UNIVERSITÁ DEGLI STUDI DI VERONA

MASTER DEGREE

in

BIOINFORMATICS AND MEDICAL BIOTECHNOLOGY

MODELING AND IMPLEMENTING A DATA WAREHOUSE TO SUPPORT GENOMIC ANALYSES

Student:

Daniele Castrovilli, VR386161

Prof. Carlo Combi
Advisor

Dott. Gabriele Pozzani
Co-advisor

Dott. Luciano Xumerle
Co-advisor

Index

List of Figures

Abstract

According to Grand View Market Research, the next generation sequencing (NGS) market size was US$2.0 billion globally in 2014 [34]. This number is expected to grow from 2015 to 2022 at an annual rate of about 40%. What drives this phenomenon is the increasing number of treatment options for Precision Medicine. In addition to that, prices of genome sequencing are quickly reducing due to the research and development of rapid, high capacity whole genome sequencers by leading vendors such as Illumina. A growing population, increasing desire for prenatal testing, and cancer cases in an aging population are expected to give boost for next generation sequencing based diagnostics.
A human genome consists of about 3 billion base pairs. If we were to store all of it we would need about 700 megabytes, assuming that there are no need to include information on data quality along with the sequence. In that case, all we would need is the string of letters A, C, G, and T that make up one strand of the human genome, but that is not a valid assumption. This number is actually much higher, about 200 gigabytes [31], assuming we store all the short reads produced by a sequencer at a 30x coverage rate. A compressed representation of this information is the genomic variant called format file (gVCF-file), because only about 0.1% of the genome is different among individuals, which equates to about 3 million variants in the average human genome.

Thus, we have to store about 3 million variants. Depending on what kind of data we keep, our storage requirements are between 200 megabytes and or 200 gigabytes per sample. For example, it would make sense to retain all BAM (tab-delimited text file that contains sequence alignment data) files associated with a particular sample to display the aligned reads in a Genome Browser. In addition to that, we have to save clinical reports and meta information about the filtering process.

The numbers above obviously require a Big Data approach. The necessity for centralized and genetically aware data management. Essentially, laboratories require an integrated approach to store, manage and gain insights from samples produced in a clinical setting.

With this task in this work we describe how we develop a fully functional dataware house in all of it's aspects, from designing the architecture to data manipulation and retreival with open source tools.

Chapter 1

Introduction

1.1 Problem

Storing, analyzing and retrieving a big amount of data is one of the most complex trouble of this century, expecially when it comes from sequencing human DNA[11]. This huge amount of data needs to be analyzed because it helps genetists to predict new genetic disorders and could help in the creation of new open databases spreading useful knowledge across scientific communities.
Leaving this information untapped creates a great gap in the comprehension of human DNA, as there are a lot of uncovered DNA secrets to be discovered and analyzed. Having a good database system capable of retrieving data and doing heavy calculations, could help in discovering new genes and their mutations.
Thanks to the calculation of the genotipic and allelic frequencies genetists can predict the spreading of new or mutated disease. Predicting new genes or their mutation could help the creation of personalized drugs, aimed specifically for the patient and the disease that is suffering from. So, the major problem we are facing is the lack of systems that can analyze this data using an approach capable of retrieve informations from such data. The data warehouse can be the best way to achieve the goal of creating a system capable of managing tons of data coming from sequencing and to make possible the analysis from different points of view from calculation of frequencies to showing the information with charts [29].

1.2 Aim

We want to develop not only a storage system with pre-processing steps, but also a data warehouse capable of managing standard format files for storing a single sequenced genome and analyzing it in order to retrieve informations. Treating the genome as a big data [4], thanks to the endless information stored in it, we can develop a system capable of reading the related informations. This system could help the analyses of huge data, like genomes, while also calculation of gene positions. Allele and genotypic frequencies calculations can benefit from a business intelligence system [17]. A data warehouse can help the creation of family trees, analyzing the DNA of an entire family calculating the probabilities of handing down a possible hereditary genetic disorder from ancestors to children. Those systems could help open a wide range of use cases, in which the patient is the main character of the entire analysis. With a data warehouse sytem it is easier to develop a

”mathematical hereditary model”. Can be developed a warehouse system model for each hereditary genetic disorder. Having a mathematical map of every genetic disorder could help the development of Artificial Intelligence (A.I.) systems capable of making predictions helping physicians in treating the disease [19]. In this thesis we describe all the steps needed to develop a data warehouse system, capable of managing genetic standard file format and calculate the allelic and genotypic frequency, and discussl how to prepare a standard formatted genomic file with bionformatic tools. Beside these important steps, we created also a dashboard needed for providing a guided analysis and visual representation to biologists thanks to charts and tables, making also possible to browse the genomic information stored.

1.3 Structure

Each one of the following chapters is focused on a particular aspect of designing and developing a data warehouse system.

- In Chapter 2 we introduce some basic concepts, useful to understand the different steps of the development .
  We range from medical genetics to data warehouse and what are the steps of developing a data warehouse system passing through the sequencing and what data is produced.
- Chapter 3 focuses on showing the heart of the system, the designin of the main components of the system describing fact and dimensions tables, the developing of a tree hierarchy from a directed acyclic graph and the steps for creating the OLAP cube ending with the design of the dashboard and it’s components.
- In Chapter 4 we face the tuning problem, i.e. what is the best configuration in order to get the best performances, focusing on read performances. The description of the folder hierarchy helps to navigate through the folders and to understand how ETL manage the sequenced genomes files. In this chapter we describe also Pentaho components and the bioinformatic tool to perform the pre-process step. A use case is showed and a validation has been described.
- Finally in Chapter 5 there are the future developments of what the system can become with further development and the main conclusions of the developer about making this kind of system.

Chapter 2

Background and related work

In this chapter we focus on the concept needed for a good readibility of the following work, starting from what medical genetics and DNA sequencing are, how to calculate genotypic and allelic frequencies, to the main components of a data warehouse and all the tools we need to make a data warehouse system for genomic analysis.

2.1 Medical genetics

Genetics is a wide research area spreading from the collection of information about genes on our chromosomes to the analysis of human genome [18]. In particular, medical genetics is a branch of medicine that involves the diagnosis and management of hereditary disorders.
In the past 20 years such disorders were highlited simply by the ”Karyotype”, that is a test to identify and evaluate the size, shape, and number of chromosomes in a sample of body cells. If some of those chromosomes had a different shape or size, then the patient have a genetic disorder. To understand what a ”Phenotype” is, we can resume it in ”What is shown externally”, from eyes colour, type of hair, height, and so on . We have in total 22 pairs of chromosomes plus the sexual chromosomes X and Y. Humans are called diploid organisms, this because our chromosomes are in double copy in each cell.

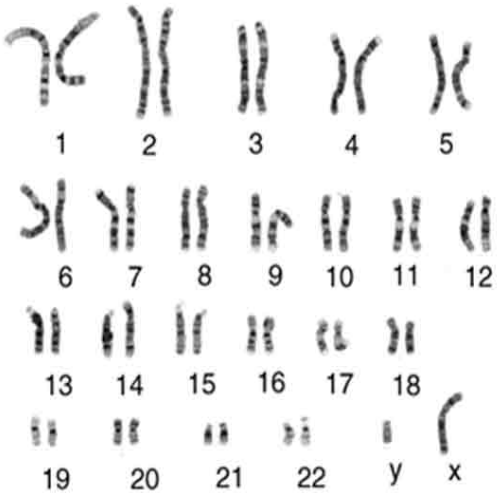

Figure 2.1: Example of male kariotype

Understanding hereditary disorders is one of the main goal of the 21st century in medicine. This can help us to avoid spreading of deadly diseases and to develop aimed drugs and therapies. Everything will flow into ”Personalized genetics” [16], where we develop drugs tailored uniquely for the patient. The sequenced DNA allows us to retrieve what is called a ”variant”. Human genetic variation is the genetic differences both within and among populations. A genetic variation among humans occurs on many scales, from whole alterations in the human karyotype to single nucleotide changes. Chromosomes stores our gene in ”locus” or ”allele”. Such ”gene” are a collection of strings containing only nitrogenous bases A, T, C, and G, called nucleotides, which influences our phenotype. The gene could also have a mutant ”brother” called ”Allele”. Sometimes, different alleles can result in different observable phenotypic traits like diseases, different eye colour, and so on. Usually the DNA is double stranded so the couple of nucleotides forms two couples AT and CG. These base pairs are called complementary.

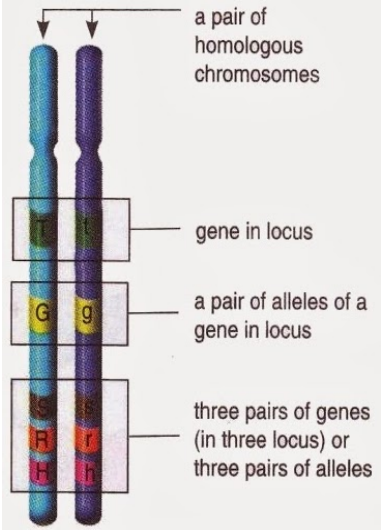

Figure 2.2: Example of allele positions in chromosomes biology [33]

As shown in Figure 2.1, we can see how are positioned our genes in the chromosomes. Figure 2.1 shows clearly the homologous alleles used by genetists underline where is a variant.
A single nucleotide polymorphism (SNP) is a difference in a single nucleotide between members of one species that occurs in at least 1% of the population. It is estimated that there are 10 to 30 million SNPs in humans. The phenotypic effects of a SNP are different and in some diseases a single SNP could cause the onset of a pathology, like ”Skickle-cell anemia”, where the Beta-globin gene is being mutated (A to T) giving the ”classic” sickle shape to the blood cell of the patient [9].
Fusing the importance of medical genetics to the power of nowadays computing we can bring rich informations of our genetic asset. We can overcome of the human brain limits like speed of calculations and reducing the error rate.

2.2 Sequencing

The first DNA sequencing was performed in the early 1970s [14], by academic researchers using laborious methods based on two-dimensional chromatography. Since then this technique evolved in a less expensive and more efficient i method using fluorescence-based sequencing [30].
In the mid to late 1990s were developed DNA sequencers. From now on they started the age of ”High-throughput sequencing methods” where the consequences werean increasing number of data acquired thanks to the lowering of the overall cost pof such technology.

Method	Cost per 1 million bases (in US$)
Chain termination	$2400
Pyrosequencing	$10
Ion semiconductor	$1
Sequencing by ligation	$0.13
Nanopore Sequencing	$0.11
Sequencing by synthesis	$0.05

Table 2.1: Cost of DNA sequencing by technique [13]

In Table 2.1 is clear the decrease of costs depending on the tecnique used. The decrease of the sequencing base price fuels the demand of DNA sequencing not only for medical purposes but even for prevention of diseases like breast or ovaric cancer, bringing to us more data to manage and store.

How does a sequencer work?
A sequencer is a machine capable of reading the DNA molecule returning a format file containing all the read DNA sequence.

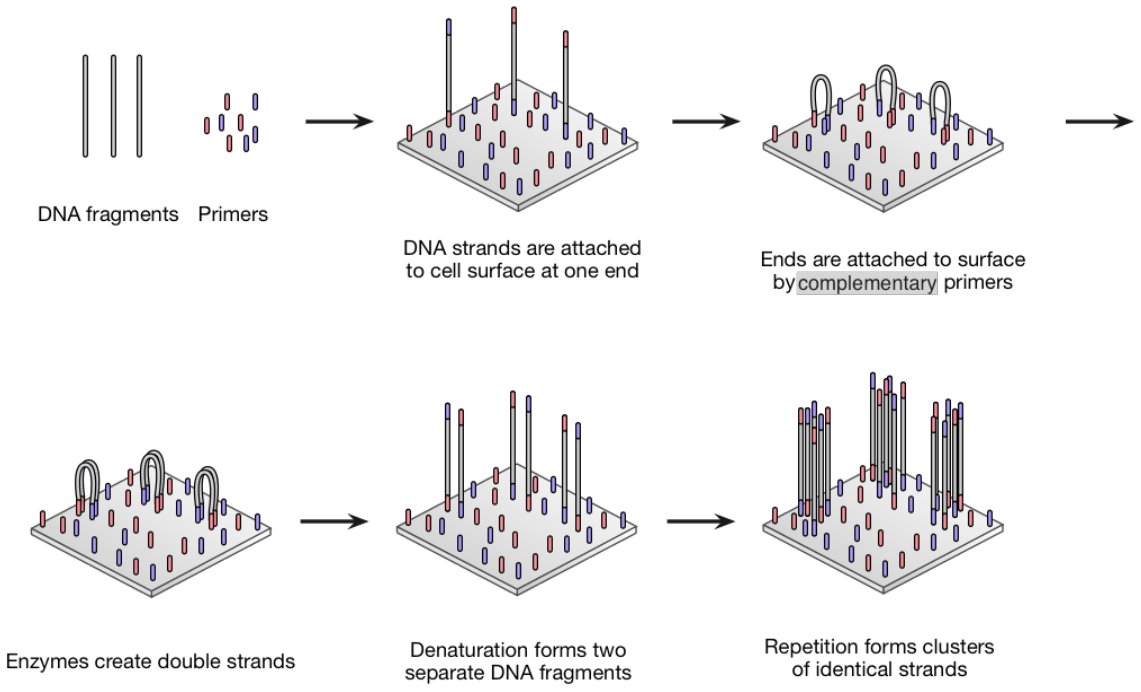

Figure 2.3: Example of sequencing

Figure 2.3 shows simple steps of sequencing by sinthesis [15].
The steps of the procedure depicted in figure 2.3 are simple:

- The DNA is being extracted from the sample human cell
- A flow cell (a plate) is being prepared with some pieces of DNA sinthetized artificially positioned on the surface
- The sample DNA will form a bridge with the complementary nucleotides on the plate basis
- A DNA enzyme will create a second strand upon the sample one
- The double bridge DNA will be separated into two single ones by increasing the temperature (Usually 70°C)
- Then a DNA polymerase (a protein that can create DNA from a strand using nucleotides) will start to create a new strand upon the single pieces of DNA previously separated
- The DNA polymerase will use special nucleotides. Those nucleotides are modified so that every molecule added in the creation of a strand will release a fluorescent light read by the machine

So if a DNA strand is composed by A-T-C, the enzyme will create the second strand using T-A-G thanks to its complementarity (AT, CG). Each nucleotide insertion will release a fluorophore (An enzyme that light up by its own) read by the sequencer. Thanks to the unique complementarity (AT and CG) of nucleotides, we know that the original strand is A-T-C. Like that we can read every DNA sample we want.
So thanks to the decline of the cost of sequencing, reading the human DNA from every individual becomes a must if we want to understand better our genetic asset, this provokes an explosion of data as consequence.

