Organization of Variation Based Personal Genetic Data with Relational Database

Araştırma Makalesi/Research Article

Onur ÇAKIRGÖZ[1], Süleyman SEVİNÇ[2]

[1]Computer Engineering Department, Bartın University, Bartın, Turkey
[2]Computer Engineering Department, Dokuz Eylül University, İzmir, Turkey
onurcakirgoz@bartin.edu.tr, suleyman.sevinc@cs.deu.edu.tr
(Geliş/Received:20.04.2018; Kabul/Accepted:30.07.2018)
DOI: 10.17671/gazibtd.417443

***Abstract*—** Relational databases are currently being used effectively in many hospitals and clinics to store patient records and assay results. With the rapid development of sequencing technologies, sequencing costs have declined considerably. In addition, the number of personalized medicine practices is increasing day by day, and accordingly the size of the personal genetic data that needs to be stored and questioned is also increasing. Although relational databases are appropriate for storing patient records and assay results, additional designs and solutions are needed to efficiently store personal genetic data. In this study, a novel solution is proposed for the integration of variation-based personal genetic data into relational database. Within the scope of this solution, formats for both non-structural and structural variation types have been developed and compression algorithms have been used. The proposed method was tested with real data of 2504 people, published by 1000 Genome Project. Compared to the space required to store the raw sequence data, it was seen that the proposed method yielded a space gain of 99.74%.

***Keywords*—** relational database, data format, genetic data, variation

Varyasyon Bazlı Kişisel Genetik Verilerin İlişkisel Veritabanı ile Organizasyonu

***Özet*—** İlişkisel veritabanları halihazırda birçok hastanede ve klinikte hasta kayıtlarını ve tahlil sonuçlarını depolamak için etkin bir şekilde kullanılmaya devam etmektedir. Sekanslama teknolojilerinin gelişmesiyle birlikte sekanslama maliyetleri önemli bir ölçüde düşmüştür. Bunun yanında, kişiselleştirilmiş tıp uygulamalarının sayısı her geçen gün artmaktadır ve buna bağlı olarak depolanması ve sorgulanması gereken kişisel genetik verilerin boyutu da yükselmektedir. Her ne kadar ilişkisel veritabanları hasta kayıtlarını ve tahlil sonuçlarını depolamak için uygun olsa da kişisel genetik verilerin verimli bir şekilde depolanması için ek tasarımlara ve çözümlere ihtiyaç vardır. Bu çalışmada, varyasyon bazlı kişisel genetik verilerin ilişkisel veritabanına entegrasyonu için yeni bir çözüm önerilmektedir. Bu çözüm kapsamında, hem yapısal olmayan hem de yapısal varyasyon tipleri için formatlar geliştirilmiştir ve sıkıştırma algoritmaları kullanılmıştır. Önerilen yöntem 1000 Genom Projesi'nin yayınlamış olduğu 2504 kişiye ait gerçek veriler ile test edilmiştir. Ham sekans verilerinin depolanması için gerekli olan alan ile karşılaştırıldığında, önerilen yöntemin % 99,74'lük bir alan kazancı sağladığı görülmüştür.

***Anahtar Kelimeler*—** ilişkisel veritabanı, veri formatı, genetik veri, varyasyon

1. INTRODUCTION

The desire of human being to recognize itself, its environment, nature and the universe has been enduring since the beginning of human history. This human desire for discovering and recognizing has caused numerous investigations and scientific studies in numerous fields, and as a result of these efforts, important turning points in the history of mankind have taken place. When we evaluate the situation in terms of bioinformatics field, the discovery of the cell, the discovery of DNA, the understanding of the working principle of the cell, and immediately after them the actualization of the human genome project [1] are considered important turning points. In particular, the human genome project has paved the way for countless scientific researches that will enable important discoveries and information to be released in terms of human life and human health. At this point, the international HapMap project [2] and 1000 genome project [3-6], which catalog the genetic variations, must be put in a separate place. Both

