VCF file, this process could take a couple of minutes:

```
# This will reformat the FORMAT column, to make it 1NF.
infoTable <- function(x) {
  # This function will take UNIQUE_ID and INFO and split up the
  # semicolon separated values and put it with the proper UNIQUE_ID.

  # This puts the required info into a data frame
  df <- data.frame(x$UNIQUE_ID,
                   x$V8)
  # This gives it proper name`s
  names(df) <-
    c("UNIQUE_ID", "INFO")

  # This removes the semicolon separated values in the conditions column
  df2 <- df %>%
    mutate(INFO =
             strsplit(as.character(INFO), ";")) %>%
    unnest(INFO)

  # This uses the tidyr package to separate the information divided by an equal sign
  # in the INFO column and puts it into a INFO_VALUE column.
  df3 <- df2 %>%
    separate(INFO, c("INFO", "INFO_VALUE"), "=")

  return(df3)
```

```
}

# This calls the function with vcf data frame, it could give a warning about too few
# values.  This can be ignored because POSITIVE_TRAIN_SITE and DB are not separated
# by '=' and there will not be a value in the INFO_VALUE table.
info.df <- infoTable(vcf.df)
```

```
## Warning: Too few values at 311022 locations: 4, 18, 25, 33, 40, 50, 58, 69,
## 77, 88, 96, 107, 115, 126, 134, 145, 153, 162, 169, 177, ...
```

```
# This uses the cast function in the reshape package to reshape2 the dataframe to
# make the INFO column headers and the INFO_VALUE column their values.
info2.df<- dcast(info.df, UNIQUE_ID~INFO, value.var = 'INFO_VALUE')

# This puts the data into a table.
dbWriteTable(vcfData, "info_table", info2.df)
```

4.1.3 The Genotype Info and GT Tables

The FORMAT and SAMPLE columns have corresponding information that is separated by a semicolon. Table 4 shows an example of how the FORMAT and SAMPLE columns are presented in a VCF file:

Table 4: Example of FORMAT and SAMPLE Columns in a VCF File

FORMAT	SAMPLE
GT:AD:DP:GQ:PL	1/1:0,2:2:40:86,6,0

Like the info table, this table in the database also uses the cast function from the reshape package. This process merges the information together to make the FORMAT column as headers with the SAMPLE column as values, while keeping the same UNIQUE_ID from the original data frame. An example is shown in Table 5.

Table 5: FORMAT and SAMPLE Columns Merged Example

UNIQUE_ID	GT	AD	DP	GQ	PL
1	1/1	0,2	2	40	86,6,0

This process also includes a step to make a second table, splitting the values of GT, to help calculate if a variant is homozygous or heterozygous.

Depending on the size of the VCF file, this process could take a couple of minutes. Also, there are some VCF files that have more than one sample. For those files, this program will omit the FORMAT and SAMPLE columns by checking the length of the data frame as shown below:

```
# For VCF files with one sample this will store the FORMAT and SAMPLE
# sections in a different table for better analysis
if (length(vcf.df) == 11) {
  # This will reformat the FORMAT column, to make it 1NF.
  formatTable <- function(x) {
    # This function will take UNIQUE_ID and FORMAT and split up the
    # colon separated values and put it with the proper UNIQUE_ID.
```

```
  # This puts the required info into a data frame
  df <- data.frame(x$UNIQUE_ID,
                   x$V9)
  # This gives it proper name`s
  names(df) <-
    c("UNIQUE_ID", "FORMAT")

  # This removes the colon separated values in the conditions column
  df2 <- df %>%
    mutate(FORMAT =
             strsplit(as.character(FORMAT), ":")) %>%
    unnest(FORMAT)

  return(df2)
}

# This calls the function with vcf data frame
format.df <- formatTable(vcf.df)

# This will reformat the SAMPLE column, to make it 1NF.
sampleTable <- function(x) {
  # This function will take UNIQUE_ID and SAMPLE and split up the
  # colon separated values and put it with the proper UNIQUE_ID.

  # This puts the required info into a data frame
  df <- data.frame(x$UNIQUE_ID,
                   x$V10)
  # This gives it proper names
  names(df) <-
    c("UNIQUE_ID", "SAMPLE")

  # This removes the colon separated values in the conditions column
  df2 <- df %>%
    mutate(SAMPLE =
             strsplit(as.character(SAMPLE), ":")) %>%
    unnest(SAMPLE)

  return(df2)
}

# This calls the function with vcf data frame
sample.df <- sampleTable(vcf.df)

# This combines format.df and sample.df to make one data frame.
genotypeInfo.df <- data.frame(format.df$UNIQUE_ID,
                              format.df$FORMAT,
                              sample.df$SAMPLE)
names(genotypeInfo.df) <-
  c("UNIQUE_ID", "FORMAT", "SAMPLE")

# This uses the cast function in the reshape package to reshape2 the dataframe to
# make the FORMAT column headers and the SAMPLE column their values.
genotypeInfo2.df<- dcast(genotypeInfo.df, UNIQUE_ID~FORMAT, value.var = 'SAMPLE')
```

```
  # This reformats GT so I can calculate if a variant is homozygous or heterozygous
  gt.df <- genotypeInfo2.df %>%
    separate(GT, c("GT1", "GT2"), "/")

  gt.df <- data.frame(gt.df$UNIQUE_ID, gt.df$GT1, gt.df$GT2)
  names(gt.df) <-
    c("UNIQUE_ID", "GT1", "GT2")

  # This puts the variant call record data into a table.
  dbWriteTable(vcfData, "genotypeInfo_table", genotypeInfo2.df)

  # This puts the gt data into a table.
  dbWriteTable(vcfData, "gt_table", gt.df)
}
```

4.1.4 Confirmation of the Tables

The next set of code confirms what tables are included in the database for reference:

```
# The database is complete
# confirm tables
dbListTables(vcfData)
```

```
## [1] "genotypeInfo_table" "gt_table"           "info_table"
## [4] "main_table"
```

4.2 Analyzing the Data

Before analyzing the data, the environment is cleared and the database is 'reloaded' to remove all unnecessary objects:

```
# clears environment
rm(list = ls())

# Set working directory
setwd("C:/R/DA5020/Final_Project/")

#Load database
vcfData <- dbConnect(SQLite(),'vcfData.db')
```

4.2.1 Understanding the Data: Decoding VCF Entries for SNPs and Indels

This section explains what to look for in the data to understand if the variant is a SNP or indel (insertion, deletion, or mixed). These examples begin by showing an example of each type of variant and how to filter the data to find them.