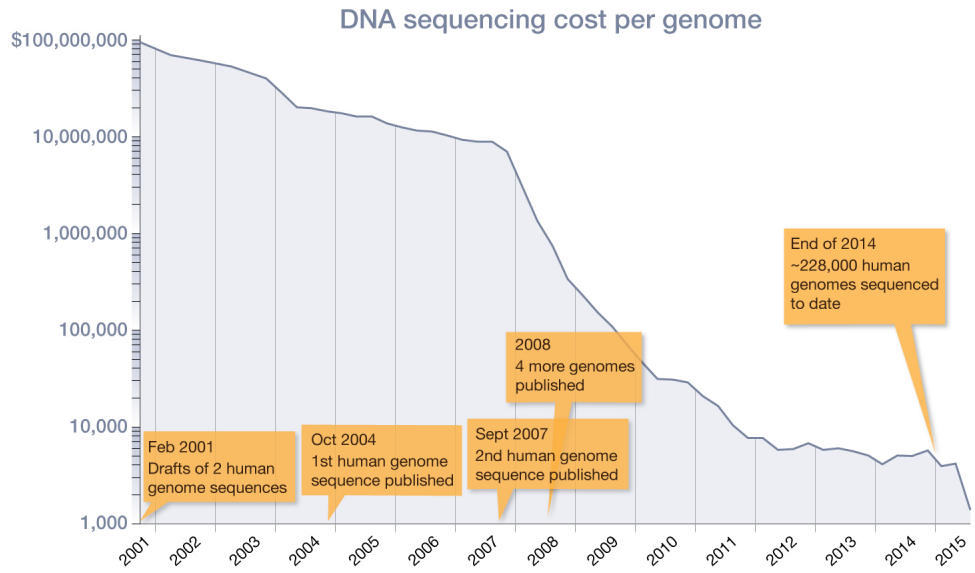

Figure 2.4: Cost per year of Dna sequencing [1]

Figure 2.1 shows the decline of sequencing costs and the number of genomes sequenced by milestone.
Storing all the information acquired by those technologies requires a specific file format called Variant Call Format (VCF) or genomic Variant Call Format (gVCF) that is the standard de facto file format for storing all the variations needed to be observed. gVCF is a text file format (most likely stored in a compressed manner). It contain meta-information lines, a header line, and then data lines each are about a position in the genome. The format also has the ability to store on samples for each position called ”Variant” and we can know which gene is mutated and what is its main function on our genome. Every variant is called via gVCF tools that uses the ”Phred scrore quality” in order to give a probability of success of calling a variant.

2.2.1 Phred quality score

Each variant is literally ”called” and this probability of calling a base were developed by Drs. Phil Green and Brent Ewing.
A Phred quality score is a measure of the quality of the identification of the nucleotides generated by DNA sequencing. Phred quality scores are assigned to each nucleotide base call in automated sequencer traces. Phred quality scores have become widely accepted to characterize the quality of DNA sequences, and can be used to compare the efficacy of different sequencing methods [7]. In todays sequencing output, by convention, most useable Phred-scaled base quality scores range from 2 to 40, with some variations in the range depending on the origin of the sequence data (see the FASTQ format documentation for details). However, Phred-scaled quality scores in general can range anywhere from 0 to infinity. A higher score indicates a higher probability that a particular decision is correct, while conversely, a lower score indicates a higher probability that the decision is incorrect.

The Phred quality score (Q) is logarithmically related to the error probability (E).

$$Q = -10logE$$

So we can interpret this score as an estimate of error, where the error is e.g. the probability that the base is called incorrectly by the sequencer, but we can also interpret it as an estimate of accuracy, where the accuracy is e.g. the probability that the base was identified correctly by the sequencer. Depending on how we decide to express it, we can make the following calculations:

If we want the probability of error (E), we take:

$$E = 10^{-(Q/10)}$$

If we want to express this as the estimate of accuracy (A), we simply take

$$A = (1-E) = 1-10^{-(Q/10)}$$

For many purposes, a Phred Score of 20 or above is acceptable, because this means that whatever it qualifies is 99% accurate, with a 1% chance of error [32].

```
##INFO=<ID=CIGAR,Number=A,Type=String,Description="CIGAR alignment for each alternate indel allele">
##INFO=<ID=RU,Number=A,Type=String,Description="Smallest repeating sequence unit extended or contracted in the indel allele relative to the reference. RUs are
 not reported if longer than 20 bases.">
##INFO=<ID=REFREP,Number=A,Type=Integer,Description="Number of times RU is repeated in reference.">
##INFO=<ID=IDREP,Number=A,Type=Integer,Description="Number of times RU is repeated in indel allele.">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=GQ,Number=1,Type=Float,Description="Genotype Quality">
##FORMAT=<ID=GQX,Number=1,Type=Integer,Description="Minimum of {Genotype quality assuming variant position,Genotype quality assuming non-variant position}">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Filtered basecall depth used for site genotyping">
##FORMAT=<ID=DPF,Number=1,Type=Integer,Description="Basecalls filtered from input prior to site genotyping">
##FORMAT=<ID=AD,Number=.,Type=Integer,Description="Allelic depths for the ref and alt alleles in the order listed. For indels this value only includes reads w
hich confidently support each allele (posterior prob 0.999 or higher that read contains indicated allele vs all other intersecting indel alleles)">
##FORMAT=<ID=DPI,Number=1,Type=Integer,Description="Read depth associated with indel, taken from the site preceding the indel.">
##FILTER=<ID=IndelConflict,Description="Locus is in region with conflicting indel calls">
##FILTER=<ID=SiteConflict,Description="Site genotype conflicts with proximal indel call. This is typically a heterozygous SNV call made inside of a heterozygo
us deletion">
##FILTER=<ID=LowGQX,Description="Locus GQX is less than 30 or not present">
##FILTER=<ID=HighDPFRatio,Description="The fraction of basecalls filtered out at a site is greater than 0.3">
##FILTER=<ID=HighSNVSB,Description="SNV strand bias value (SNVSB) exceeds 10">
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  default
chr1    12699   .       T       .       .       PASS    END=12719;BLOCKAVG_min30p3a     GT:GQX:DP:DPF   0/0:30:11:0
chr1    12720   .       A       .       .       PASS    END=12723;BLOCKAVG_min30p3a     GT:GQX:DP:DPF   0/0:33:12:0
chr1    12724   .       G       .       .       PASS    END=12731;BLOCKAVG_min30p3a     GT:GQX:DP:DPF   0/0:45:16:0
chr1    12733   .       G       .       .       PASS    END=12747;BLOCKAVG_min30p3a     GT:GQX:DP:DPF   0/0:48:17:0
chr1    12748   .       G       .       .       PASS    END=12757;BLOCKAVG_min30p3a     GT:GQX:DP:DPF   0/0:63:22:0
chr1    12758   .       G       .       .       PASS    END=12782;BLOCKAVG_min30p3a     GT:GQX:DP:DPF   0/0:81:28:0
chr1    12783   .       G       A       117     PASS    SNVSB=0.0;SNVHPOL=2     GT:GQ:GQX:DP:DPF:AD     0/1:118:114:30:1:13,17
chr1    12784   .       T       .       .       PASS    END=12806;BLOCKAVG_min30p3a     GT:GQX:DP:DPF   0/0:102:35:0
chr1    12807   .       C       .       .       PASS    .       GT:GQX:DP:DPF   0/0:47:42:0
chr1    12808   .       G       .       .       PASS    END=12831;BLOCKAVG_min30p3a     GT:GQX:DP:DPF   0/0:129:44:0
chr1    12832   .       C       .       .       PASS    END=12849;BLOCKAVG_min30p3a     GT:GQX:DP:DPF   0/0:169:57:0
chr1    12850   .       T       .       .       PASS    END=12855;BLOCKAVG_min30p3a     GT:GQX:DP:DPF   0/0:226:76:1
chr1    12856   .       C       .       .       PASS    END=12869;BLOCKAVG_min30p3a     GT:GQX:DP:DPF   0/0:253:85:0
chr1    12870   .       T       .       .       PASS    END=12875;BLOCKAVG_min30p3a     GT:GQX:DP:DPF   0/0:285:103:1
chr1    12876   .       A       .       .       PASS    END=12881;BLOCKAVG_min30p3a     GT:GQX:DP:DPF   0/0:382:128:1
chr1    12882   .       C       .       .       PASS    END=12884;BLOCKAVG_min30p3a     GT:GQX:DP:DPF   0/0:376:126:3
chr1    12885   .       C       .       .       PASS    END=12901;BLOCKAVG_min30p3a     GT:GQX:DP:DPF   0/0:403:135:0
chr1    12902   .       G       .       .       PASS    .       GT:GQX:DP:DPF   0/0:479:160:5
chr1    12903   .       C       .       .       PASS    END=12907;BLOCKAVG_min30p3a     GT:GQX:DP:DPF   0/0:446:164:0
chr1    12908   .       C       .       .       PASS    END=12915;BLOCKAVG_min30p3a     GT:GQX:DP:DPF   0/0:494:165:1
chr1    12916   .       T       .       .       PASS    END=12924;BLOCKAVG_min30p3a     GT:GQX:DP:DPF   0/0:524:175:0
chr1    12925   .       A       .       .       PASS    .       GT:GQX:DP:DPF   0/0:518:173:5
chr1    12926   .       A       .       .       PASS    END=12931;BLOCKAVG_min30p3a     GT:GQX:DP:DPF   0/0:518:173:0
chr1    12932   .       C       .       .       PASS    END=12945;BLOCKAVG_min30p3a     GT:GQX:DP:DPF   0/0:510:182:1
chr1    12946   .       T       .       .       PASS    .       GT:GQX:DP:DPF   0/0:524:175:9
chr1    12947   .       C       .       .       PASS    END=12953;BLOCKAVG_min30p3a     GT:GQX:DP:DPF   0/0:461:164:2
[daniele@kgn03-ddlab GVCF]$ _
```

Figure 2.5: Example of gVcf file

In Figure 2.2 we can observe a typical gVCF file structure

2.2.2 gVCF file structure explanation

- CHROM:
  Chromosome: an identifier from the reference genome or an angle-bracketed ID String (”¡ID¿”) pointing to a contig.

- POS:
  Position: The reference position, with the 1st base having position 1. Positions are sorted numerically, in increasing order, within each reference sequence CHROM. There can be multiple records with the same POS. Telomeres are indicated by using positions 0 or N+1, where N is the length of the corresponding chromosome or contig.

- ID:
  Semi-colon separated list of unique identifiers where available. If this is a dbSNP variant it is encouraged to use the rs number(s). No identifier should be present in more than one data record. If there is no identifier available, then the missing value should be used.

- REF:
  Reference base(s): A,C,G,T,N; there can be multiple bases. The value in the POS field refers to the position of the first base in the string. For simple insertions and deletions in which either the REF or one of the ALT alleles would otherwise be null/empty, the REF and ALT strings include the base before the event (which is reflected in the POS field), unless the event occurs at position 1 on the contig in

which case they include the base after the event. If any of the ALT alleles is a symbolic allele (an angle-bracketed ID String "¡ID¿") then the padding base is required and POS denotes the coordinate of the base preceding the polymorphism.

- ALT:
  Comma separated list of alternate non-reference alleles called on at least one of the samples. Options are: Base strings made up of the bases A,C,G,T,N
  - Angle-bracketed ID String (¡ID¿)
  - Breakend replacement string as described in the section on breakends.
  - If there are no alternative alleles, then the missing value should be used. QUAL: Phred-scaled quality score for the assertion made in ALT.
    If ALT is . (no variant) then this is p(variant), and if ALT is not . this is p(no variant). High QUAL scores indicate high confidence calls. Although traditionally people use integer phred scores, this field is permitted to be a floating point to enable higher resolution for low confidence calls if desired. If unknown, the missing value should be specified. (Numeric)

- FILTER:
  PASS if this position has passed all filters, i.e. a call is made at this position. Otherwise, if the site has not passed all filters, a semicolon-separated list of codes for filters that fail. gVCF files use the following values:
  - PASS: position has passed all filters
  - IndelConflict: Locus is in region with conflicting indel calls.
  - SiteConflict: Site genotype conflicts with proximal indel call. This is typically a heterozygous SNV call made inside of a heterozygous deletion.
  - LowGQX:
    Locus GQX is less than 30 or not present.
  - HighDPFRatio:
    The fraction of basecalls filtered out at a site is greater than 0.3.
  - HighSNVSB:
    SNV strand bias value (SNVSB) exceeds 10.
  - HighSNVHPOL:
    SNV contextual homopolymer length (SNVHPOL) exceeds 6.
  - HighREFREP:
    Indel contains an ellele which occurs in a homopolymer or dinucleotide track with a reference repeat greater than 8.
  - HighDepth:
    Locus depth is greater than 3x the mean chromosome depth.

- INFO:
  Additional information. INFO fields are encoded as a semicolon-separated series of short keys with optional values in the format: ¡key¿=¡data¿[,data]. gVCF files use the following values:

  - END:
    End position of the region described in this record.
  - BLOCKAVG_min30p3a:
    Non-variant site block. All sites in a block are constrained to be non-variant, have the same filter value, and have all sample values in range $[x,y], y <= max(x+3,(x*1.3))$. All printed site block sample values are the minimum observed in the region spanned by the block.
  - SNVSB:
    SNV site strand bias.
  - SNVHPOL:
    SNV contextual homopolymer length.
  - CIGAR:
    CIGAR alignment for each alternate indel allele
  - RU:
    Smallest repeating sequence unit extended or contracted in the indel allele relative to the reference. RUs are not reported if longer than 20 bases.
  - REFREP:
    Number of times RU is repeated in reference.
  - IDREP:
    Number of times RU is repeated in indel allele.

- FORMAT:
  Format of the sample field. FORMAT specifies the data types and order of the subfields. gVCF files use the following values:

  - GT:
    Genotype.
  - GQ:
    Genotype Quality.
  - GQX:
    Minimum of Genotype quality assuming variant position,Genotype quality assuming non-variant position.
  - DP:
    Filtered basecall depth used for site genotyping.
  - DPF:
    Basecalls filtered from input prior to site genotyping.
  - AD:
    Allelic depths for the ref and alt alleles in the order listed. For indels this value only includes reads which confidently support each allele (posterior prob 0.999 or higher that read contains indicated allele vs all other intersecting indel alleles).
  - DPI:
    Read depth associated with indel, taken from the site preceding the indel.