these enormous projects, as well as countless scientific researches carried out before or after these projects, have made it possible for human beings to recognize their own biology and properties (relationship between genetic and diseases) to a very large extent [7-11]. In addition to this, the boundaries of traditional medicine have been overcome and personalized medicine concept and approach has entered into our lives [12-15]. For a particular disease, traditional medicine applies the same treatment to everyone and gives everyone the same drug. Unlike traditional medicine, personalized medicine has adopted the person-specific treatment and the person-specific drug approaches. At this point, the most important mainstay of personalized medicine is genetic characteristics and genetic variations [12]. Therefore, it is necessary to identify the genetic factors that cause diseases and that reveal physical and behavioral similarities / differences between individuals. For this reason, the need for the storage and processing of genetic data of a large number of people has arisen.

Relational databases have been used effectively to store and query data in numerous fields since the day they were first created [36-37]. An important one of these fields is medicine [38]. In hospitals and clinics, relational databases have been used to record and query patient records and laboratory results and are still in use today. Besides, the size of the genetic data produced is increasing day by day because of the reasons such that the development of sequencing technology at a dizzying pace [16], correspondingly, the reduction of sequencing costs, understanding the effects of genetic characteristics on human health and disease, the acceleration of personalized medicine, the anticipation that preventive medicine will greatly reduce the general health expenditures. On the other hand, sequencing devices often produce raw sequence data in the form of files, usually in fasta [17] or in their own format. It is very difficult to organize and manage personal genetic data in this way. Even, as the number of people increases, it becomes impossible. In this kind of approach, clinicians and researchers have to cope with a large number of files and folders. Apart from that, variations and genotypes/haplotypes are very significant in personalized medicine. Naturally, clinicians obtain the variations from the raw sequence data and usually keep these variation data in excel spreadsheets. On the one hand, the patient's records and laboratory results are stored in the relational database, on the other hand the patient's personal variation information is stored in the form of an excel table. Therefore, it is necessary to transfer personal genetic data to the relational database in order to store and manage personal genetic data in a structured and systematic manner, and to easily evaluate genetic data together with other information about the patient held in the database.

As is known, human DNA consists of approximately 3.2 billion base pairs and chromosomes are in pairs. From a computer science perspective, this data corresponds to a string of approximately 3.2 billion lengths. Assuming that we represent each base with 1 byte and that the pairs of chromosomes are identical, the space occupied by this data is approximately 3.2 GB. On the other hand, this data is the raw sequence data produced by the sequencing device, that is, the unprocessed data. As we have already mentioned, the practices of personalized medicine mainly use personal variation data (genotype). Sequence alignment algorithms are utilized to obtain variations from the raw sequence data, but the execution of these algorithms takes a long time [18-20]. The use of variation data in personalized medicine applications and the necessity of using costly sequence alignment algorithms to obtain this data has led to the necessity of organizing variation-based personal genetic data in a structured way. The logic here is: Instead of using sequence alignment algorithms each time to obtain variations, identifying the variations (genotypes/haplotypes) once and store them permanently. The crucial question that needs to be asked at this point is: What is the average number of variations detected in a person's entire genome and what is the average length of those variations? The answer to this question will demonstrate the feasibility and efficiency of the proposed method. In a study [21] conducted using the variation data published by the 1000 genome project, it was determined that the average number of genotypes/haplotypes in the whole genome of a person is approximately 4.5 million. In addition, in the same study, it was determined that the average length of these variations was below 3 bases. These results have shown that it is a very accurate and logical decision to store personal genetic data in a variation-based manner.