4.2.1.1 SNPs

To make new cells, an existing cell divides in two. But first it copies its DNA so the new cells will each have a complete set of genetic instructions. Cells sometimes make mistakes during the copying process - kind of like typos. These typos lead to variations in the DNA sequence at particular locations, called SNPs (pronounced "snips")[7].

SNPs can generate biological variation between people by causing differences in the recipes for proteins that are written in genes. Those differences can in turn influence a variety of traits such as appearance, disease susceptibility or response to drugs. While some SNPs lead to differences in health or physical appearance, most SNPs seem to lead to no observable differences between people at all[7].

Table 6 shows an example of a SNP:

Table 6: Example of a SNP

UNIQUE_ID	CHROM	POS	ID	REF	ALT	QUAL	FILTER
5	chr1	944296	rs6605067	G	A	176.77	PASS

This is a SNP since its only single base substitution and there are only two alleles where G base (reference) is an A (alternate) in some individuals.

The following search can be used to find all SNPs:

```
# This uses basic SQL syntax; everything is selected with '*'. Then the length
# between REF and ALT is compared and if they are equal, then it is a SNP. This
# also removed ALT data with a comma (since that is considered mixed not an SNP)
allSNP <- dbGetQuery(
  vcfData,
  "SELECT *
  FROM main_table
  WHERE LENGTH(REF) = LENGTH(ALT)
  AND ALT NOT LIKE '%,%'
  "
)

# The summary gives information about the data that is stored in the dataframe.
summary(allSNP)
```

```
##    UNIQUE_ID         CHROM                POS
##  Min.   :     1   Length:158832      Min.   :       73
##  1st Qu.: 44855   Class :character   1st Qu.: 32250350
##  Median : 89828   Mode  :character   Median : 64084452
##  Mean   : 89645                      Mean   : 76916554
##  3rd Qu.:134547                      3rd Qu.:112221384
##  Max.   :179086                      Max.   :248845034
##       ID                 REF                ALT
##  Length:158832      Length:158832      Length:158832
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##
##
##
##      QUAL              FILTER
##  Length:158832      Length:158832
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```
# head prints the first six lines.
head(allSNP)
```

```
##   UNIQUE_ID CHROM    POS         ID REF ALT   QUAL FILTER
## 1         1  chr1 895427  rs6422669   G   C  58.74   PASS
## 2         2  chr1 925308 rs60837925   G   A  51.74   PASS
## 3         3  chr1 930939  rs9988021   G   A  185.9   PASS
## 4         4  chr1 941119  rs4372192   A   G  33.77   PASS
## 5         5  chr1 944296  rs6605067   G   A 176.77   PASS
## 6         6  chr1 944307     rs2839   T   C 152.77   PASS
```

4.2.1.2 Indels

As noted in the SNP section, cells sometimes make mistakes during the copying process, one such mistake is a SNP and another is an indel. An indel is a term term for an insertion or deletion (or sometimes both) of bases in the genome. Like SNPs, indels can generate biological variation between people by causing differences in the recipes for proteins that are written in genes[8].

4.2.1.2.1 Indels: Insertions

Table 7 shows an example of a insertion:

Table 7: Example of an Insertion

UNIQUE_ID	CHROM	POS	ID	REF	ALT	QUAL	FILTER
198	chr1	1758687	rs79655042	A	AAC	361.74	PASS

This is an insertion since the reference base A is being replaced by A (the reference base) plus two insertion bases AC (alternate).

The following search can be used to find all insertions:

```
# This uses basic SQL syntax; everything is selected with '*'. Then the length
# between REF and ALT is compared and if ALT is greater, then the variant is an
# insertion.  This also removed ALT data with a comma (since that is considered
# mixed not an insertion).
allInsertions <- dbGetQuery(
  vcfData,
  "SELECT *
  FROM main_table
  WHERE LENGTH(REF) < LENGTH(ALT)
  AND ALT NOT LIKE '%,%'
  "
)

# The summary gives information about the data that is stored in the dataframe.
summary(allInsertions)
```

```
##    UNIQUE_ID         CHROM                POS
##  Min.   :    55   Length:9023        Min.   :      252
##  1st Qu.: 43570   Class :character   1st Qu.: 34382682
##  Median : 85890   Mode  :character   Median : 66354802
##  Mean   : 88206                      Mean   : 79013485
##  3rd Qu.:131492                      3rd Qu.:113704704
```

```
##  Max.   :179087                      Max.   :248684041
##       ID                 REF                 ALT
##  Length:9023        Length:9023        Length:9023
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##
##
##
##      QUAL              FILTER
##  Length:9023        Length:9023
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```
# head prints the first six lines.
head(allInsertions)
```

```
##   UNIQUE_ID CHROM     POS          ID REF    ALT    QUAL FILTER
## 1        55  chr1 1013466   rs3841266   T     TA  533.73   PASS
## 2        90  chr1 1168310 rs139946018   T   TCA   98.25   PASS
## 3       139  chr1 1341593 rs145370195   G GACAC 2151.73   PASS
## 4       193  chr1 1752908 rs150880809   C  CCCT 1666.73   PASS
## 5       198  chr1 1758687  rs79655042   A   AAC  361.74   PASS
## 6       222  chr1 1922176   rs3039777   T TCTGA  187.87   PASS
```

4.2.1.2.2 Indels: Deletions

Table 8 shows an example of a deletion variant:

Table 8: Example of a Deletion

UNIQUE_ID	CHROM	POS	ID	REF	ALT	QUAL	FILTER
8	chr1	945259	rs35916504	TC	T	97.76	PASS

This is a deletion of one reference base since the reference allele TC is being replaced by just the T (the reference base).