- SAMPLE:
  Sample fields as defined by the header.

File *.gvcf* is the starting point to develop the SQL algorithms for calculating the allelic and genotypic frequencies. In fact the column ref and alt will be used in order to count the occurrencies of the desired variant and or allele and extract the frequency.

Allelic and genotype frequencies

The calculationof the frequencies is the heart of our system just because, as explained in Aim section we want to develop a system capable of makig those calculations. We start giving two important definitions:

- **Genotype frequency** in a population is the number of individuals with a given genotype divided by the total number of individuals in the population [28].
- **Allele frequency**, or gene frequency, is the relative frequency of an allele (variant of a gene) at a particular locus in a population, expressed as a fraction or percentage [10].

The study of those two frequencies helps genetist to discover new genetic disease, observe the human genetic evolution and helps to create prevention systems aganist genetic disorders. A typical example of genetic study is the *Beta thalassemia* in Sardinia population, involving the study of their genetic asset [2], that helped in recognize the genetic descendance of the pathology. The relationship between allele frequencies and genotype frequencies in populations is the Hardy-Weinberg Equilibrium, that is usually described using a trait for which there are two alleles present at the locus of interest. HardyWeinberg equilibrium, states that allele and genotype frequencies in a population will remain constant from generation to generation in the absence of other evolutionary influences.

Question: How do we use the Hardy-Weinberg model to predict genotype and allele frequencies?
The Hardy-Weinberg model consists of two equations: one that calculates allele frequencies and one that calculates genotype frequencies. Because we are dealing with frequencies, both equations must add up to 1. The equation

$$p+q=1$$

describes allele frequencies for a gene with two alleles.
In a diploid organism with alleles A and a at a given locus, there are three possible genotypes: AA, Aa, and aa. If we use p to represent the frequency of A and q to represent the frequency of a, we can write the genotype frequencies as (p)(p) or p2 for AA, (q)(q) or q2 for aa, and 2(p)(q) for Aa. The equation for genotype frequencies is:

$$p2+2pq+q2=1.$$

We can use these data to calculate the allelic frequencies for a given locus, such as the EST locus in a population (n = 64). Each individual with the genotype SS has two copies of the S allele; therefore the 37 individuals with this genotype have a count of 74 S alleles. Heterozygote individuals (SF) have one of each allele, so there are 20 S alleles and 20 F alleles among them. Like the SS homozygotes, individuals with the FF genotype have two copies of the F allele, so these seven individuals contribute 14 F alleles to our count. In other words, among the 64 individuals in this sample there are 94 S alleles and 34 F alleles. To calculate the allelic frequencies we simply divide the number of S or F alleles by the total number of alleles: $94/128 = 0.734 = p$= frequency of the S allele, and $34/128 = 0.266 = q$= frequency of the F allele [12].

Once the variant has be confirmed in a particular gene, and the frequency trend is positive (which means that the frequency of the variant becomes higher in time) it is assigned to one of the relative Human Phenotype Ontology terms.

Human Phenotype Ontology

Human phenotype Ontology is a powerful tool for annotating the hereditary human diseases. The main aim of this tool is to give a disease name for each phenotypic combination found during DNA sequencing. With over 8000 terms representing individual phenotypic anomalies this tool is really powerful for annotating diseases. HPO is able to capture phenotypic similarities between diseases in a useful and highly significant fashion [26]. This tool has been developed by Sebastian Kohler and all relationships in the HPO are is-a relationships, i.e. simple class-subclass relationships. For example, Abnormality of the feet is-a Abnormality of the lower limbs. The relationships are transitive, meaning that they are inherited up all paths to the root. Phenotypic abnormality is the main subontology of the HPO and contains descriptions of clinical abnormalities. All those informations are stored in a *.SQL* file format, ready to be imported in every database that needs those informations. In this file are stored every HPO term and its unique id. We can retreive the term we needed most simply looking for its ID or knowing the disease we could know the associated id. Every method of retreiving data (IDs, names, terms etc...) from the databases is possible thanks to the Structured Query Language, which is a powerful language developed for databases.

2.3 Data warehouse

A data warehouse (DW or DWH) is a system designed mainly for business intelligence and decision support where a user can retrieve the informations from a business unit or operation. DWs are central repositories of integrated data from one or more disparate sources. They store current and historical data and are used for creating analytical reports for knowledge workers for each area of the enterprise [20].
A data warehouse is described as a "Decision making process support system" which has some features:

- **Time-Variant**
  Stores historical data.
  One of the best features of a data warehouse is the availability of historical data. A user can observe the trend of a business unit and take decision from them. A trend can be developed using historical data from first upload to the last one.

- **Integrated**
  Allows multiple sources of information
  For a business is important to collect as many data as it can, expecially if they comes from external sources. We can take advantages from other projects and integrate the information in the system. A data warehouse is capable of integrating different kind of source information in order to use at full potential.
  Every data that comes from other sources is being integrated into the system thanks to the ETL step.

- **Subject oriented**
  Analyse a particular subject area.
  A particular feature of the data warehouse is that every business unit can develop their own data mart and start to observe all the information they need in order to make decisions. Every data mart can be seen as a particular point of view of the analysis.

- **Non volatile**
  Once data is written it cannot change
  Every data written in the system will not be phisically deleted or modified. This feature allows a complete view of data. A business unit can observe all the changes, uploads and logical deletions that happened on the data.

The data stored in the warehouse is uploaded from the operational systems (like repository databases), the data may pass through an operational data store for additional operations before it is used in the DW for reporting. Operational databases are repositories where data is stored and managed with the main aim of Online Transactional Processing (OLTP) UPDATE, SELECT, DELETE.
In other hand a data warehouse is used for Online Analytical Processing (OLAP), operation in which we can compute actions such as SUM, SUBTRACT, AVG computing complex calculations and reads the historical data for business decisions.

A typical data warehouse architecture is showed in figure 2.3 , where we have all the possible source of information that are integrated and polished via the ETL steps which stores them in the data warehouse system which is used for dashboard, reporting and data mining. The modeling phase of a data warehouse is called "Dimensional Fact Model"

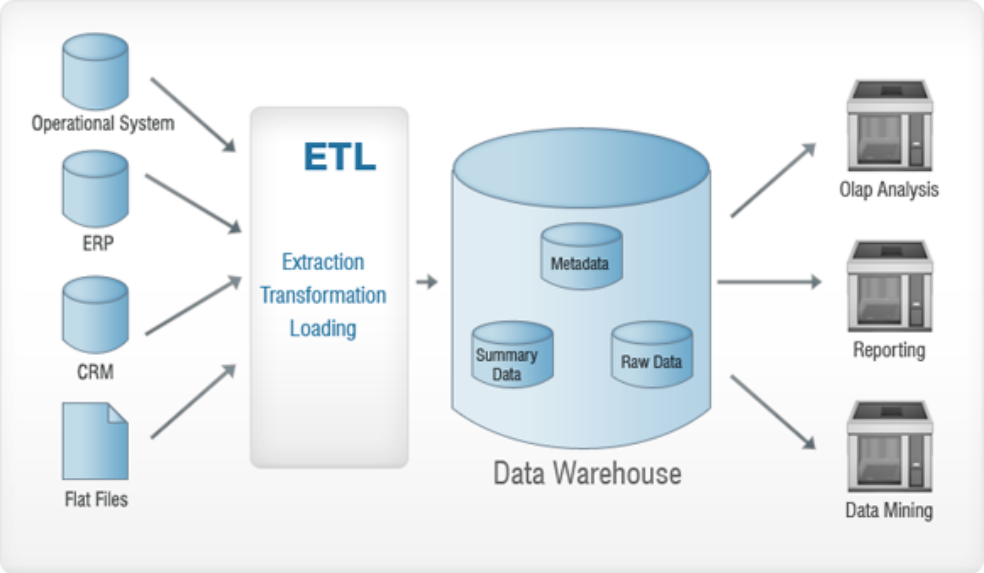

Figure 2.6: Data warehouse architecture

(DFM) which is an ad-hoc graphical formalism and is extremelly intuitive from tech to non-tech users.

The Dimensional Fact Model

The basic concepts of a DFM are:

- Fact: Is a concept relevat for the decision making process
  This is the main reason, why the data warehouse should exists and the main objective to be analyzed. (E.g. We want to see the number of variant in our chromosome asset)

- Measure: Is a numerical property of the fact and describes quantitative properties to the analysis
  Every count is stored in those ”Measures” that can be identified as the numerical aspect of our fact (E.g. Number of variant)

- Dimension: It describes the analysis coordinate of the fact
  (E.g. Number of variant in a chromosome or number of variant in a range of position.)

The time is always a very important dimension, because data warehouse systems are made primarily to get an historical information of our data.

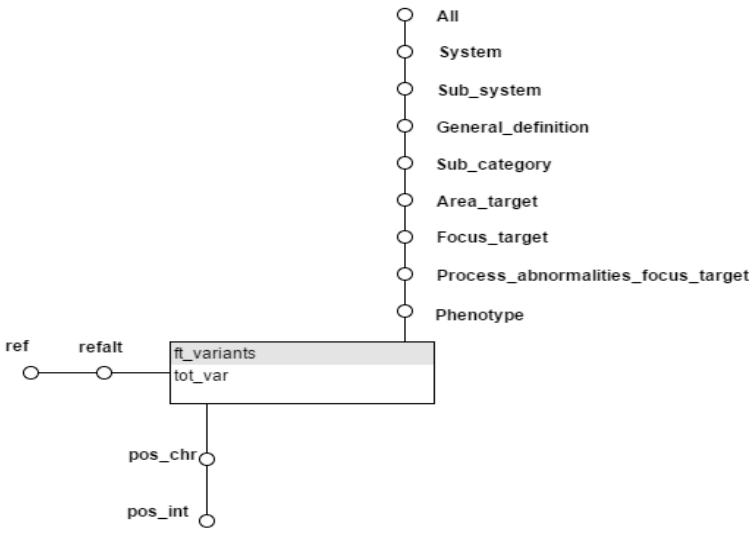

Figure 2.7: DFM modeling example

Conceptual design of DW is based on dimensional modeling (DM), that is a set of techniques and concepts used to describe data warehouses. It is considered to be different from entity-relationship modeling (ER). Dimensional modeling does not necessarily involve a relational database. The same modeling approach, at the logical level, can be used for any physical form, such as multidimensional database or even flat files. Dimensional modeling always uses the concepts of facts (measures), and dimensions (context). Facts are typically (but not always) numeric values that can be aggregated, and dimensions are groups of hierarchies and descriptors that define the facts. For example, sales amount is a fact; timestamp, product, register#, store#, etc. are elements of dimensions.

The benefits of a dimensional model are:

- **Understandability**.
  Dimensional model is easier to understand and more intuitive to any other kind model used in database data modeling. In dimensional models, information is grouped into coherent dimensions, making it easier to read and interpret. It is also supported by the DMF design described in Figure 2.4, understandable by all users who will use the system and this can lead in a malleable conceptual steps where also the final concept can be updated following the needs of a dynamic business. Understand the DM is really important in a business this is a basic requirement for the next steps that makes the system stable and reliable.

- **Query performance**.
  Dimensional models are more denormalized and optimized for data querying, while normalized models seek to eliminate data redundancies and are optimized for transaction loading and updating. This model allows the database to make strong assumptions about the data, which may have a positive impact on performance. Each dimension is an equivalent entry point into the fact table, this allows to use an all indexed environment that brings significant improvements in speed when searching for elements. Thanks to the understandability of the DM, we can also project all the queries needed for retreiving the data stored in the data warehouse this is clear that is a major point in favour of this system instead of using a classical RDBMS

for OLAP queries.

- **Extensibility**.
  Existing tables can be changed in place either by simply adding new data rows into the table or executing SQL alter table commands. No queries or applications that sit on top of the data warehouse need to be reprogrammed to accommodate changes. Old queries and applications continue to run without yielding different results. So every table can be updated or truncated and updated following the needs of the business analysts, those aspects are important because they offers a flexible environment with less time for maintenance needed for the system.

2.3.1 Development of a data warehouse

Typically, we have four development levels in a data warehouse which can be seen in Figure 2.3, from the data gathered from operational databases to the final data mart (or cube). A cube, or data mart, is the final object we want to develop thanks and to this data structure we can operate in a faster way compared to a classic SQL system. Part of those benefits are described in the benefits of the dimensional model described in the previous section.

- Source level
  Level where we have our operational databases in which are stored our data, plus other databases to integrate in the system. A source can be an Excel file, a CSV file, a database along the internet. For example even Twitter can be used as source database, thanks to its APIs (Application programming interface), for analyze the moods or preventing flu [23]. Every source of information can be used and integrated into a data warehouse thanks to the feeding level which integrates all the data into the DB.

- Feeding Level
  This is the step where we proceed with the Extraction, Trasformation and Loading (ETL), it is required in order to extract all the data needed from the operational database and external ones, manipulate and polish the data (e.g. avoid headers, clean or removing of corrupted data), and start load data in the data warehouse. This level is important because defines the identity of our data warehouse creating an environment which can be further used and improved.

  ETL steps:

  - Extraction.
    The first feeding of data warehouse from operational databases is being made often initially by a snapshot of the source DB (e.g. Excel file, other online databases), then the snapshot will be only updated or deleted and re-created each time. The extraction can be done reading files or using APIs offered by those services which allows the read of their data like Twitter, which allows to read the stramline of the users, or one in particular.
  - Cleaning.
    Removal or correction of all inconsistencies in data, impossible or incorrect values and removal of typing errors. This step is crucial to delete all the data which isn't useful or corrupted, giving at first glance a polished system.
  - Trasformation.
    Selecting only data needed, matching between fields of different sources and manipulation of data. This step allows in specific to define how your data is stored definng the columns name making them standard inside the business.
  - Loading.
    Final part of this step, we upload our data in two different ways:
    * **Truncate and update**
      Deleting everything previously written in DW and loading from scratch every time, this could be a faster approach but it's more difficult to mantain a history of data because we delete everything each upload;

  - **Update only**
    Updating the data already in, avoiding huge uploads and updating data if being modified, this approach could be slower in the longtime expecially when there are TB of data to lookup for an update, but this approach guarantee a history in data mantaining all the older information.
    This choice could impact on ETL depending if data is updated often or if every uploads adds lot of new data.

- Warehouse level
  This is the main level where data is stored after being polished and eventually modified in order to satisfy business requirements. In order to use the ”Multi dimensional model”, data must be stored following the ”Star schema” as showed in Figure 2.4.