2. RELATED WORK

When we consider human genome from computational perspective, we will see a string made up of letters A,G,C,T. Since Dna sequence is a string consisting of letters A, G, C, T, researches have intensively used textual data compression techniques to reduce the huge storage costs. Textual data compression techniques are basically classified under four main headings: Substitutional-Statistical methods, Grammar-based methods, Transformational methods, and Table compression methods. Among these methods, Substitutional-Statistical methods have showed the greatest improvements. The basic logic of this class of methods is, as the name implies, combining the substitutional techniques and statistical techniques in order to improve the compression ratio. Biocompress 1 [22] is the first example of Substitutional-Statistical methods. Later, many studies were published. Among them, the well-known studies are DNACompress [23] and DNAPack [24]. Although these studies combine the substitutional and statistical techniques to improve the compression performance, XM [25] a pure statistical compression method, yielded better results than DNAPack, the best performing of the Substitutional-Statistical methods that we investigated. If XM is used to compress the human genome, approximately 1,20 GB and 1,18 GB will be sufficient to store female genome and male genome, respectively.

The 1000 Genomes Project published the variation data of 2504 anonymous people as VCF [26] and BCF [27] files.

The VCF (Variant Call Format) format [28] is a tab delimited text file format for storing variations and individual genotypes. Although VCF is widely used in the community, it has substantial drawbacks. Because the file is text, it requires a lot of space on disk and is excessively slow to parse. BCF is a binary, compressed equivalent of VCF. A BCF file is composed of a series of compressed blocks of binary records. BCF files are faster and take up less space compared to VCF files. Because these two file formats are equivalent, both have some common shortcomings. While storing the genetic data of many people; low allele frequencies lead to so much redundant space usage. These formats don't provide any structure to divide the chromosome into regions and to store the data in this way. Finally, to add the variation-based genetic data of a new person to the file, the file needs to be updated from beginning to end.

Structural approaches for storing and querying genomic data are Genomics Algebra [29], the Phd thesis of Tata [30] and GQL [31]. J. Hammer and M. Schneider have proposed an integrating approach that is based on two fundamental structures. These are genomics algebra and genomics functions. While genomic algebra consists of genomic data types (e.g., genome, gene, protein, nucleotide), genomic functions consist of functions (e.g., translate, transcribe). They propose extending SQL by embedding these two structures into it. However, in order to extend SQL, data structures that will be running in the background should be created and added to the system. For instance, to add a new function such as get_variations( ) to SQL, necessary data structure should be added to the system too. The second and the third studies, the Phd thesis of Tata and GQL, are very similar to the first study. They are also based on genome query algebra and use a standard SQL-like syntax. In fact, the main difference between the first study and others is that while the first study recommends making additions to existing database management systems, other studies recommend constructing the system from scratch. Naturally, they have also data-structures requirement. These three studies have other common shortcomings too. The space requirements of the methods were not calculated. The methods were not tested on real personal genetic data.

As mentioned earlier, within this study, the variation data published by the 1000 genome project has been used. This dataset contains whole-genome genetic data of 2504 individuals selected from different populations and is open to the use of researchers. This dataset has also been used in many scientific studies for different purposes [32-35].

3. RELATIONAL DATABASE SCHEMA

The relational database schema, designed to hold variation-based personal genetic data in relational database, appears in Figure 1. As shown in Figure 1, the relational database consists of four tables. The names of these four tables are “Records”, “Reference_Chromosomes”, “Individuals” and “Variations”. Each of these tables will be discussed separately in the following sections.