The following search can be used to find all deletions:

```
# This uses basic SQL syntax; everything is selected with '*'. Then the length
# between REF and ALT is compared and if REF is greater, then it is a deletion. This
# also removed ALT data with a comma (since that is considered mixed not an SNP)
allDeletions <- dbGetQuery(
  vcfData,
  "SELECT *
  FROM main_table
  WHERE LENGTH(REF) > LENGTH(ALT)
  AND ALT NOT LIKE '%,%'
  "
)

# The summary gives information about the data that is stored in the dataframe.
```

```
summary(allDeletions)
```

```
##    UNIQUE_ID         CHROM                POS
##  Min.   :     8   Length:11156       Min.   :      268
##  1st Qu.: 44392   Class :character   1st Qu.: 35400527
##  Median : 88408   Mode  :character   Median : 67264604
##  Mean   : 89244                      Mean   : 79037132
##  3rd Qu.:133334                      3rd Qu.:113163962
##  Max.   :179076                      Max.   :248273945
##       ID                 REF                ALT
##  Length:11156       Length:11156       Length:11156
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##
##
##
##      QUAL              FILTER
##  Length:11156       Length:11156
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```
# head prints the first six lines.
head(allDeletions)
```

```
##   UNIQUE_ID CHROM     POS          ID                                REF ALT
## 1         8  chr1  945259  rs35916504                                 TC   T
## 2        31  chr1  965666  rs34612277                                ACC   A
## 3        88  chr1 1104818        <NA>                                GAA   G
## 4        96  chr1 1197935 rs142789284                                AAC   A
## 5       102  chr1 1217383  rs71768002                             AGCCGCG   A
## 6       107  chr1 1229060        <NA> CGCCTTCTCCAGACCACACGTGGCACT   C
##     QUAL FILTER
## 1  97.76   PASS
## 2 103.73   PASS
## 3  35.71   PASS
## 4 124.73   PASS
## 5  79.73   PASS
## 6  39.73   PASS
```

4.2.1.2.3 Indels: Mixed VCF Record

Table 9 shows an example of a mixed record variant:

Table 9: Example of a Mixed Record

UNIQUE_ID	CHROM	POS	ID	REF	ALT	QUAL	FILTER
42276	chr3	186577628	rs60278421	AT	A,ATT	344.73	PASS

This is a mixed type record containing a one base insertion and a one base deletion.

The following search can be used to find all mixed records:

```
# This uses basic SQL syntax; everything is selected with '*'. Then the ALT column
# is analyzed for a presence of a comma and if there is a comma then it is a mixed
# record.
allMixed <- dbGetQuery(
  vcfData,
  "SELECT *
  FROM main_table
  WHERE ALT like '%,%'
  "
)

# The summary gives information about the data that is stored in the dataframe.
summary(allMixed)
```

```
##    UNIQUE_ID         CHROM                POS
##  Min.   :  1059   Length:76          Min.   :  2565133
##  1st Qu.: 32353   Class :character   1st Qu.: 34276585
##  Median : 77554   Mode  :character   Median : 50940672
##  Mean   : 82394                      Mean   : 81440479
##  3rd Qu.:122047                      3rd Qu.:119405352
##  Max.   :174737                      Max.   :246528724
##       ID                 REF                ALT
##  Length:76          Length:76          Length:76
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##
##
##
##      QUAL              FILTER
##  Length:76          Length:76
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```
# head prints the first six lines.
head(allMixed)
```

```
##   UNIQUE_ID CHROM       POS          ID     REF       ALT   QUAL FILTER
## 1      1059  chr1  10457365   rs2781227       T       A,G 716.77   PASS
## 2      4748  chr1  46936815        <NA> TTGTGTG     T,TTG 677.73   PASS
## 3      4890  chr1  50404638 rs143043731     ATT       A,AT 329.73   PASS
## 4      5236  chr1  54614962   rs1655525       G GTGTGCA,A 310.77   PASS
## 5      7969  chr1 103031249  rs36076089      GA     G,GAA 159.73   PASS
## 6     10213  chr1 156736767      rs8658       A       C,G 743.77   PASS
```

4.3 Filtering the Data

This section shows how to combine tables to filter with additional parameters.

4.3.1 Filtering for Depth of Coverage

It is important to have a good depth of coverage. Depth of coverage describes the number of times that a given nucleotide in the genome has been read in an experiment. In most next generation sequencing (NGS) protocols, the DNA is fragmented into short sections of a few hundred base pairs. These are individually read and then bioinformatically overlapped or "tiled" to generate longer contiguous sequences making up the meaningful end data. At first, it might seem like it's only necessary to read each nucleotide position once in order to do this; however, for tiling to work, there need to be several individual reads with significant overlaps in order to line them up with any confidence. These overlap regions therefore of necessity have each nucleotide read more than once[9].

According to a medical and molecular geneticist at GeneDX, Dr. Maria Jose Guillen Sacoto, a good depth of coverage for clinical applications is greater than or equal to 10[10]. Below is a filter removing variants with a depth of coverage less than 10. This information also includes the GT, AD, DP, PL, and GQ data from the genotype table:

```
# This uses basic SQL syntax; everything is selected with '*' from the main table.
# The main table also inculdes data from the genotype table, by using join function.
# Then the data is filtered based off of the depth of coverage scores greater than
# or equal to 10 in the DP column of the genotype table. The end result is the main
# table filtered from the results of the genotype table. This can be achieved
# because the tables share the same unique ID. The DP and GQ information was stored
# as a character, for better analysis, it was changed to integer using the cast
# command, as shown below.
filterDP <-
  dbGetQuery(
    vcfData,
    "SELECT m.*, g.AD, CAST(g.DP as INT) AS DP, CAST(g.GQ as INT) AS GQ, g.GT, g.PL
    FROM main_table m
    JOIN genotypeInfo_table g ON m.UNIQUE_ID = g.UNIQUE_ID
    WHERE m.UNIQUE_ID in (
    SELECT UNIQUE_ID
    FROM genotypeInfo_table
    WHERE CAST(DP AS INT)  >= 10)
    "
  )

# The summary gives information about the data that is stored in the dataframe.
summary(filterDP)
```

```
##    UNIQUE_ID         CHROM                POS
##  Min.   :     5   Length:58394       Min.   :      189
##  1st Qu.: 41980   Class :character   1st Qu.: 33327595
##  Median : 91169   Mode  :character   Median : 63725982
##  Mean   : 88946                      Mean   : 77967995
##  3rd Qu.:134526                      3rd Qu.:113920726
##  Max.   :179050                      Max.   :248810937
##       ID                 REF                ALT
##  Length:58394       Length:58394       Length:58394
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##
##
##
##      QUAL              FILTER               AD                  DP
```

```
##  Length:58394       Length:58394       Length:58394       Min.   : 10.00
##  Class :character   Class :character   Class :character   1st Qu.: 15.00
##  Mode  :character   Mode  :character   Mode  :character   Median : 24.00
##                                                           Mean   : 33.11
##                                                           3rd Qu.: 41.00
##                                                           Max.   :461.00
##        GQ             GT                 PL
##  Min.   : 0.00   Length:58394       Length:58394
##  1st Qu.:54.00   Class :character   Class :character
##  Median :99.00   Mode  :character   Mode  :character
##  Mean   :79.19
##  3rd Qu.:99.00
##  Max.   :99.00
```

```
# head prints the first six lines.
head(filterDP)
```

```
##   UNIQUE_ID CHROM    POS        ID REF ALT   QUAL FILTER    AD DP GQ  GT
## 1         5  chr1 944296 rs6605067   G   A 176.77   PASS   4,6 10 99 0/1
## 2         7  chr1 944858 rs3748592   A   G 313.77   PASS 15,10 25 99 0/1
## 3        10  chr1 948245 rs4970378   A   G 448.77   PASS  0,12 12 36 1/1
## 4        13  chr1 952180 rs3748595   A   C 315.77   PASS 18,10 28 99 0/1
## 5        15  chr1 953259 rs3748596   T   C 661.77   PASS 18,20 38 99 0/1
## 6        16  chr1 953279 rs3748597   T   C 708.77   PASS 21,21 42 99 0/1
##          PL
## 1 205,0,124
## 2 342,0,515
## 3  477,36,0
## 4 344,0,657
## 5 690,0,605
## 6 737,0,669
```