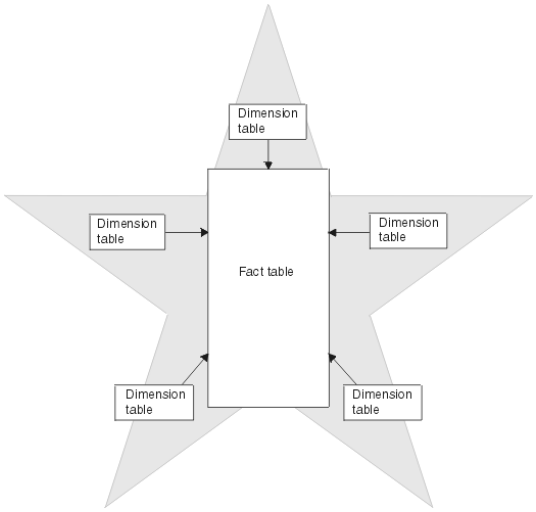

Figure 2.8: Example of star schema

The concept of star schema derives from the five edges of a drwaed star as showed in figure 2.4.

The star schema

The star schema architecture is the simplest data warehouse schema. It is called a star schema because the diagram resembles a star, with points radiating from a center. The center of the star consists of **fact table** and the points of the star are the **dimension tables**. More precisely the Fact table contains the collection of all the Primary key of each dimension table.

Structured Query Language

SQL is the programming language used for retreiving information from databases. Originally based upon relational algebra and tuple relational calculus, SQL consists of a data definition language, data manipulation language, and data control language. The scope of SQL includes data insert, query, update and delete, schema creation and modification, and data access control. Although SQL is often described as, and to a great extent is, a declarative language, it also includes procedural elements.

The operation used to retreive data is called ”query”, which is the use of at least three simple clause which are self-explanatory.

```
SELECT element
FROM table
WHERE a = 1
```

- SELECT
  Is the clause in which we ask the system what column we want to read, if we want to read all columns stored in the table we use the special character ’*’
- FROM
  The FROM clause express which tables must be queried in order to retreive the columns we want
- WHERE
  Is the last clause where we put some conditions.

```
SELECT reference
FROM dna_table
WHERE alternative = 'A'
```

With this simple query we are asking the database to retreive all the reference alleles from dna_table where the alternative allele is ’A’.
This huge amount of data starting to flow from machines to files has arised the needs of technologies capable of managing terabytes of data. Curently data warehousing techniques and technologies begin to be an important part to analyze all this data. In fact combining the power of sequencing with data warehouse structure we open an antire world from analyzing the number of variants per chromosomes to calculate the percent of variant in a population or sub population, even giving the possibility to decide if a variant could be defined rare or not. Unleashing this potential we could bring more decision power to physicians and genetists on diagnose new disease or improve their prevention.

Primary and foreign keys

In database a Primary key consist a collection of data that cannot be repeated and identifies uniquely the row (like an ID). A foreign key is a key used to link two tables together. This is sometimes called a referencing key. Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table.

E.G.

Chromosome	Variant
1	AA
2	TA
3	GG

Table 2.2: Example of Dimension table

Chromosome	Gene
1	ApoE
1	BCRA
2	NIH
3	HBB

Table 2.3: Example of fact table

As you can see the fact table could contain repeated values depending on the primary key of the first table.

- **Fact Table**
  A fact table typically has two types of columns: foreign keys to dimension tables and measures those that contain numeric facts. A fact table can contain fact's data on detail or aggregated level.
- **Dimension Table**
  A dimension is a structure usually composed of one or more hierarchies that categorizes data. If a dimension hasn't got a hierarchies and levels it is called flat dimension or list. The primary keys of each of the dimension tables are part of the composite primary key of the fact table. Dimensional attributes help to describe the dimensional value. They are normally descriptive, textual values. Dimension tables are generally small in size then fact table.

This design will lead to the development of a cube which is our main source of multidimensional queries. The facts of interest are represented in cubes, where:

- each cell of the cube contains numerical measures that quantify the fact
- each axis of the cube represents a dimension of interest for the analysis
- each dimension can be the root of a hierarchy of attributes used to aggregate data.

Dimensions tables are fully denormalized and it is sufficient to perform a join to retrieve all data of a dimension speeding up analysis.

A representation of a DW cube is reported in Figure 2.5.

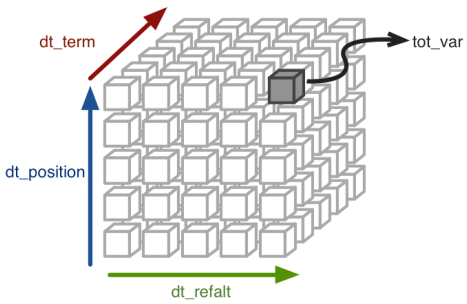

Figure 2.9: Example of OLAP cube

In figure 2.5 we can see a typical representation of a OLAP cube, obviously this example has only 3 dimensions (Time, Product and Location), but typically a OLAP cube has more than 3 dimensions.

- Analysis level
  Level in which we perform all the analyses from counting and reporting to data mining, extracting useful information from DW. Often we use the database which stores the data warehouse with the star schema and OLAP cubes, and the ”staging” database in which we proceeed with all the changes needed in our data.
  Normally in the staging database we use OLTP and in the multidimensional model we use OLAP where we perform calculations such as SUM, AVERAGE, MEAN as mentioned on the beginning of this section.

OLAP is used mainly on data warehouse thanks to its ability to make calculations easy and so creating rich charts were enterprise users can take important decisions. Imagine if you could see the trend of the business and know which area is more profitable, has more potential or is performing poorely. Obviously these calculations can be used on everything is needed like, in our case, the calculation of genotypic and allelic frequency.

The entire data warehouse development could be helped even with an interview step were the final users are interviewed and asked their needs. This phase can be achieved by two ways:

- **Pyramid**: the interviewer starts the conversation with detailed question and at last some general definition. For example the interviewer could start with: ”It is more interesting to see directly the frequencies or general information?”, ending with ”Data warehouse could benefit your business?”

- **Funnel**: this is a more relaxed way to get information starting from general question ending with specific ones.
  For example: ”How do you measure frequencies?”, ”How many samples do you want to analyze?” and ending with ”What kind of representation do you want to see?”

The choice is up to the data warehouse architect in charge of building the system, the developer can also use an hybrid method combining both of them depending on the situation.

2.3.2 OLAP CUBE

An OLAP cube is a data structure that overcomes the limitations of relational databases by providing rapid analysis of data. Cubes can display and sum large amounts of data while also providing users with searchable access to any data points. This way, the data can be rolled up, sliced, and diced as needed to handle the widest variety of questions that are relevant to a users area of interest. The useful feature of an OLAP cube is that the data in the cube can be contained in an aggregated form. To the user, the cube seems to have the answers in advance because assortments of values are already precomputed. Without having to query the source OLAP database, the cube can return answers for a wide range of questions almost instantaneously [21]. A cube can be considered a multi-dimensional generalization of a two- or three-dimensional spreadsheet. For example, a company might want to summarize financial data by product or by time-period to compare budget expenses. Product and time are the data’s dimensions as already described. All the elements of the cube can be organized as hierarchies, a set of parent-child relationship where parent levels typically summarize what children level has. In our data warehouse an implemented hierarchy is well described in section 3.1.1 describing its parts.

Operations

Analyzing the cube as a multidimensional spreadsheets we can develop some useful way of browsing like:

- Slice:
  Is the act of picking a rectangular subset of a cube by choosing a single value for one of its dimensions, creating a new cube with one fewer dimension.

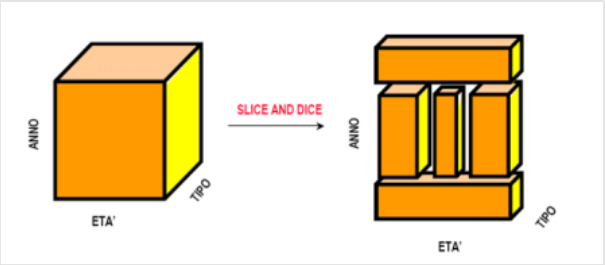

Figure 2.10: A slicing operation

The picture shows a slicing operation where we can observe the sells between 2010 and 2011.

- Dice:
  The dice operation produces a subcube by allowing the analyst to pick specific values of multiple dimensions.[6] The picture shows a dicing operation: The new cube shows the sales figures of a limited number of product categories, the time and region dimensions cover the same range as before.

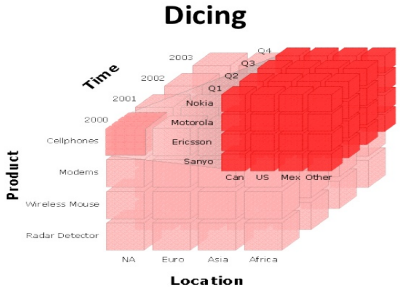

Figure 2.11: A dicing operation

In this figure, we can observe how a sub cube is generated picking up some elements of the cubes creating a new one.

- Drill Down/Up:
  This operation allows the user to navigate among levels of data ranging from the most summarized (up) to the most detailed (down).

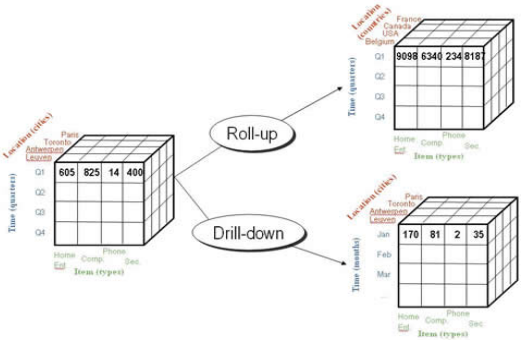

Figure 2.12: A roll up and drill down operation

- Roll-up:
  A roll-up involves summarizing the data along a dimension. The summarization rule might be applying a set of formulas such as ”profit = sales - expenses”

Cube is a shortcut for multidimensional dataset, given that data can have an arbitrary number of dimensions. The term hypercube is sometimes used, especially for data with more than three dimensions. A cube is not a ”cube” in the strict mathematical sense, as all the sides are not necessarily equal.

Security

We can implement different levels of read security in our data warehouse and clearly in our cube, by implementing a table containing read permission dependly by the hierarchy level associated to the account.
E.g. You can develop a system permission hierarchy from ”Boss” to ”Student”, the boss can read all the information stored inside drill-down operations from the Boss leaf to the Student one, in other hand a student-level can read only its level not more. This implementation grant the possibility to have more users using your system, and leveling the read access prevent the reading of your information from not-so-gentle eyes.

Chapter 3

Design considerations

In this chapter we are going to explain step by step the main phases we carried out for designing and developing a data warehouse for manipulating and analyzing genomic data. We focus on optimizing our tools in order to achieve better performances and to solve all the problems we faced during the development of the system. We will discuss the design of the data warehouse, the ETL process, and we will end up with the dashboard components.

3.1 Conceptual design

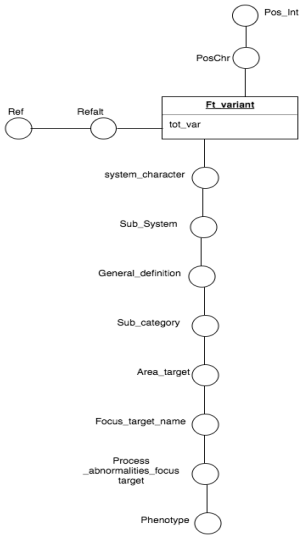

Figure 3.1: Conceptual schema of the data cube

In Figure 3.1 we show the result of conceptual design of the DW using the DFM here further explained:

- ref, refalt are two important items of our system. As first level of the hierarchy we have *refalt* containing all the couple combination of A, T, C, and G. Then as second level we have *ref* being the first character in *refalt*. It gives us the possibility to count,for example, the number of reference allele in a selected chromosome.

- pos_chr & pos_int
  In pos_chr are listed all the sequenced position in a number format associated to the

chromosome in which has been sequenced.
As second level we have *pos_int*, which stores all the variant position ranges on our chromosome. For example, we can look for the number of variants in a particular range of positions.

- Term hierarchy
  Each term in the hierarchy is explained in ”Dimension table Term” Section 3.2.1.

- ft_variant
  In the DFM is the fact table, and the only measure we have in it is ”tot_var”, which count distintly each variant in their relative position. This measure alow us to calculate the allelic and genotypic frequency.

3.2 Fact and Dimensions table design

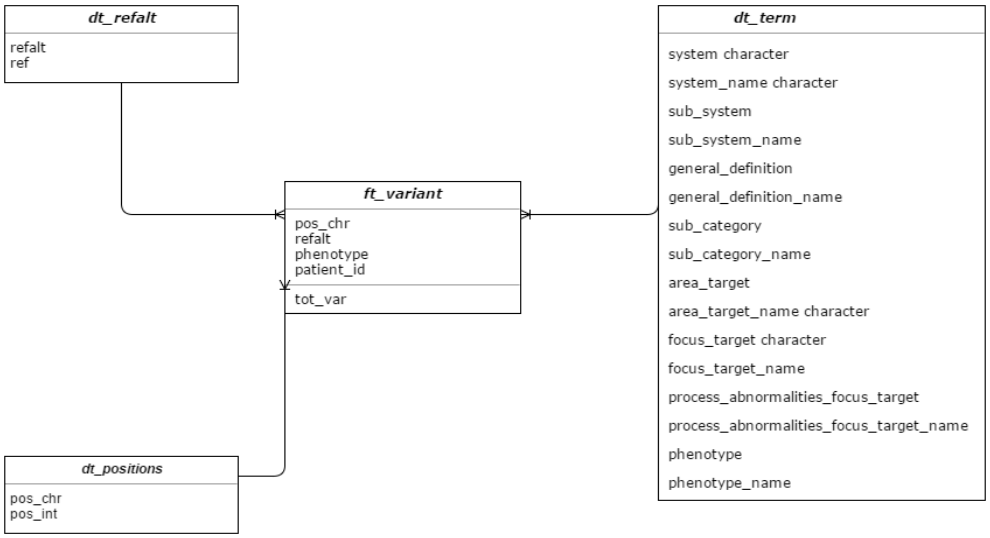

Figure 3.2: The data warehouse main schema

As shown in figure 3.2 the data warehouse is composed in dimensions and fact table here listed:

- dt_refalt
  In this table values ref and alt are uploaded.

- dt_position
  Table dt_position contains two hierarchy levels, the first one is *pos_chr* and the second is *pos_int*.

- dt_term
  This is an important dimension table in which there are all the disease terms in a hierarchy. Further information in section ”Dimension table Term” in section 3.2.1.