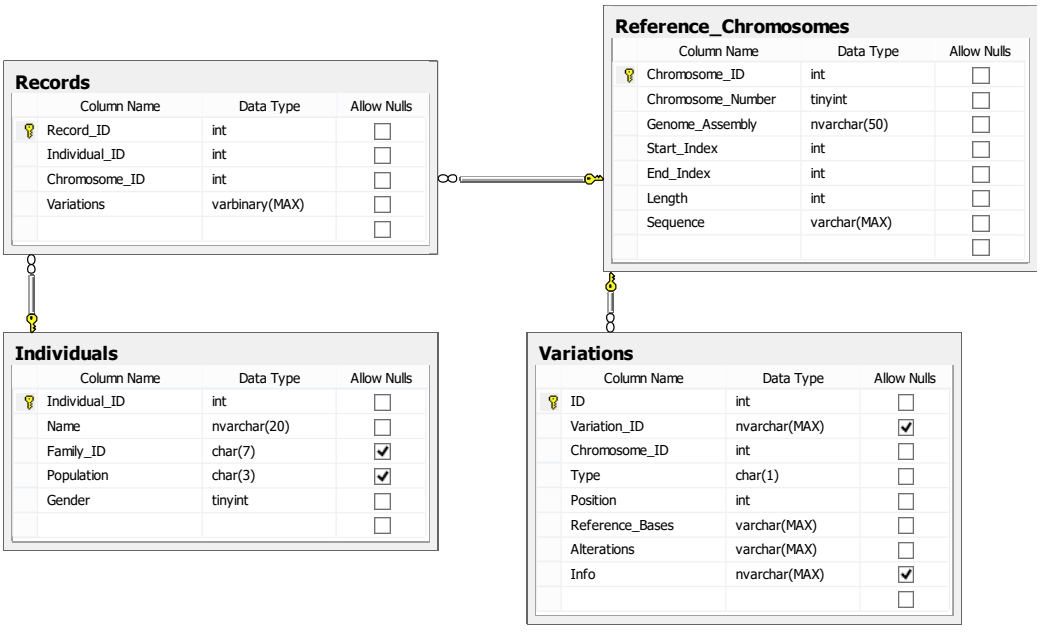

Figure 1. Relational Database Schema

3.1. Table Individuals

The table "Individuals", as the name suggests, is a table designed to hold information about individuals. Normally, this table is one of the indispensable tables in any hospital information system. Naturally, the integration of personal genetic data into the hospital information system will be carried out through this table. Consciously, the table was kept as simple as possible and the reason for this will be explained below. This table has a total of five fields. The first of these, the field with the name "Individual_ID", is the primary key of the table. Therefore, thanks to this index, other information about any person whose “Individual_ID” is known, can be retrieved from the

database very quickly. In addition, this field is the foreign key in the "Records" table, which will be explained in detail later and the relationship between these two tables appears in Figure 1. Other fields of the "Individuals" table hold the name, family id, population and sex of the person and the names of these fields are "Name", "Family_ID", "Population" and "Gender", respectively. In fact, the fields of the Individuals table can be further increased in line with the needs of clinical applications; but because we used fundamentally the variant data that the 1000 genomes project published, we had to adhere to some of the designs in this project. On the other hand, basically, even only 3 fields ("Individual_ID", "Name" and "Gender") are sufficient to hold variation-based personal genetic data. From these areas, the "Gender" field is important for reading (transforming into genotype / haplotype objects from the byte sequence created in accordance with the general format) individual genotypes or haplotypes of sex chromosomes. Since females have two X chromosomes, the variation-based genetic data for female x chromosomes must be stored in a diploid format and read. On the other hand, males have one x and one y chromosome. For this reason, variation-based genetic data for male sex chromosomes should be stored and read in haploid format.

3.2. Table Reference_Chromosomes

Another one, from the tables that are required to hold variation-based personal genetic data, is the "Reference_Chromosomes" table. As is known, the personal variations are determined by the alignment of the raw sequence data of the person to the reference chromosome. Therefore, the reference chromosome used for this operation is important. If the reference chromosome used to align the raw sequence data is changed, the individual variations that will emerge will also change. For this reason, the reference chromosome used for alignment is one of the indispensable data in a database that holds variation-based personal genetic data. Accordingly, the "Reference_Chromosomes" table has been designed and added to the schema.

This table consists of a total of seven fields and each reference chromosome corresponds to a row of this table. The fields of the table were named as "Chromosome_ID", "Chromosome_Number", "Genome_Assembly", "Start_Index", "End_Index", "Length" and "Sequence", respectively. From these fields, the "Chromosome_ID" field is the primary key of the table. Also, this field is the foreign key in the "Records" table, and the relationship between these two tables is also shown in Figure 1. The second field of the table, "Chromosome_Number", as the name implies, holds the number of the chromosome. At this point, let's say that number 23 represents the X chromosome and number 24 represents the Y chromosome. The third field of the table, "Genome_Assembly", holds the assembly of the reference chromosome. Thanks to this field, different versions (assembly) of the same chromosome (e.g. chromosome-1) can be stored in the table. In this way, we have the possibility to store multiple records (provided that the assemblies are different) of the same chromosome of the same person in the "Records" table. The other three fields of the table, "Start_Index", "End_Index" and "Length" represent the start position, end position and length of the reference chromosome, respectively.