4.3.2 Filter for Quality

Building off of the previous example, it is also important to filter based off of quality. There are some different quality metrics in a VCF file, the two main ones are QUAL (in the main table) and GQ (in the genotype table).

- QUAL refers to the confidence that there is some kind of variation at a given site. The variation may be present in one or more samples.
- GQ refers to the confidence that the genotype we assigned to a particular sample is correct. It is simply the second lowest PL, because it is the difference between the second lowest PL and the lowest PL (always 0).

QUAL is redundant if there is only one sample, especially if it has a good GQ. Therefore, in this case, it is better to filter based off of GQ for quality.

According to Dr. Maria Jose Guillen Sacoto, a good quality score for clinical applications is greater than or equal to 50[10]. Below is a filter removing variants with a quality score less than 50:

```
# This uses basic SQL syntax; everything is selected with '*' from the main table.
# The main table also inculdes data from the genotype table, by using join function.
# Then the data is filtered based off of the depth of coverage scores greater than
# or equal to 10 in the DP column of the genotype table. The same is applied to a
# quality score greater than or equal to 40 in the GQ column. The end result is the
# main table filtered from the results of the genotype table. This can be achieved
```

```
# because the tables share the same unique ID. The DP and GQ information was stored
# as a character, for better analysis, it was changed to integer using the cast
# command, as shown below.
filterDPQuality <-
  dbGetQuery(
    vcfData,
    "SELECT m.*, g.AD, CAST(g.DP as INT) AS DP, CAST(g.GQ as INT) AS GQ, g.GT, g.PL
    FROM main_table m
    JOIN genotypeInfo_table g ON m.UNIQUE_ID = g.UNIQUE_ID
    WHERE m.UNIQUE_ID in (
    SELECT UNIQUE_ID
    FROM genotypeInfo_table
    WHERE CAST(DP AS INT)  >= 10
    AND CAST(GQ AS INT)  >= 50)
    "
  )

# The summary gives information about the data that is stored in the dataframe.
summary(filterDPQuality)
```

```
##    UNIQUE_ID         CHROM                POS
##  Min.   :     5   Length:46162       Min.   :      263
##  1st Qu.: 39344   Class :character   1st Qu.: 34750691
##  Median : 89980   Mode  :character   Median : 63738244
##  Mean   : 87869                      Mean   : 78473065
##  3rd Qu.:134235                      3rd Qu.:113456288
##  Max.   :179029                      Max.   :248810937
##       ID                 REF                ALT
##  Length:46162       Length:46162       Length:46162
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##
##
##
##      QUAL              FILTER               AD                  DP
##  Length:46162       Length:46162       Length:46162       Min.   : 10.00
##  Class :character   Class :character   Class :character   1st Qu.: 20.00
##  Mode  :character   Mode  :character   Mode  :character   Median : 30.00
##                                                           Mean   : 38.34
##                                                           3rd Qu.: 47.00
##                                                           Max.   :461.00
##        GQ             GT                 PL
##  Min.   :50.00   Length:46162       Length:46162
##  1st Qu.:86.00   Class :character   Class :character
##  Median :99.00   Mode  :character   Mode  :character
##  Mean   :90.24
##  3rd Qu.:99.00
##  Max.   :99.00
```

```
# head prints the first six lines.
head(filterDPQuality)
```

```
##   UNIQUE_ID CHROM    POS         ID REF ALT   QUAL FILTER    AD DP GQ  GT
## 1         5  chr1 944296  rs6605067   G   A 176.77   PASS   4,6 10 99 0/1
## 2         7  chr1 944858  rs3748592   A   G 313.77   PASS 15,10 25 99 0/1
```

```
## 3        13  chr1 952180  rs3748595   A   C 315.77   PASS 18,10 28 99 0/1
## 4        15  chr1 953259  rs3748596   T   C 661.77   PASS 18,20 38 99 0/1
## 5        16  chr1 953279  rs3748597   T   C 708.77   PASS 21,21 42 99 0/1
## 6        17  chr1 953778 rs13303056   G   C 160.77   PASS   7,5 12 99 0/1
##          PL
## 1 205,0,124
## 2 342,0,515
## 3 344,0,657
## 4 690,0,605
## 5 737,0,669
## 6 189,0,279
```