- ft_variant
  This is the warehouse main table, containing all the information gathered from the operational database, containing all the variants sequenced and stored. It contains all the *foreign keys* connected with the *primary key* of each dimensions table. As

measure we have *tot_var* which performs a distinct count for every row granting us the possibility to performall the calculation we need or want.

Chosing the star schema as structure fullfill the requirements we meet during the development of the system. For this thesis the main aim is to count each variant called in every DNA string we have sequenced. In order to get a meaningful information of those data we also implemented a system of calculating the frequency for each variant in a specified position. The inteview approach used was a ”Funnel”, because leaves the time needed to the operators to understand the kind of system we are going to build.

3.2.1 Dimension table Term

Table ”dt_terms” is really important because it contains the hierarchy of phenotypic terms. The table has 16 columns containing the id referring to the HPO term, and their respective term name. Every column specifies the level of deepening of the term, from a general system to a single organ.
E.g.

1 All	2 Abnormality of nervous system morphology	57 Morphological abnormality
1 All	3 Abnormality of cardiovascular system	53 Faint(Syncope)

Table 3.1: Schema explaining the dt_terms table where there are few of the couple id/term This table has being populated modifying the hierarchy of *Human phenotype ontology* [22]. At first the terms were stored as a *Directed aciclic graph* (DAG) were a term could have more than one father. This structure is difficult to be implemented in a data warehouse making impossible to retrieve the correct counts on the various operation. For a correct implementation of the dimension we need to transform a DAG in a tree, forcing each term to have only one father. Each term has a rank of similarity in every branch, two terms at the same level has the same degree of similarity to their respective father [26].

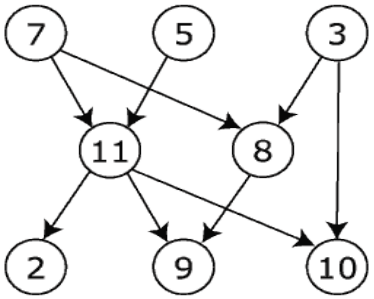

Figure 3.3: Example of DAG

The change of the data structure (from DAG to tree), is needed by the fact that there aren’t databases capable of managing a DAG data structure, so we have to perform some operation in order to use those data.
To create a tree from DAG , we have cut the link between terms. As we can see in the Figure 3.3, term 11 has two parents (5 and 7). In order to obtain a tree we need to cut one of the links. In HPO documentation[26] is explained that we can cut the father with higher difference number (11-5=6, 11-7=4). An high difference number means less similarity between those terms, so we cut the link between 11 and 5. The same operation is done for 9 and 10 cutting respectiively the link with 8 and 3. With this operations we can obtain a simple tree from a DAG.

With a tree we can compute all the calculations we need to perform on our system, a DAG is not supported by all the calculation like SUM,AVERAGE, MIN and MAX. When doing this type of calculations we need to obtain a coherent number of terms when browsing up and down each level of the hierarchy, with a DAG is not possible to achieve this.
In order to explain better what we have done to achieve this goal, here are described all of the SQL operation used.
We create the tree thanks to a recursive SQL query obtaining 15 terms levels:

```
SELECT min(term1_id), term2_id
FROM terms
WHERE  distance=1
```

Listing 3.1: Query for extracting the father-son couple of terms

There are 15 levels because is the maximum number of terms needed to describe a pathology. Only few diseases uses all the 15 levels.
With this query we can populate the table "Temp_table" extracting each couple of terms with a distance of 1, so we obtain a father-son couples.

```
WITH RECURSIVE cte(path) AS (
        SELECT array[r.term1_id, r.term2_id] AS path
        FROM   temp_table r
        LEFT   JOIN temp_table r0 ON r0.term1_id = r.term2_id
        WHERE  r0.term2_id IS NULL
UNION ALL
        SELECT r.term1_id || c.path
        FROM   cte c
        JOIN   temp_table r ON r.term2_id = c.path[1]
),
max_len AS (
    SELECT max(array_length(path, 1)) max_len
    FROM cte
    ),

        SELECT path as col, array_length(path,1) as length
        FROM cte
        CROSS JOIN max_len
        ORDER BY length
```

Then this query recursively fill the array CTE.

```
        SELECT array[r.term1_id, r.term2_id] AS path
        FROM   temp_table r
        LEFT   JOIN temp_table r0 ON r0.term1_id = r.term2_id
        WHERE  r0.term2_id IS NULL
UNION ALL
        SELECT r.term1_id || c.path
        FROM   cte c
        JOIN   temp_table r ON r.term2_id = c.path[1]
),
```

With this query we start from the bottom (NULL) of the hierarchy, creating an array of terms browsing the HPO hierarchy previously depicted in Figure 3.1.

```
max_len AS (
    SELECT max(array_length(path, 1)) max_len
    FROM cte
    ),
```

Listing 3.2: Calculating maximum length of the array

Then we calculate with max_len like in Figure 3.2 , the maximum length of the array dynamically because we can't know how long the array can be. and and then we obtain the final array.
After this operation we have counted the number of terms for each "level", obtaining a gaussian distribution of terms were in the 15th level, we had only 5 terms. We have cut the tree at the 8th level, where the number of terms starts declining, obtaining our final tree of terms.
The cut has been made for performances improvements and to avoid the presence of levels with few terms that are not useful for our purpose, making the hierarchy more usable than before. Clearly as mentioned before the final table will contain a column for each term level, and nerby every single term it's id retreived from HPO table term. For an example see Table 3.1. Each level has being named based on the typology of term contained in it:

Level	HPO term
System	Abnormality of the nervous system
Sub system	Abnormality of nervous system morphology
General definition	Morphological abnormality of the central nervous system
Sub category	Abnormality of brain morphology
Area target	Abnormality of hindbrain morphology
Focus target	Abnormality of the metencephalon
Process abnormalities focus target	Abnormality of the cerebellum
Phenotype	Cerebellar malformation

Table 3.2: Example of a term hierarchy in our dimension table

The table 3.1 contains an example of what we can find after extracting the final hierarchy and the relative possible term for each level.

3.3 ETL design

As explained in the section 2.3, ETL is the main step for populating the warehouse and transforming and modify the data before the upload.

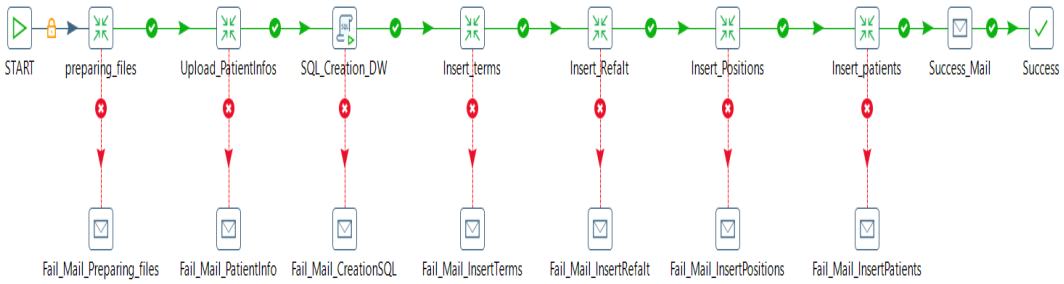

Figure 3.4: ETL process from Pentaho data integration

Figure 3.4 represents the ETL steps that have been used for feeding the DW.
Steps for feeding the DW are reported in figure 3.4:

- Preparing_files
  Process the sequenced file, stored in a *.gvcf.gz* format deleting the header and adding a column containing the patient id, then it will be uploaded on the staging database.
- Upload_patientInfos
  Upload an excel file on the staging database, the file contains all the information about sex, age, phenotype, and family information if there are other family components (father, mother or sons/daughters).
- SQL_creation_DW
  This is the SQL file needed to create the star schema of all the DW
- Insert_terms
  Step needed to upload the hierachical terms into the *dt_terms table*
- Insert_position
  We insert the chr and pos_int data into the *dt_position table*
- Insert_patients
  We load all the variants information in the fact table *ft_variant*

Possible errors occurring in each step and causing the failure of the ETL process are caught and managed by ad-hoc steps where an automatic report about the error is sent by email to the system administrator.
This kind of approach helps the data warehouse architect to understand better all the difficulties behind the development of the system and to fix all the bugs a user or an another developer could face.

We have two different jobs doing quite the same thing:

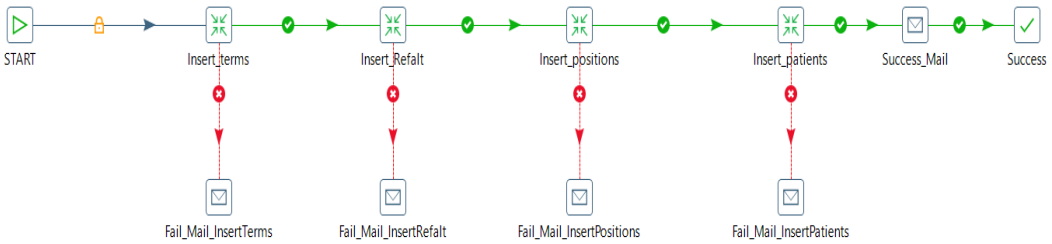

Figure 3.5: Update version of ETL

ETL job described in Figure 3.5 only updates the tables in our data warehouse instead of truncate every time and reupload the data. This brings a better time performances.

Figure 3.6: Example of transformation updating already existing data and adding new ones

Figure 3.7: Example of transformation updating truncating data before wiriting them from scratch

The main differences between Figure 3.6 and Figure 3.7 are that in the first figure we performs an UPDATE in the second one we firstly TRUNCATE the table and then reupload data. As mentioned in the data warehouse development section, in the ETL we have two different ways to upload data, one is to truncate all tables and re-load from scratch all data, a second one is to update the changed rows and adds the new one, this choice depends on how many time new rows are being introduced at every upload, if are performed few changes and new uploads an update system is the best choice avoiding huge re-writing of the same data on the server disks.

3.4 Cube design

After populationg the data warehouse we need to create the cube used for OLAP operations, thanks to Pentaho's Schema Workbench we can achieve this easily.

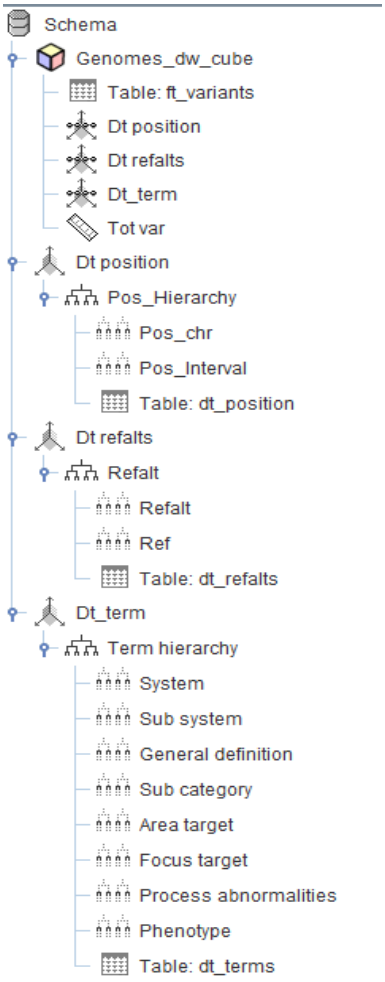

Figure 3.8: Mondrian schema cube

In Figure 3.4 we can see the cube structure of the data warehouse making it easy to query and understand, and overall improving drastically the speed of calculations. The cube respects the foundaments of the star schema in the data warehouse mantaining coherence in the schemas and understandaibility.
The OLAP cube creation process si simple and dynamic we have:

- Cube logical design
  Is the item *Table: ft_variants*, which are described the dimensions used and the measure it contains. We can specify if the measure does an aggregative operation or a COUNT or DISTINCT_COUNT. For our needs a DISTINCT_COUNT is used, this command differs from COUNT because we count distinctly an item instead of counting every item in the set. E.g. In a set we have (A,A,B,C), if we use COUNT the result is 4, with DISTINCT_COUNT the result is 2 A, 1 B and 1C.

- Dimension tables
  Those are *Dt_position*,*DT_refalts*,*Dt_term*, in each dimension table are specified the

relative hierarchy needed for the dimension. The hierarchies can be made simply concatenating one by one every item.

The cube analysis is performed with a Pentaho's plugin named SAIKU which elaborates all the information inside the OLAP cubes thanks to MDX queries. MDX stands for Multi Dimensional eXpression, this kind of queries is created in order get information from the cube.

3.5 Dashboard design

Dashboards are web pages capable of dynamically retrieving data. With different kind of tables and charts we can represent our data giving useful information. The user also can interact with each table and chart, browsing hierarchies and extracting meaningfull information from data.

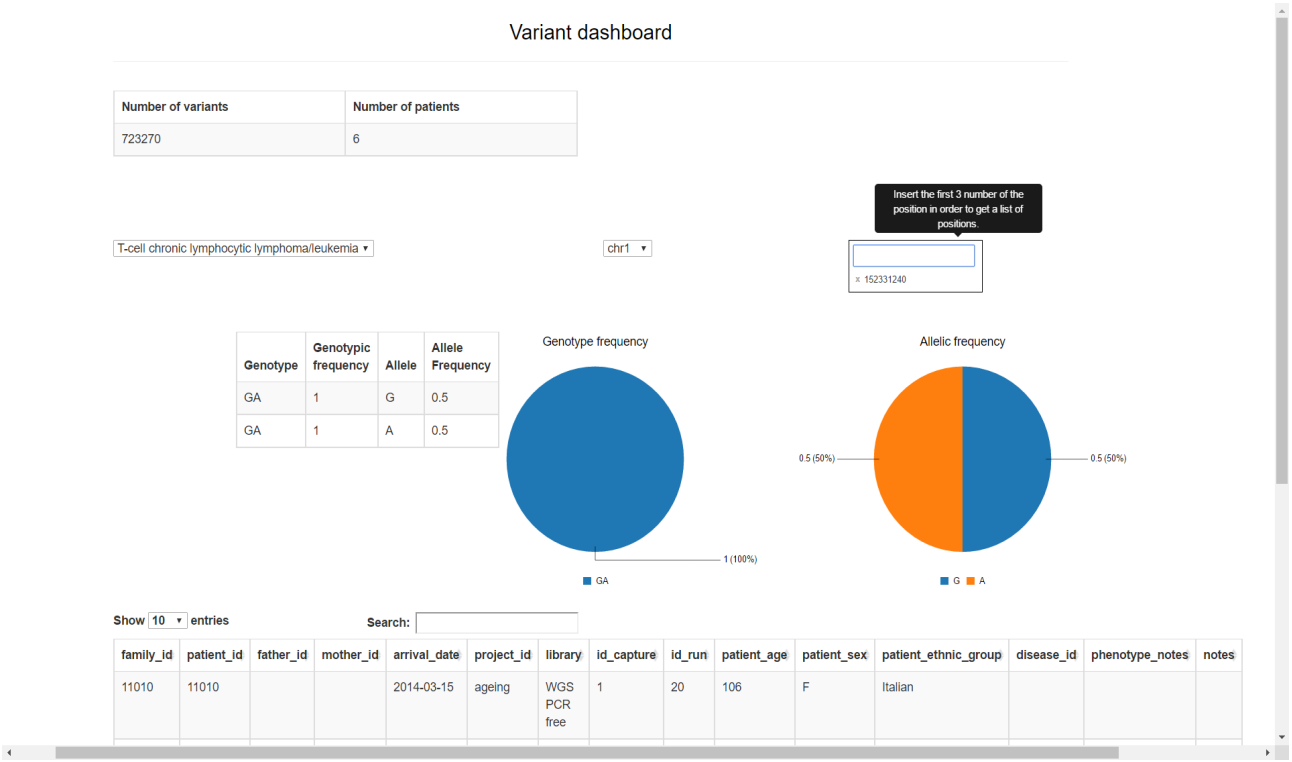

Figure 3.9: Dashboard representation

In Figure 3.5 , the first version of the dashboard is represented. It provides an overview of the database where it can be seen the number of patients in the database and the number of variants collected since now. Moreover the user can select the disease that wants to observe and count the variants specifically for that disease or all diseases.
Selecting all diseases grants the possibility to know the frequency of a variation in the entire population not only selecting a sub-population (people with the selected disease only). The dashboard allows the users to select a chromosome and a position in it, in order to calculate the allelic and genotype frequencies and show them in a pie chart. In the last table are showed all the patients stored in the data warehouse and eventually the family tree , if a father or mother are being sequenced and loaded in the system.