The last field of the table, "Sequence", as the name suggests, holds the reference chromosome sequence. Because chromosomes are made up of millions of base pairs, the data held in this field is quite big. For example, suppose that the chromosome consists of 300,000,000 base pairs. In this case, the sequence data held in this field takes up approximately 300 MB. Although this data is quite large, it can be retrieved from the database in a short time because it is a single data (whole). On the other hand, any part of the raw sequence data can be fetched by the SqlDataReader.GetChars() method in the System.Data.SqlClient library, which is provided by .net. Actually, the structure of the "Reference_Chromosomes" table also provides the possibility of storing reference chromosome sequences in pieces; but, the structure of the "Records" table and the general format we have designed prevent it. To store the reference chromosome sequences in pieces, one more table should be added to the database, and the newly added table must be associated with the "Reference_Chromosomes" table.

3.3. Table Records

The "Records" table is the most significant table of our relational database and consists of only four fields. These fields are "Record_ID", "Individual_ID", "Chromosome_ID", and "Variations", respectively. The field "Record_ID" is the primary key of this table. On the other hand, the "Individual_ID" field of this table is linked to the "Individual_ID" field of the "Individuals" table. The same also applies to the "Chromosome_ID" field and the "Chromosome_ID" field of the "Reference_Chromosomes" table. The field, where the variation-based personal genetic data is stored, is the "Variations" field. As can be noted, the data type of this field is varBinary (MAX), that is, the data is stored in the form of a byte array in this field. In our practice, the personal genetic data encoded in the form of a byte sequence in accordance with the general format is held in this field in a compressed form. The details about compression will be given in the section that describes the analysis results. To summarize, one row of the "Records" table holds genotypes/haplotypes associated with a chromosome of a person. In this way, for a single assembly, the data of the whole female genome is kept in 23 rows. This number is 24 for men.

3.4. Table Variations

The "Variations" table is designed to hold generic variations. Both non-structural variations and structural variations are stored in this table. For this purpose, there are eight fields on the table. Also, the only table that the "Variations" table is linked to is the

"Reference_Chromosomes" table and the fields "Chromosome_ID" of both tables are linked together to form this relationship. Naturally, the "Chromosome_ID" field of the "Variations" table is the foreign key. Other fields of the table and the intended use of these fields are as follows: The "ID" field is the primary key of the table. The field "Variation_ID" represents the id of the generic variation. At this point, the "ID" field and the "Variation_ID" field should not be confused. The "ID" field uniquely identifies the variation in the table and this field is automatically assigned a value by the database itself. On the other hand, the "Variation_ID" field also uniquely identifies the variation and the variation IDs are determined by the organizations with international validity; but, for newly discovered variations, the value of this field is null. Already, the IDs of thousands of newly detected variations have been indicated as null in the variation catalog published by 1000 genomes project. The fourth field of the table, the "Type" field, indicates the type of variation and this field only takes two values: while the 'N' character represents non-structural variations, the 'S' character represents structural variations.

Other areas of the table named "Position", "Reference_Bases" and "Alterations" hold the position at which the variation occurs, the reference allele and the alternate alleles of the variation, respectively. In addition, alternate alleles are separated from each other by ';' in the "Alterations" field. The last field of the table, "Info", holds the other information associated with the variation. The information in this field is stored in the form of colon separated <property>=<value> pairs.