4.3.3 Removing Unsupported and Unmapped Chromosomes

VCF files include unsupported and unmapped chromosomes such as chrM and chrUn. These can be removed for analysis as shown below:

```
# This uses basic SQL syntax; everything is selected with '*' from the main table.
# The main table also inculdes data from the genotype table, by using join function.
# Then the data is filtered based off of the depth of coverage scores greater than
# or equal to 10 in the DP column of the genotype table. The same is applied to a
# quality score greater than or equal to 40 in the GQ column. Then syntax is added
# to remove unsupported chromosomes. The end result is the main table filtered from
# the results of the genotype table. This can be achieved because the tables share
# the same unique ID. The DP and GQ information was stored as a character, for
# better analysis, it was changed to integer using the cast command, as shown below.

filtDPQualChr <-
  dbGetQuery(
    vcfData,
    "SELECT m.*, g.AD, CAST(g.DP as INT) AS DP, CAST(g.GQ as INT) AS GQ, g.GT, g.PL
    FROM main_table m
    JOIN genotypeInfo_table g ON m.UNIQUE_ID = g.UNIQUE_ID
    WHERE m.UNIQUE_ID in (
    SELECT UNIQUE_ID
    FROM genotypeInfo_table
    WHERE CAST(DP AS INT)  >= 10
    AND CAST(GQ AS INT)  >= 50)
    AND CHROM NOT IN ('chrM')
    AND CHROM NOT LIKE ('%chrU%')
    "
  )

# The summary gives information about the data that is stored in the dataframe.
summary(filtDPQualChr)
```

```
##    UNIQUE_ID         CHROM                POS
##  Min.   :     5   Length:46151       Min.   :    46405
##  1st Qu.: 39330   Class :character   1st Qu.: 34753346
##  Median : 89939   Mode  :character   Median : 63742630
##  Mean   : 87848                      Mean   : 78491767
##  3rd Qu.:134220                      3rd Qu.:113480132
##  Max.   :178934                      Max.   :248810937
##       ID                REF                ALT
```

```
##  Length:46151       Length:46151       Length:46151
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##
##
##
##      QUAL              FILTER               AD                  DP
##  Length:46151       Length:46151       Length:46151       Min.   : 10.00
##  Class :character   Class :character   Class :character   1st Qu.: 20.00
##  Mode  :character   Mode  :character   Mode  :character   Median : 30.00
##                                                           Mean   : 38.34
##                                                           3rd Qu.: 47.00
##                                                           Max.   :461.00
##        GQ             GT                 PL
##  Min.   :50.00   Length:46151       Length:46151
##  1st Qu.:87.00   Class :character   Class :character
##  Median :99.00   Mode  :character   Mode  :character
##  Mean   :90.24
##  3rd Qu.:99.00
##  Max.   :99.00
```

```
# head prints the first six lines.
head(filtDPQualChr)
```

```
##   UNIQUE_ID CHROM    POS         ID REF ALT   QUAL FILTER    AD DP GQ  GT
## 1         5  chr1 944296  rs6605067   G   A 176.77   PASS   4,6 10 99 0/1
## 2         7  chr1 944858  rs3748592   A   G 313.77   PASS 15,10 25 99 0/1
## 3        13  chr1 952180  rs3748595   A   C 315.77   PASS 18,10 28 99 0/1
## 4        15  chr1 953259  rs3748596   T   C 661.77   PASS 18,20 38 99 0/1
## 5        16  chr1 953279  rs3748597   T   C 708.77   PASS 21,21 42 99 0/1
## 6        17  chr1 953778 rs13303056   G   C 160.77   PASS   7,5 12 99 0/1
##          PL
## 1 205,0,124
## 2 342,0,515
## 3 344,0,657
## 4 690,0,605
## 5 737,0,669
## 6 189,0,279
```

4.4 Functions for Productive Analysis

Before beginning the final section, the environment is cleared and the database is 'reloaded' to remove all unnecessary objects:

```
# clears environment
rm(list = ls())

# Set working directory
setwd("C:/R/DA5020/Final_Project/")

#Load database
vcfData <- dbConnect(SQLite(),'vcfData.db')
```

4.4.1 Search by Type

This function takes the filters from the previous section and puts them together so the user can enter SNP, insertion, deletion, mixed, or ALL and get the results. This also makes depth of coverage and quality adjustable:

```
filterByType <- function(x,y,z) {
  # For SNPs
  if (x == "SNP") {
    # This takes the filter string and the SNP string
    fn$dbGetQuery(
      vcfData,
      "SELECT m.*, g.AD, CAST(g.DP as INT) AS DP, CAST(g.GQ as INT) AS GQ, g.GT, g.PL
      FROM main_table m
      JOIN genotypeInfo_table g ON m.UNIQUE_ID = g.UNIQUE_ID
      WHERE m.UNIQUE_ID in (
      SELECT UNIQUE_ID
      FROM genotypeInfo_table
      WHERE CAST(DP AS INT)  >= $y
      AND CAST(GQ AS INT)  >= $z)
      AND CHROM NOT IN ('chrM')
      AND CHROM NOT LIKE ('%chrU%')
      AND LENGTH(REF) = LENGTH(ALT)
      AND ALT NOT LIKE '%,%'"
    )
  }

  # For insertions
  else if (x == "insertion") {
    # This takes the filter string with the insertion string
    fn$dbGetQuery(
      vcfData,