Chapter 4

System description

In this chapter we explain all the tools used to achieve our goal, from the RDBMS used and the main BI software to the bioinformatic tool used for pre-processing data and the configurations used in order to obtain good performances with the RDBMS and the data warehouse.
It is described a simple use case and system validation in order to confirm wether the system is good enough and well developed.

4.1 Tools

Tools used for the creation of the data warehouse are:

- CentOS Linux
- PostgreSQL-9.6
- Pentaho BI community suite
- gVCF tools

4.1.1 CentOS Linux

The main server runs a virtualized Linux distribution CentOs [3], derived from RedHat corporation, and gives to the developer a good environment in which we can find all the libraries needed to run every sotware we need.
The system specifications are:

```
Disk space = 17TB
RAM = 58 GB
CPU = 6 Intel Xeon at 2Ghz
SWAP = 31GB
```

In order to maintain files well organized a directory hierarchy has being created with the structure depicted in Figure 4.1.

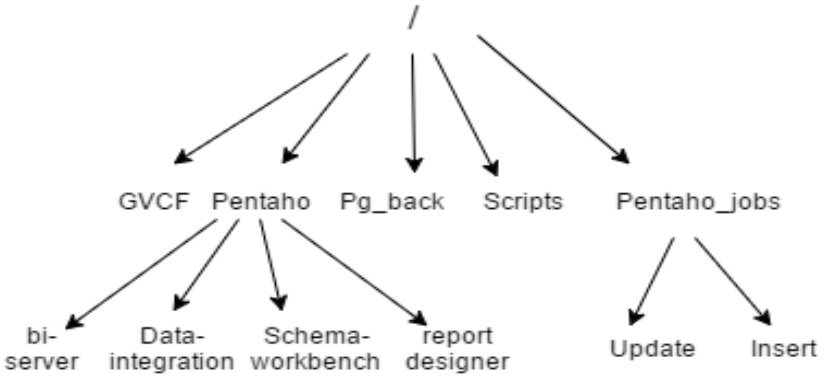

Figure 4.1: Directory structure

- GVCF folder contains the .genome.vcf.gz files;
- Pentaho folder contains all the Pentaho's suite tools;
- Pg_back folder contains the entire database backups;
- Script folder contains all the scripts for automatize the files pre-processing;
- Pentaho_jobs folder contains the job used for feeding the data warehouse.

4.1.2 PostgreSQL-9.6

PostgreSQL[27] is one of the most famous RDBMS, it is open source and highly customizable with a strong community behind it. PostgreSQL has good big data features [8] and performances suited perfectly for our needs. We have also tested the system on MySQL but having poor read performances against PostgreSQL.
For achieving the best performances for PostgreSQL we have configured the software, modifying the *Postgresql.conf* file with those parameters:

- max_connections = 20
  Number of maximum connection allowed incoming to the server
- shared_buffers = 14848MB
  Sets the amount of memory the database server uses for shared memory buffers.
- effective_cache_size = 44544MB
  Sets the planner's assumption about the effective size of the disk cache that is available to a single query.
- work_mem = 380108kB
  Specifies the amount of memory to be used by internal sort operations and hash tables before writing to temporary disk files.
- maintenance_work_mem = 2GB
  Specifies the maximum amount of memory to be used by maintenance operations, such as VACUUM, CREATE INDEX, and ALTER TABLE ADD FOREIGN KEY.

- min_wal_size = 4GB
  Mainimum size to let the WAL grow to between automatic WAL checkpoints.
- max_wal_size = 8GB
  Maximum size to let the WAL grow to between automatic WAL checkpoints.
- checkpoint_completion_target = 0.9
  Specifies the target of checkpoint completion, as a fraction of total time between checkpoints.
- wal_buffers = 16MB
  The amount of shared memory used for WAL data that has not yet been written to disk.
- default_statistics_target = 500
  Sets the default statistics target for table columns without a column-specific target set via ALTER TABLE SET STATISTICS. Larger values increase the time needed to do ANALYZE, but might improve the quality of the planner's estimates.

For a front-end configuration and query test *PgAdmin3* has been used.
Following there are some test query measuring the time needed for some of most used queries.
All the tables have been indexed for all the column needed for our queries. Indexing tables improves a lot overall performances of the database.
How indexing works?
A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional writes and storage space to maintain the index data structure. Indexes are used to quickly locate data without having to search every row in a database table every time a database table is accessed. Its like creating a street map, with the aid of this tool, if you want to locate a store in a city, you know excatly where it is instead of looking around the city. The same happens in a database, improving overall performances looking for a data.
The system has responded perfectly responding in 403ms for the query:

```
SELECT patient_id
FROM ft_variants
GROUP BY patient_id;
```

For the query the system has responded after 1.2sec:

```
SELECT COUNT(*)
FROM ft_variants;
```

And at last the system has responded in 501msec for:

```
SELECT COUNT(refalt), refalt
FROM ft_variants
WHERE split_part(pos_chr, '_',1) = 'chr1'
GROUP BY refalt;
```

This indicates that the system has been succesfully configured for a data warehouse approach for the server's specifications.

4.1.3 Pentaho BI community suite

Pentaho community suite is a Business Intelligence platform giving all the instruments needed to the data warehouse architect to build a stable and reliable system. In order to run correctly Pentaho needs Java7 developed by Oracle.
Pentaho's suite is composed by:

- BI-server
  It is the server running all the Pentaho's components for render the dashboard, connection the cube's publish system and reporting. Pentaho publishing system allows the user to connect Pentaho softwares to the BI-server in order publish all the contents created instead of manually upload files to the server.

Figure 4.2: Pentaho web user interface

Figure 4.2 depicts the main page of the Pentaho server, this can be configured to be reachable from everywhere or to serve customers needs. Here, you can browse all the published files, manage databases connections and cube definitions, and at last browse the dashboard. The server can be configured from remote access or to limit the access for specific IP addresses. The server manager underneath Pentaho-BI is Apache Tomcat [25] making possible to configure every aspect of Pentaho's home page.

- Data-Integration

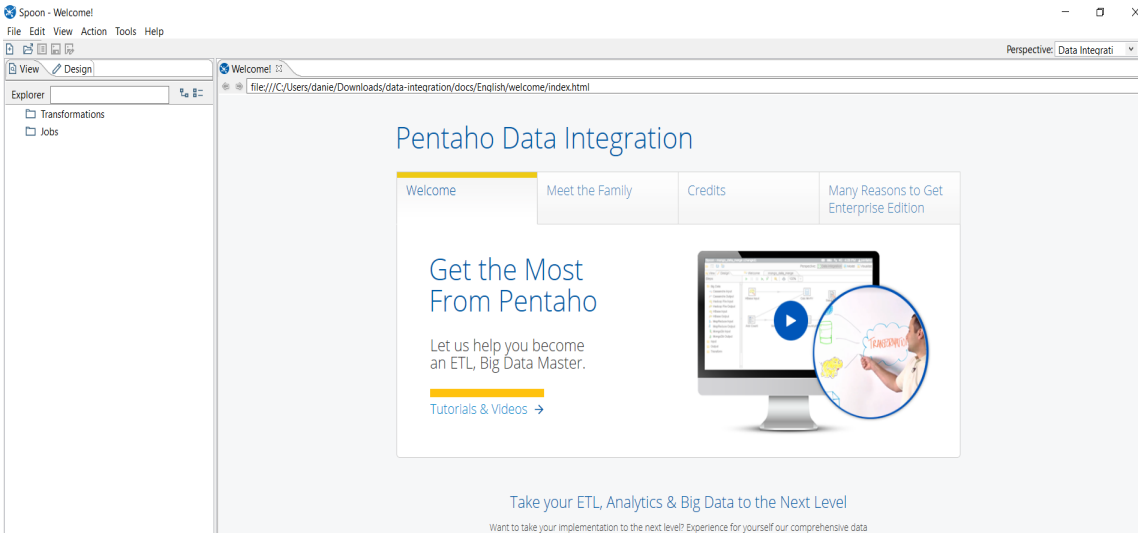

Figure 4.3: Pentaho spoon data integration

In Data Integration, you can interactively create the ETL steps for DW populations, creating connections string with JDBC drivers compatible with your selected RDBMS underneath. Data integration calls *Jobs* the entire ETL schema from start step to success step passing all the *transformations* created in the ETL step as showed in figure 3.5. Pentaho Data Integration has a GUI to start jobs or the single transformation, but has two scripts , witten for both Windows and Unix systems, in order to execute the jobs using the respective system scheduler.
The scripts are **Pan.sh or .bat** for transformations and **Kitchen.sh or .bat** for the entire job process.

- Schema Workbench

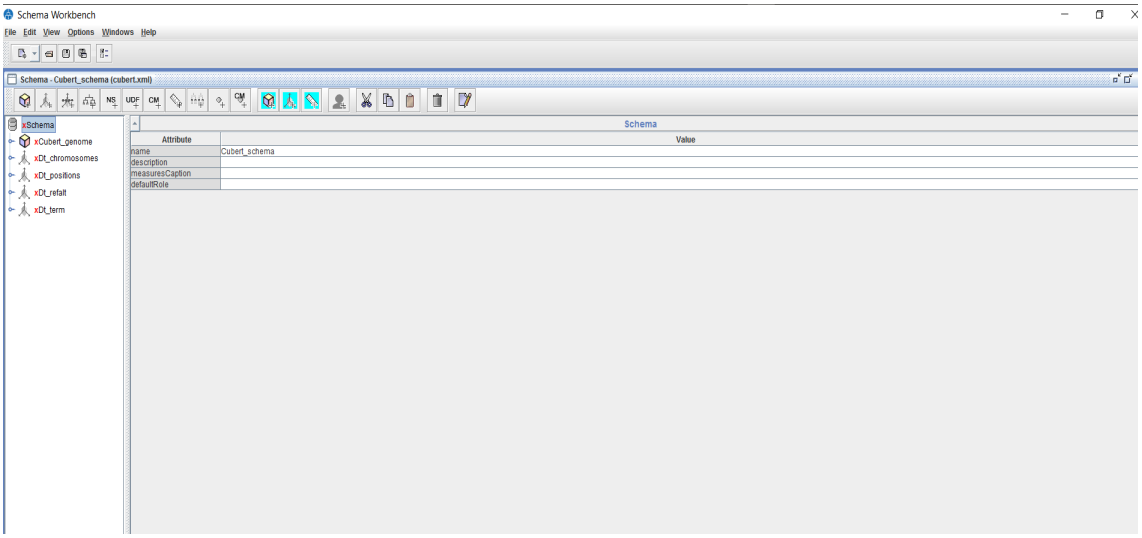

Figure 4.4: Pentaho Schema Workbench

In Schema Workbench you can create your own cube of interest and publish it directly into BI-Server specifying the repository destination. With Mondrian, an open source Online Analytical Processing server (OLAP) written in JAVA, the system responds to queries fast enough to allow an interactive exploration of the data. It brings multidimensional analysis to users allowing examination of business data by drilling and cross-tabulating information [24]. In Pentaho a mondrian cube is developed via GUI, but the software allows the user to modify the source code which is a XML file. Schema workbench allows to configure the mondrian cube and to tune perfectly the settings, also you can modify directly the *.xml* file for more specific settings.

- Report Designer

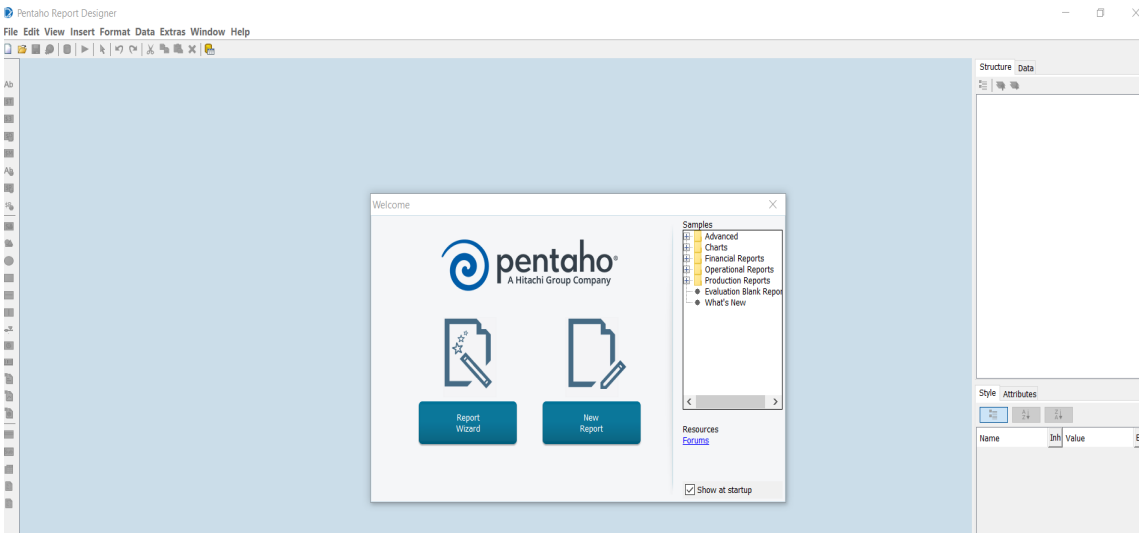

Figure 4.5: Pentaho Report Designer

This tool aims to create a rich PDF document reporting all the data queried from the cube or the database. You are able to personalize the report suiting the business brand needs.

All the Pentaho's tools are multi platform as they only requires Java7, and you can configure each component to connect to your system.

4.1.4 gVCF tools

This is the main tool used to pre-process the gVCF files obtaining the one needed for our use case. The tool is composed in:

- extract_variants
  Removes all non-variant blocks from a gVCF file to produce a smaller variant-only VCF file.
- create_bed
  Creates a *.bed* file from the previous step and this file is used in the next step.
  BED (Browser Extensible Data) format provides a flexible way to define the data lines that are displayed in an annotation track.
- break_blocks
  Breaks non-variant gVCF blocks into individual positions in all regions of a specified BED file.
  - Optionally, it writes out full gVCF with broken blocks, or broken blocks only
  - Optionally, it include all variants when writing broken blocks only

This system has been completely automatized with python and bash scripting, creating some simple scripts, and scheduling with the Linux system the automatic activation of scripts overnight.

4.2 Use cases

In this section is described a simple use case that the user is going to face.

- **Use case**:
  Know the genotypic and allelic frequency of a variant knowing the chromosome and position
- **Actors**:
  Genetist with a registered account on the system
- **Triggers**:
  Select the desired chromosome and insert at least the first 3 digits
- **Pre-conditions**:
  User has selected the chromosome and the position to look for
- **Post-conditions**:
  The system gives the response with two charts containing the information about the allelic and genotypic frequencies.

Normal flow

- The user indicates the chromosome that wants
- The system will calculate all available position depending on the selected chromosome

- The user selects the desired position inserting at least the first 3 digits on the text box
- The system will calculate the frequencies and will present to the user the response with two updated charts with genotypic and allelic frequencies.

4.3 System validation

A system validation can be conducted in two ways, first one is how the system is usable by majority of potential users how many errors the interviewers have done testing the system and if it is sufficiently intuitive and second is how portable the system is, which means if a server upgrade is needed and how difficult could be to achieve a transfer.
Those are a good metric to take in consideration if you need a dynamic scene achieving the needs of a fast changing needs.

4.3.1 System transfer

A system transfer means the transferring of the entire data warehouse system from a server to an other, for example for upgrading disk space or using a brand new machine developed entirelly for a business analysis system.
This data warehouse system is based on open source technologies which are available on Mac, Linux and Windows system for both PostgreSQL and Pentaho BI suite, even the configuration files does not change inside their structure so those are fully portable from a system to other.
The only difficulty that a system change could face are the **gVCF** bioinformatic tools which are available only for Unix systems making the pre-processing system more difficult on a Windows environment.

4.3.2 System testing

The system has been tested by different kind of users, from non tech people to physicians, in specific:

- 1 Physician (Cardiologist)
- 3 Students
- 1 Worker (Full stack web developer)

All the testers has been asked to:

- Navigate and explain the main page *(See Figure 3.9)*
- Try a simple search with no clues
- Exaplain if what has been showed is what the expected to find

Overall the test has been passed by all the users where the only difficulty was to navigate trough the main page, all of them in fact have found some disappointing position of the search bars.
All the testers were able to navigate and explain what the main page is offering to them,

they have described the page as ”Simple and effective”
For the search task only one student was not able to complete the search task, the search bar for Position was not sufficiently self explanatory so it was not possible to complete the task but all the other testers has completed the task with no significant difficulties.
For the third task only two student and the physician were able to explain perfectly what the results showed means in a biological manner, and the worker was able to understand what they means and the aim of the search but not the biological meaning.

4.3.3 Example of dashboard browsing

In this section we will show a simple dashboard browse.
We want to get the frequency of a particular variant positioned on a selected chromosome.

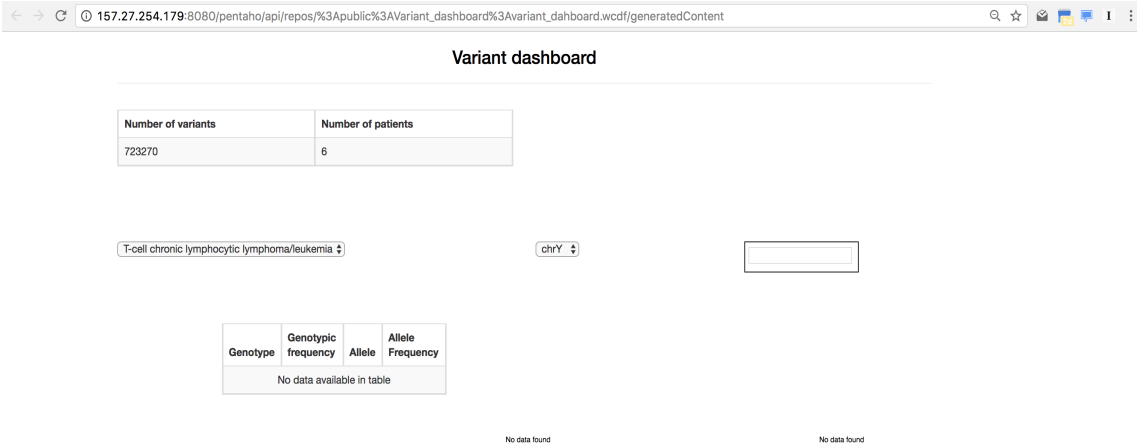

Figure 4.6: Dashboard main page

Figure 4.6 depicts the main page of the dashboard, where we start our search.

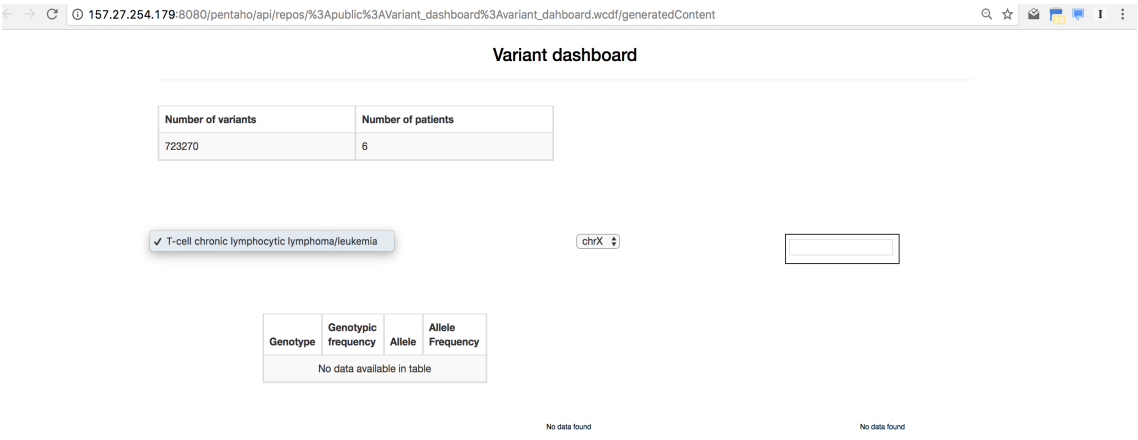

Figure 4.7: Selecting disease

Initially we select the desired diseases and the chromosome from the drop down menus.
In the drop down menu are listed all the available chromosomes in the desired disease.

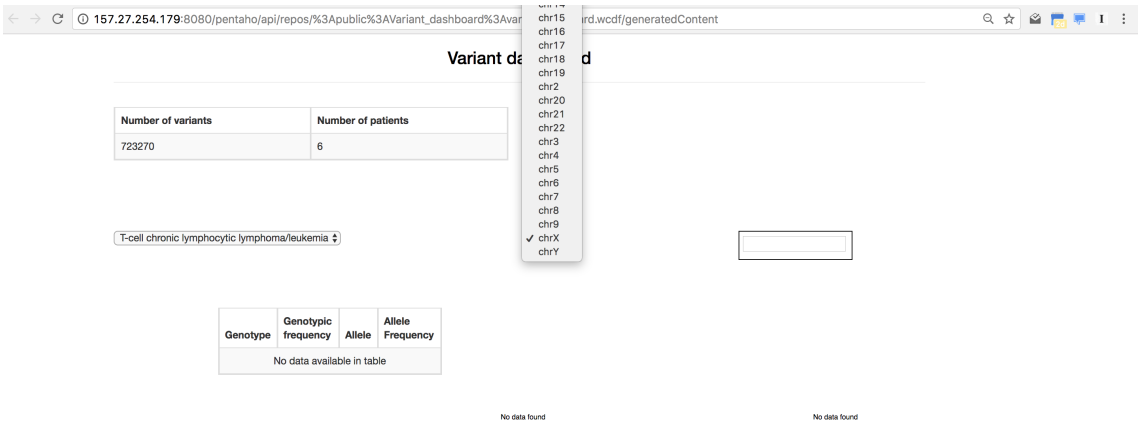

Figure 4.8: Chromosome selection

After the chromosome selection we star writing the first 3 numbers of the position desired. Even if we don't know exactly what is the position number of a variant, the system gives you all the possible position containing the numbers inserted in the textbox.

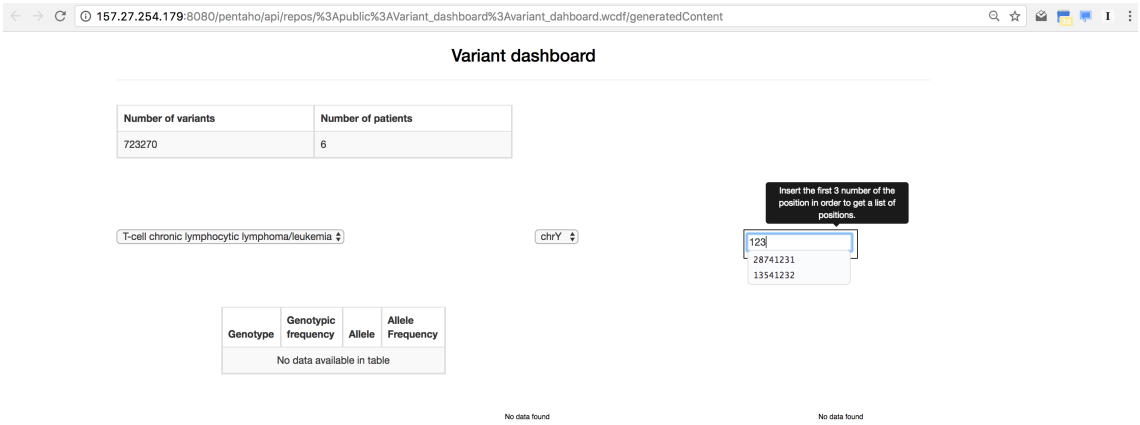

Figure 4.9: Selecting position

Once we have selected the position the system starts to calculate the genotypic and allelic frequency shown in figure 4.10.

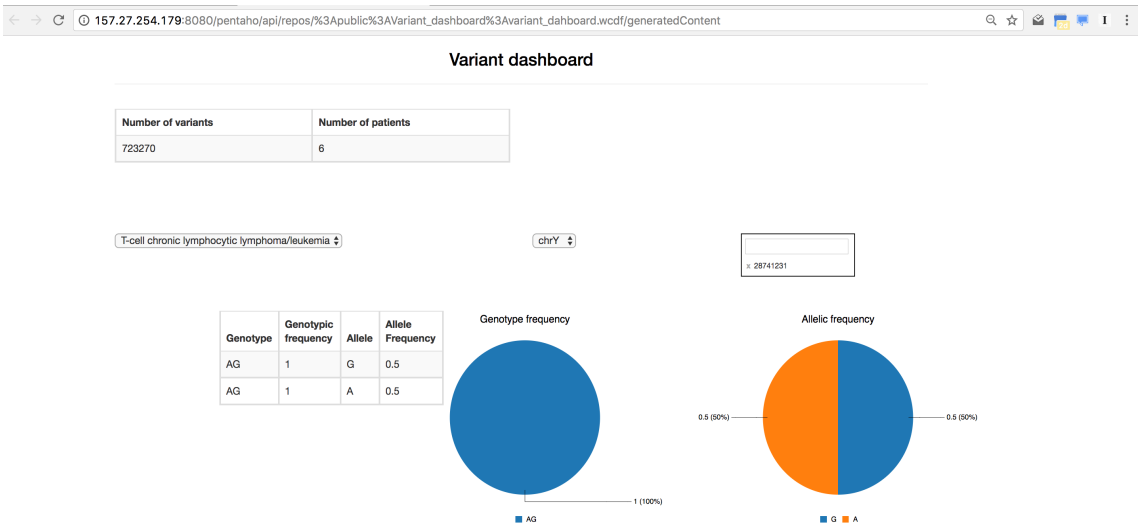

Figure 4.10: Final result

Fiure 4.10 shows all the information we were looking for. We can observe the results via three tools:

- A simple table containing the information searched
- Two pie charts, one for allelic frequency and one for genotypic frequency.

4.3.4 Example of cube browsing with SAIKU

Thanks to SAIKU, a Pentaho plugin for cube browsing, we perfrom some heavy queries easily. Simply via drag and drop we can look for the information we need.
In this example we want to know the number of variant we have for each kind (AT,AA,AC,AG and so on)
First we select the cube and then via drag and drop we create the query

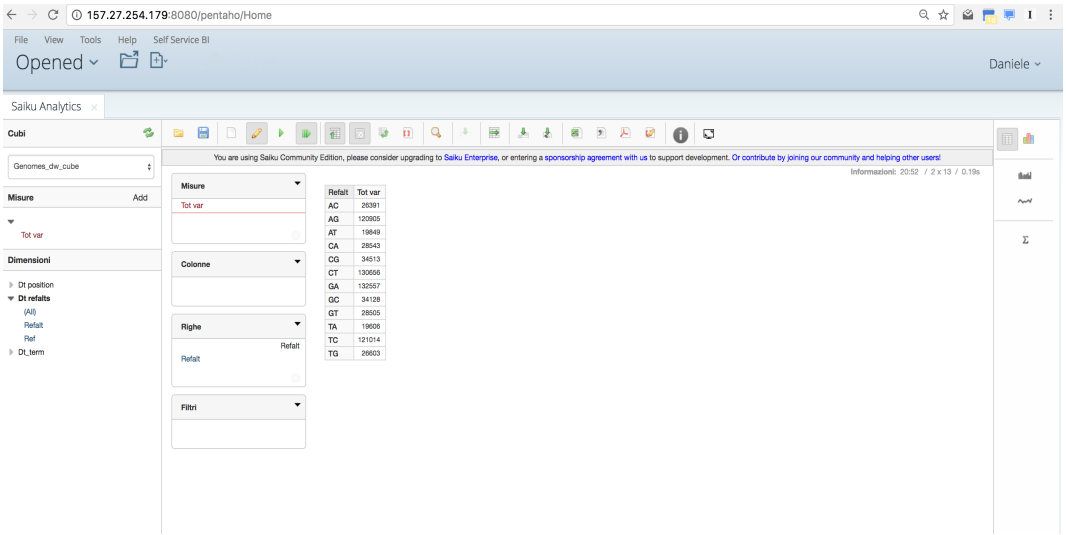

Figure 4.11: SAIKU start page

Figure 4.11 depicts the result we have after quering the system.
We can also select different kind of charts creating rich and complete information.