      "SELECT m.*, g.AD, CAST(g.DP as INT) AS DP, CAST(g.GQ as INT) AS GQ, g.GT, g.PL
      FROM main_table m
      JOIN genotypeInfo_table g ON m.UNIQUE_ID = g.UNIQUE_ID
      WHERE m.UNIQUE_ID in (
      SELECT UNIQUE_ID
      FROM genotypeInfo_table
      WHERE CAST(DP AS INT)  >= $y
      AND CAST(GQ AS INT)  >= $z)
      AND CHROM NOT IN ('chrM')
      AND CHROM NOT LIKE ('%chrU%')
      AND LENGTH(REF) < LENGTH(ALT)
      AND ALT NOT LIKE '%,%'"
    )
  }

  # For deletions
  else if (x == "deletion") {
    # This takes the filter string with the deletion string
    fn$dbGetQuery(
      vcfData,
      "SELECT m.*, g.AD, CAST(g.DP as INT) AS DP, CAST(g.GQ as INT) AS GQ, g.GT, g.PL
      FROM main_table m
```

```
    JOIN genotypeInfo_table g ON m.UNIQUE_ID = g.UNIQUE_ID
    WHERE m.UNIQUE_ID in (
    SELECT UNIQUE_ID
    FROM genotypeInfo_table
    WHERE CAST(DP AS INT)  >= $y
    AND CAST(GQ AS INT)  >= $z)
    AND CHROM NOT IN ('chrM')
    AND CHROM NOT LIKE ('%chrU%')
    AND LENGTH(REF) > LENGTH(ALT)
    AND ALT NOT LIKE '%,%'"
  )
}

# For mixed records
else if (x == "mixed") {
  # This takes the filter string with the mixed string
  fn$dbGetQuery(
    vcfData,
    "SELECT m.*, g.AD, CAST(g.DP as INT) AS DP, CAST(g.GQ as INT) AS GQ, g.GT, g.PL
    FROM main_table m
    JOIN genotypeInfo_table g ON m.UNIQUE_ID = g.UNIQUE_ID
    WHERE m.UNIQUE_ID in (
    SELECT UNIQUE_ID
    FROM genotypeInfo_table
    WHERE CAST(DP AS INT)  >= $y
    AND CAST(GQ AS INT)  >= $z)
    AND CHROM NOT IN ('chrM')
    AND CHROM NOT LIKE ('%chrU%')
    AND ALT like '%,%'"
  )
}

# For mixed records
else if (x == "ALL") {
  # This takes the filter string with the mixed string
  fn$dbGetQuery(
    vcfData,
    "SELECT m.*, g.AD, CAST(g.DP as INT) AS DP, CAST(g.GQ as INT) AS GQ, g.GT, g.PL
    FROM main_table m
    JOIN genotypeInfo_table g ON m.UNIQUE_ID = g.UNIQUE_ID
    WHERE m.UNIQUE_ID in (
    SELECT UNIQUE_ID
    FROM genotypeInfo_table
    WHERE CAST(DP AS INT)  >= $y
    AND CAST(GQ AS INT)  >= $z)
    AND CHROM NOT IN ('chrM')
    AND CHROM NOT LIKE ('%chrU%')"
  )
}

else
  (print("Please use argument 'SNP', 'insertion', 'deletion', 'mixed', or
         'ALL'."))
```

```
  }

# This calls the function with argument in quotes.  Use argument SNP, insertion,
# deletion, mixed, or ALL in quotes followed by comma with a number for depth of
# coverage followed by a comman and a number for quality (up to 99) as shown below:
type <- filterByType("ALL",10,50)

# The summary gives information about the data that is stored in the dataframe.
summary(type)
```

```
##    UNIQUE_ID         CHROM                POS
##  Min.   :     5   Length:46151       Min.   :    46405
##  1st Qu.: 39330   Class :character   1st Qu.: 34753346
##  Median : 89939   Mode  :character   Median : 63742630
##  Mean   : 87848                      Mean   : 78491767
##  3rd Qu.:134220                      3rd Qu.:113480132
##  Max.   :178934                      Max.   :248810937
##       ID                 REF                ALT
##  Length:46151       Length:46151       Length:46151
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##
##
##
##      QUAL              FILTER               AD                  DP
##  Length:46151       Length:46151       Length:46151       Min.   : 10.00
##  Class :character   Class :character   Class :character   1st Qu.: 20.00
##  Mode  :character   Mode  :character   Mode  :character   Median : 30.00
##                                                           Mean   : 38.34
##                                                           3rd Qu.: 47.00
##                                                           Max.   :461.00
##        GQ             GT                 PL
##  Min.   :50.00   Length:46151       Length:46151
##  1st Qu.:87.00   Class :character   Class :character
##  Median :99.00   Mode  :character   Mode  :character
##  Mean   :90.24
##  3rd Qu.:99.00
##  Max.   :99.00
```

```
# This prints the first 3 lines.
head(type, 3)
```

```
##   UNIQUE_ID CHROM    POS        ID REF ALT   QUAL FILTER    AD DP GQ  GT
## 1         5  chr1 944296 rs6605067   G   A 176.77   PASS   4,6 10 99 0/1
## 2         7  chr1 944858 rs3748592   A   G 313.77   PASS 15,10 25 99 0/1
## 3        13  chr1 952180 rs3748595   A   C 315.77   PASS 18,10 28 99 0/1
##          PL
## 1 205,0,124
## 2 342,0,515
## 3 344,0,657
```

```
# This prints the last 3 lines.
tail(type, 3)
```

```
##       UNIQUE_ID                  CHROM    POS       ID REF ALT    QUAL
```

```
## 46149    178845 chr17_GL000205v2_random  57239     <NA>   G   T  179.77
## 46150    178929    chr22_KI270879v1_alt 270497   rs4630   G   A  680.77
## 46151    178934    chr22_KI270879v1_alt 278129 rs140313   C   T 2529.77
##       FILTER   AD DP GQ  GT         PL
## 46149   PASS 13,7 20 99 0/1  208,0,489
## 46150   PASS 0,18 18 54 1/1   709,54,0
## 46151   PASS 0,64 64 99 1/1 2558,193,0
```

4.4.2 Search by Chromosome

It is common practice to analyze SNPs and other records by chromosome. To make the search easier, this program includes a function that will find all SNPs, insertions, deletions, mixed records, or all variant call records by chromosome using the filters from the previous section. This also makes depth of coverage and quality adjustable:

```
filterByChrom <- function(x, y, a, b) {
  # For SNPs
  if (x == "SNP") {
    # This takes the filter string and the SNP string and searches by specified
    # chromosome
    fn$dbGetQuery(
      vcfData,
      "SELECT m.*, g.AD, CAST(g.DP as INT) AS DP, CAST(g.GQ as INT) AS GQ, g.GT, g.PL
      FROM main_table m
      JOIN genotypeInfo_table g ON m.UNIQUE_ID = g.UNIQUE_ID
      WHERE m.UNIQUE_ID in (
      SELECT UNIQUE_ID
      FROM genotypeInfo_table
      WHERE CAST(DP AS INT)  >= $a
      AND CAST(GQ AS INT)  >= $b)
      AND LENGTH(REF) = LENGTH(ALT)
      AND ALT NOT LIKE '%,%'
      AND CHROM = ('$y')"
    )
  }

  # For insertions
  else if (x == "insertion") {
    # This takes the filter string with the insertion string and searches by
    # specified chromosome
    fn$dbGetQuery(
      vcfData,
      "SELECT m.*, g.AD, CAST(g.DP as INT) AS DP, CAST(g.GQ as INT) AS GQ, g.GT, g.PL
      FROM main_table m
      JOIN genotypeInfo_table g ON m.UNIQUE_ID = g.UNIQUE_ID
      WHERE m.UNIQUE_ID in (
      SELECT UNIQUE_ID
      FROM genotypeInfo_table
      WHERE CAST(DP AS INT)  >= $a
      AND CAST(GQ AS INT)  >= $b)
      AND LENGTH(REF) < LENGTH(ALT)
      AND ALT NOT LIKE '%,%'
      AND CHROM = ('$y')"
    )
```

```
}

# For deletions
else if (x == "deletion") {
  # This takes the filter string with the deletion string and searches by
  # specified chromosome
  fn$dbGetQuery(
    vcfData,
    "SELECT m.*, g.AD, CAST(g.DP as INT) AS DP, CAST(g.GQ as INT) AS GQ, g.GT, g.PL
    FROM main_table m
    JOIN genotypeInfo_table g ON m.UNIQUE_ID = g.UNIQUE_ID
    WHERE m.UNIQUE_ID in (
    SELECT UNIQUE_ID
    FROM genotypeInfo_table
    WHERE CAST(DP AS INT)  >= $a
    AND CAST(GQ AS INT)  >= $b)
    AND LENGTH(REF) > LENGTH(ALT)
    AND ALT NOT LIKE '%,%'
    AND CHROM = ('$y')"
  )
}

# For mixed records
else if (x == "mixed") {
  # This takes the filter string with the mixed string and searches by
  # specified chromosome
  fn$dbGetQuery(
    vcfData,
    "SELECT m.*, g.AD, CAST(g.DP as INT) AS DP, CAST(g.GQ as INT) AS GQ, g.GT, g.PL
    FROM main_table m
    JOIN genotypeInfo_table g ON m.UNIQUE_ID = g.UNIQUE_ID
    WHERE m.UNIQUE_ID in (
    SELECT UNIQUE_ID
    FROM genotypeInfo_table
    WHERE CAST(DP AS INT)  >= $a
    AND CAST(GQ AS INT)  >= $b)
    AND ALT like '%,%'
    AND CHROM = ('$y')"
  )
}

else if (x == "ALL") {
  # This takes the filter string with everything and searches by
  # specified chromosome
  fn$dbGetQuery(
    vcfData,
    "SELECT m.*, g.AD, CAST(g.DP as INT) AS DP, CAST(g.GQ as INT) AS GQ, g.GT, g.PL
    FROM main_table m
    JOIN genotypeInfo_table g ON m.UNIQUE_ID = g.UNIQUE_ID
    WHERE m.UNIQUE_ID in (
    SELECT UNIQUE_ID
    FROM genotypeInfo_table
    WHERE CAST(DP AS INT)  >= $a
```

```
      AND CAST(GQ AS INT)  >= $b)
      AND CHROM = ('$y')"
    )
  }

  else
    (print("Please use argument 'SNP', 'insertion', 'deletion', 'mixed', or
           'ALL'."))
  }

# This calls the function with argument in quotes.  Use argument SNP, insertion,
# deletion, mixed, or ALL in quotes followed by a comma and the chromosome in quotes
# followed by a comma with a number for depth of coverage followed by a comman and
# a number for quality (up to 99) as shown below:
chrom <- filterByChrom("ALL", "chrX", 10, 50)

# The summary gives information about the data that is stored in the dataframe.
summary(chrom)
```

```
##    UNIQUE_ID         CHROM               POS
##  Min.   :176416   Length:234         Min.   :   283858
##  1st Qu.:177045   Class :character   1st Qu.: 35994108
##  Median :177639   Mode  :character   Median : 74308990
##  Mean   :177701                      Mean   : 83416600
##  3rd Qu.:178399                      3rd Qu.:139899424
##  Max.   :178782                      Max.   :155781810
##       ID                 REF                ALT
##  Length:234         Length:234         Length:234
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##
##
##
##      QUAL              FILTER               AD                  DP
##  Length:234         Length:234         Length:234         Min.   :10.00
##  Class :character   Class :character   Class :character   1st Qu.:18.00
##  Mode  :character   Mode  :character   Mode  :character   Median :22.00
##                                                           Mean   :26.39
##                                                           3rd Qu.:31.00
##                                                           Max.   :74.00
##        GQ             GT                 PL
##  Min.   :50.00   Length:234         Length:234
##  1st Qu.:57.00   Class :character   Class :character
##  Median :69.00   Mode  :character   Mode  :character
##  Mean   :73.32
##  3rd Qu.:95.75
##  Max.   :99.00
```

```
# This prints the first 3 lines.
head(chrom, 3)
```

```
##   UNIQUE_ID CHROM     POS   ID REF ALT    QUAL FILTER   AD DP GQ  GT
## 1    176416  chrX  283858 <NA>   C   T  802.77   PASS 0,20 20 60 1/1
## 2    176424  chrX  302007 <NA>   A   G  875.77   PASS 0,21 21 63 1/1
```

```
## 3    176462  chrX 1310262 <NA>   T   G 1232.77   PASS 0,32 32 96 1/1
##          PL
## 1  831,60,0
## 2  904,63,0
## 3 1261,96,0
```

```
# This prints the last 3 lines.
tail(chrom, 3)
```

```
##     UNIQUE_ID CHROM       POS   ID REF ALT    QUAL FILTER   AD DP GQ  GT
## 232    178780  chrX 155781554 <NA>   T   A 2914.77   PASS 0,70 70 99 1/1
## 233    178781  chrX 155781614 <NA>   C   T 1710.77   PASS 0,43 43 99 1/1
## 234    178782  chrX 155781810 <NA>   T   A  844.77   PASS 0,22 22 66 1/1
##             PL
## 232 2943,211,0
## 233 1739,129,0
## 234   873,66,0
```