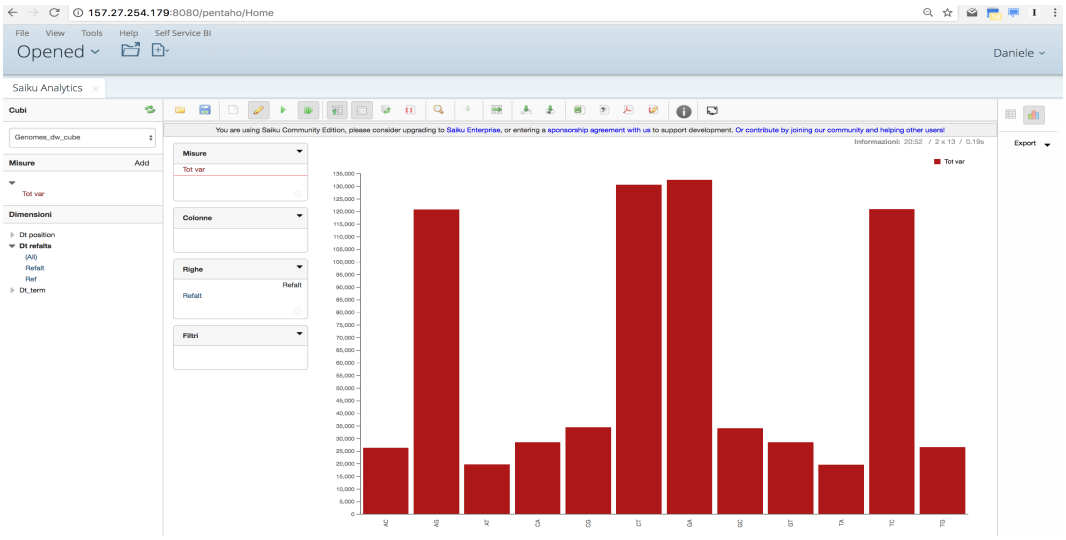

Figure 4.12: Barchart

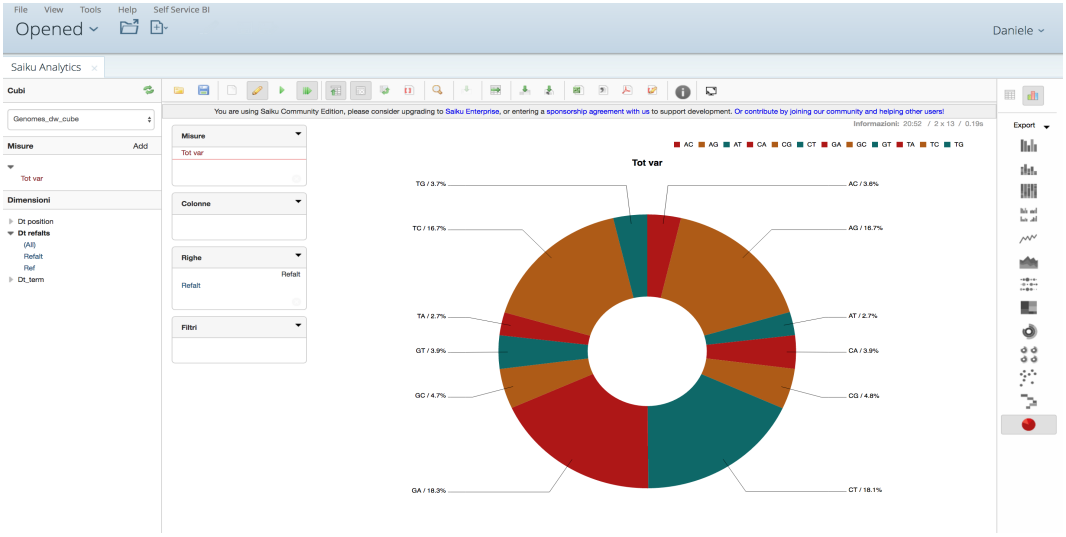

Figure 4.13: Piechart

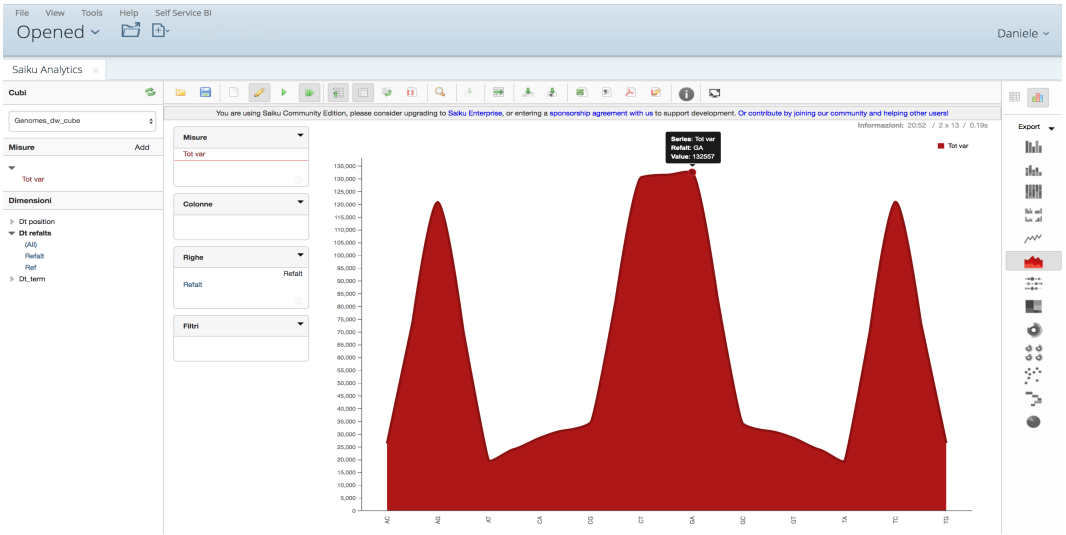

Figure 4.14: Series chart

The above figures are some of the available charts in SAIKU, other charts are available depending on the analysis. This kind of query take a few second to give the result we want even if there are a lot of data inside it. The cube logical structure bring faster queries compared to classic SQL with the same amount of data.

Chapter 5

Conclusions

In this chapter are described what are the future challenges of the project, what will be the future development and possible evolutions of the system.
Also are described the thoughts of the developer and if developing this system is a useful challenge.

5.1 Personal thoughts

In this thesis we have reached our goal of developing a stable, fully functional data warehouse. This system is capable to create, read and import standard genetic files, feed the warehouse level thanks to the ETL step, to calculate genotipic and allelic frequencies and show them with a dashboard.
Developing a warehouse system opens a whole world in data analysis. This could be really interesting when more and more operational genomic databases becomes available, in order to use them as source of analysis for genomic data. This knowledge and data availability combined with a growing computational power grants the developing of powerful tools for data mining and discovering new knowledge for mankind. Difficulties are few and really hard to overcome sometimes:

- Disk read performances
  This can be a really bad bottleneck for every system, Solid State Disks are not ready for a massive deployment so performances with HDD are really compromised with tons of data. A solution can be the use of NO-SQL databases where data is queried as a document, partitioning big files into a multi smaller files speeding up the searches. Another solution is using RAID systems dividing the operations load across multi disks and multiple servers.

- Softwares
  Only nowadays softwares are being developed to face the needs of a warehousing system much of the open source softwares are not ”Warehouse” ready. A solution is using GPUs power instead of the CPU [6]. Using GPU power instead of CPU brings more power thanks to the high number of microprocessors stored in a video card like an Nvidia or ATI.

- MDX queries
  MDX queries are not user-friendly and sometimes could be really difficult to develop a good query. In the last summit Microsoft said that MDX will receive two

or three more releases then it will be only supported not more developed, leaving the space for a in-memory approach thanks to more and more space being available [5].

Knowing a powerful system like data warehouse allows to achieve endless possibilities for OLAP searches, reporting and data mining, obtaining more and more information to interpret and modify.

5.2 Future developments

A data warehouse system has a infinite potential thanks to it's modular nature in where you can add and develop many cubes for each analisys is needed, integrate other services in the BI server and develop SQL algorithms that brings more powerful data analisys and faster queries.
The system described in this thesis is a stable "1.0" version which means that is ready for a "Production" environment but lacking of some, maybe, useful features such as:

- A reporting system capable of creating useful rich reports, where you can add all the disease information.
- More automatized processes in order to make every upload simpler and efficient
- Open sourcing the ETL steps and queries in order to take advantage of the Open Source community and create a standard in the Genomic scientific community for data warehouse development.
- Propose a standard for the hierarchical system of Pathology names (see *"Dimension table Term"* section) for a data warehouse system in the direction of making data warehousing more used in those sceneries.
- Create an A.I. system in order to develop better predictive diagnosis system.

Those are some of the future developments needed to bring a more complete experience of a data warehouse system in a genomic environment. With a data warehouse in genomics there are potentially infinite applications, from diagnosis to gene discovery even creating and proposing new standards.

References

[1] Wilhelm J Ansorge. “Next Generation DNA Sequencing (II): Techniques, Applications”. In: *Journal of Next Generation Sequencing and Applications* (2016).

[2] A. Cao. “Thalassaemia types and their incidence in Sardinia.” In: *Journal of medical genetics* (1978).

[3] CentOS. *https://www.centos.org/about/*. 2016.

[4] Hsinchun Chen. “”BUSINESS INTELLIGENCE AND ANALYTICS: FROM BIG DATA TO BIG IMPACT””. In: ().

[5] Microsoft Data and Analytics. *”Will SSAS, Cubes and MDX be abandoned because of the BI Semantic Model?”*

[6] Gaurav P. Dave. “Utilizing Power of Graphics Processors in Data Processing”. In: *International Journal of Advances in Management, Technology & Engineering Sciences* (2012).

[7] Brent Ewing and Phil Green. “Base-Calling of Automated Sequencer Traces Using Phred. Error Probabilities”. In: *Genome research* (1998).

[8] Mathieu Le Faucheur. *”Is Your Database Ready for Big Data?”*

[9] Abram Gabriel. “Sickle Cell Anemia: A Look at Global Haplotype Distribution”. In: *Nature* (2010).

[10] John H Gillespie. “Population genetics : a concise guide”. In: *The Johns Hopkins University Press* (2004).

[11] Khalid Adam Ismail Hammad. “Big Data Analysis and Storage”. In: *International Conference on Operations Excellence and Service Engineering* (2015).

[12] Hardy. “Mendelian proportions in a mixed population”. In: *Science* (2011), pp. 49–50.

[13] National human genome research instituter. “The Cost of Sequencing a Human Genome”. In: *NHGRI* (2016).

[14] Clyde A. Hutchison. “DNA sequencing: bench to bedside and beyond”. In: *Nucleic acids research* (2007), pp. 1–2.

[15] Illumina. *Sequencing by Synthesis (SBS) Technology*.

[16] Barbara Jennings. “A vision of personalised medicine revealed through modern genomics and open science”. In: *European Journal of Human Genetics (2014)* (2014).

[17] Sang-Woo Jun. “BlueDBM: An Appliance for Big Data Analytics”. In: *Department of Electrical Engineering and Computer Science Massachusetts Institute of Technology* (2016).

[18] Bruce R Korf. “Genetics in medical practice”. In: *Nature* (2002).

[19] Jake Luoi. “Big Data Application in Biomedical Research and Health Care: A Literature Review”. In: *Biomedical Informatics Insights* (2016).

[20] Mr. Dishek Mankad. ““The Study on Data Warehouse Design and Usage””. In: *International Journal of Scientific and Research Publications* ().

[21] Tapio Niemi. “Constructing OLAP cubes based on queries”. In: *DOLAP ́01 Proceedings of the 4th ACM international workshop on Data warehousing and OLAP* (2001).

[22] Giorgio Valentini Sebastian Kohler Matteo Re Marco Notaro and Peter N. Robinson. “Prediction of human gene - phenotypeassociations by exploiting the hierarchicalstructure of the Human Phenotype Ontology”. In: ().

[23] Michael J. Paul. “Twitter Improves Influenza Forecasting”. In: *Plos* (2014).

[24] Pentaho. *”http://community.pentaho.com/projects/mondrian/”*.

[25] Pentaho. *”https://help.pentaho.com/Documentation/5.3/0H0/060/010/000”*.

[26] Denise Horn Peter N. Robinson Sebastian Kohler Sebastian Bauer Dominik Seelow and Stefan Mundlos. “The Human Phenotype Ontology: A Tool for Annotating and Analyzing Human Hereditary Disease”. In: *The American Journal of Human Genetics* (2008), p. 4.

[27] PostgreSQL. *https://www.postgresql.org/docs/9.6/static/index.html*. 2016.

[28] Brooker R. “Biology”. In: (2011).

[29] Luna Rajbhandari. “Data Warehouse and Analytics Discussion Paper”. In: *NORTHWESTERN UNIVERSITY* (2013).

[30] Jason A. Reuter. “High Throughput Sequencing Technologies”. In: *Molecular Cell* (2015).

[31] Strand Life Sciences. “NGS Data Storage Requirements”. In: *http://www.strand-ngs.com/support/ngs-data-storage-requirements* (2016).

[32] Sheila. “Phred-scaled Quality Scores”. In: *GATK BROAD INSTITUTE* (June 2014).

[33] Cecie Starr. “The unity and diversity of life”. In: *Biology* (2004).

[34] Grand view research. *Next Generation Sequencing Market To Reach 27.8 Billion By 2022*.