4.4.3 Heterozygous or Homozygous

Another usefull tool is looking at if the variant is heterozygous or homozygous. A homozygous trait is when two of the same kind of alleles combine to form a trait. These traits are common in two different creatures which will continue in their offspring. Heterozygous is a trait when two different kinds of alleles combine to form a trait. When this occurs there is a dominant and a regressive allele in the pair, and the dominant allele will indicate which trait the child will get[11]. To filter for heterozygous or homozygous (GT), the GT column in the genotype table needs to be referenced. If the GT is 0/0, it means its a homozygous reference and can be discarded. If the numbers are the same, like 1/1, 2/2,..etc, then it is a homozygous alternate and should be classified as homozygous. If the numbers are different, like 0/1, 1/2,..etc, then it is heterozygous and should be classified as heterozygous:

```
filterByGT <- function(x, y, z) {
  # For SNPs
  if (x == "homozygous") {
    # This takes the filter string and the SNP string
    fn$dbGetQuery(
      vcfData,
      "SELECT m.*, g.AD, CAST(g.DP as INT) AS DP, CAST(g.GQ as INT) AS GQ, g.GT, g.PL
      FROM main_table m
      JOIN genotypeInfo_table g ON m.UNIQUE_ID = g.UNIQUE_ID
      WHERE m.UNIQUE_ID in (
      SELECT UNIQUE_ID
      FROM genotypeInfo_table
      WHERE CAST(DP AS INT)  >= $y
      AND CAST(GQ AS INT)  >= $z
      AND GT NOT LIKE '0/0')
      AND m.UNIQUE_ID in (
      SELECT UNIQUE_ID
      FROM gt_table
      WHERE GT1 == GT2)
      AND CHROM NOT IN ('chrM')
      AND CHROM NOT LIKE ('%chrU%')"
    )
  }

  # For insertions
```

```
  else if (x == "heterozygous") {
    # This takes the filter string with the insertion string
    fn$dbGetQuery(
      vcfData,
      "SELECT m.*, g.AD, CAST(g.DP as INT) AS DP, CAST(g.GQ as INT) AS GQ, g.GT, g.PL
      FROM main_table m
      JOIN genotypeInfo_table g ON m.UNIQUE_ID = g.UNIQUE_ID
      WHERE m.UNIQUE_ID in (
      SELECT UNIQUE_ID
      FROM genotypeInfo_table
      WHERE CAST(DP AS INT)  >= $y
      AND CAST(GQ AS INT)  >= $z
      AND GT NOT LIKE '0/0')
      AND m.UNIQUE_ID in (
      SELECT UNIQUE_ID
      FROM gt_table
      WHERE GT1 != GT2)
      AND CHROM NOT IN ('chrM')
      AND CHROM NOT LIKE ('%chrU%')"
    )
  }

  else
    (print("Please use argument 'homozygous' or 'heterozygous'."))
}

# This calls the function with argument in quotes.  Use argument homozygous or
# heterozygous in quotes followed by comma with a number for depth of
# coverage followed by a comman and a number for quality (up to 99) as shown below:
gt <- filterByGT("heterozygous",10,50)

# The summary gives information about the data that is stored in the dataframe.
summary(gt)
```

```
##    UNIQUE_ID         CHROM                POS
##  Min.   :     5   Length:24591       Min.   :    51661
##  1st Qu.: 33378   Class :character   1st Qu.: 38047969
##  Median : 82017   Mode  :character   Median : 64357508
##  Mean   : 82571                      Mean   : 80877059
##  3rd Qu.:133216                      3rd Qu.:113296648
##  Max.   :178845                      Max.   :248810937
##       ID                 REF                ALT
##  Length:24591       Length:24591       Length:24591
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##
##
##
##      QUAL              FILTER               AD                  DP
##  Length:24591       Length:24591       Length:24591       Min.   : 10.00
##  Class :character   Class :character   Class :character   1st Qu.: 17.00
##  Mode  :character   Mode  :character   Mode  :character   Median : 28.00
##                                                           Mean   : 37.08
##                                                           3rd Qu.: 47.00
```

```
##                                                        Max.   :461.00
##        GQ            GT                PL
##  Min.   :50.0   Length:24591       Length:24591
##  1st Qu.:99.0   Class :character   Class :character
##  Median :99.0   Mode  :character   Mode  :character
##  Mean   :96.1
##  3rd Qu.:99.0
##  Max.   :99.0
```

```
# This prints the firtst 3 lines.
head(gt, 3)
```

```
##   UNIQUE_ID CHROM    POS        ID REF ALT   QUAL FILTER    AD DP GQ  GT
## 1         5  chr1 944296 rs6605067   G   A 176.77   PASS   4,6 10 99 0/1
## 2         7  chr1 944858 rs3748592   A   G 313.77   PASS 15,10 25 99 0/1
## 3        13  chr1 952180 rs3748595   A   C 315.77   PASS 18,10 28 99 0/1
##          PL
## 1 205,0,124
## 2 342,0,515
## 3 344,0,657
```

```
# This prints the last 3 lines.
tail(gt, 3)
```

```
##       UNIQUE_ID                    CHROM   POS   ID REF ALT   QUAL FILTER
## 24589    178843 chr17_GL000205v2_random 57098 <NA>   C   G 181.77   PASS
## 24590    178844 chr17_GL000205v2_random 57142 <NA>   T   C 312.77   PASS
## 24591    178845 chr17_GL000205v2_random 57239 <NA>   G   T 179.77   PASS
##         AD DP GQ  GT        PL
## 24589  6,6 12 99 0/1 210,0,196
## 24590 9,10 19 99 0/1 341,0,309
## 24591 13,7 20 99 0/1 208,0,489
```

5 Conclusion

This program addresses the stated problems with VCF files by putting the data into a normalized 3NF database. The "Analyzing the Data" section acts as a tutorial to help the user understand VCF files and understand the data by performing analysis and filtering.

5.1 Challenges Encountered

There were some issues that were realized while writing this program. One issue was figuring out what to do with VCF files that have more than one sample. Originally the program was designed for a VCF files with one sample, however there are some VCF files that have more than one sample. The fix implemented for this issue was to omit the FORMAT and SAMPLE columns for data greater than one sample. The problem with this approach is it could be difficult to perform advanced filtering techniques without those columns. This is something that could be addressed for future revisions of the program.

Another issue discovered while writing this program was how to store the data. Table 10 shows how the FORMAT and SAMPLE columns are stored in a VCF file:

Table 10: Example FORMAT and SAMPLE columns

FORMAT	SAMPLE
GT:AD:DP:GQ:PL	1/1:0,2:2:40:86,6,0

Originally, the the data was stored as shown below in Table 11:

Table 11: Proposed table

UNIQUE_ID	FORMAT	SAMPLE
1	GT	1/1
1	AD	0,2
1	DP	2
1	GQ	40
1	PL	86,6,0

This approach seemed fine until this data needed to be accessed. It became too complicated to join the data in SQL with the data stored like this.

After some trial and error, the reshape2 package in R proved to be the best. Table 12 shows how the data was reshaped:

Table 12: Final table

UNIQUE_ID	GT	AD	DP	GQ	PL
1	1/1	0,2	2	40	86,6,0

This approach also helped reduce the size of the database by more than 50%.

5.2 Future Work

The filtering needs to be improved. With the current filters there are over 46K variants (reduction of over 66% from the original data), this is still too many for an analyst to review. This data needs to be compared to the UCSC database (https://genome.ucsc.edu/index.html) to remove known harmless variants. With this approach the variants could be reduced to less than one hundred variants, making the data easier to analyze.

6 References

1. https://genomicsandhealth.org/working-groups/our-work/file-formats
2. http://samtools.github.io/hts-specs/
3. Lawrence, M.: 'Introduction to Variant Calling', https://www.bioconductor.org/help/course-materials/2014/CSAMA2014/3_Wednesday/lectures/VariantCallingLecture.pdf
4. https://gatkforums.broadinstitute.org/gatk/discussion/1268/what-is-a-vcf-and-how-should-i-interpret-it
5. https://samtools.github.io/hts-specs/VCFv4.2.pdf
6. Conte, J.: 'PDF workflow of a pipeline using GATK Best Practices', in Editor (Ed.)ˆ(Eds.): 'Book PDF workflow of a pipeline using GATK Best Practices' (2017, edn.), https://docs.wixstatic.com/ugd/f35475_948ba333d23c4d7f91d0524e18591527.pdf.
7. https://www.23andme.com/en-int/gen101/snps/

8. https://en.wikipedia.org/wiki/Indel
9. https://www.mlo-online.com/in-depth-coverage-some-useful-ngs-terms.php
10. Dr. Maria Jose Guillen Sacoto, personal communication, July 21, 2017
11. “Anthropology: A Brief Introduction”; “Ember, Carol and Ember, Melvin”; 1998