4. ENCODING AND DATA FORMATS

4.1. Encoding the Sequence

As known, DNA sequence is made up of nucleotide bases and there are four bases. These are Adenine, Guanine, Cytosine and Thymine. Apart from these, sequencing devices return "N" character for the bases which cannot be identified. Therefore, there are totally 5 characters. Three bits are sufficient to represent these 5 characters. But, since a byte is composed of 8 bits, we preferred using 4 bits to represent the bases. Naturally, one byte can store two characters. Accordingly, the formula "Ceiling(Length/2)" is used to compute the number of bytes adequate to store the sequence, where "Length" stands for the length of the sequence in the formula.

Table 1. The Format for Small Indels

Variation Type	Field		Description	Type
Substitution & Insertion	L_T		Length of the Sequence(Higher 4 bits) and Type of the Variation(Lower 4 bits)	Unsigned Byte
	If Length < 15	Alt	A byte array storing the sequence, byte[Ceiling(Length/2)]	Byte Array
	If Length = 15	Len	Length of the Sequence	Unsigned 16 Bit Integer
		Alt	A byte array storing the sequence, byte[Ceiling(Length/2)]	Byte Array
Deletion	L_T		Length of the Sequence(Higher 4 bits) and Type of the Variation(Lower 4 bits)	Unsigned Byte
	If Length < 15	-	Null, no information is stored	-
	If Length = 15	Len	Length of the Sequence	Unsigned 16 Bit Integer

4.2. Small Indels Variation Format

The basic element of the data format we developed is variation. There are two different formats for non-structural variations. While the same format was designed for both substitution and insertion, a different format was designed for deletion. The cause that gives rise to this situation is while we should store the sequence in insertion and substitution, we don't need to store the sequence in deletion. The first field of both formats is "L_T". While higher 4 bits of "L_T" indicates the length of the sequence, lower 4 bits indicates the type of the variation. If the value of the higher 4 bits of "L_T" is 15, there are two alternatives. The length of the sequence might be either 15 or more than 15. Therefore, there is also an extra field "Len" in the case where the value of the higher 4 bits of "L_T" is 15. The field "Len" stores the length of the sequence. The last field of the format devised for substitution and insertion is "Alt" and this field is used to store the sequence. Since we don't need to store the sequence in deletion, there is no "Alt" field in the deletion

format. You can attain the other details about the variation format from Table 1.

4.3. Structural Variation Format

As is known, the structural variations are basically divided into five. These are Cnv (Copy number variation), Del (Deletion), Dup (Duplication), Ins (Insertion) and Inv (Inversion). Furthermore, the structural variation of the insertion type is also divided into four within itself, depending on the type of the inserted element. As a result, there are a total of 8 different structural variations. The information that defines these eight different structural variations and naturally, that needs to be stored shows similarities and/or differences. Therefore, it would be a right and rational approach to group structural variations that have more common features and to design a common format for each group. This choice will both provide a significant convenience in terms of design and reduce space requirement. From this point of view, 5 structural variations with the most common features were collected in one group and the other 3 structural variations were collected in a separate group. The formats designed for these two groups appear in Table 2 and Table 3, respectively. For the sake of clarity, the table structure created for small indels is adopted here as well. In addition, since the formats of the structural variations are more complex, they were divided into a certain number of sections. Our goal in doing so is to be able to explain the formats more easily. Accordingly, the thick lines in the tables separate these sections. For example, in Table 2 there are two thick lines and these two thick lines separate three sections.

Table 2. Structural Variation Formats-1

Variation Type	**Field**		**Description**	**Type**
CNV & DEL & DUP & INS:MT & INV	CNV DEL DUP	nCNA_Type	How many CNAs (Higher 4 bits) and Type of the Variation (Lower 4 bits)	Unsigned Byte
		CNA	Copy Number Allele: How many copies	Signed byte[nCNA]
	INS:MT INV	U_Type	Higher 4 bits unused and Type of the Variation (Lower 4 bits)	Unsigned Byte
	L_ID		Length of the Variation ID	Unsigned Byte
	ID		Variation ID	Char[L_ID]
	L_CS		Length of the Source Call Set	Unsigned Byte
	CS		Source Call Set	Char[L_CS]
	END		End coordinate of the Variation	Unsigned 32 Bit Integer
	If not INS:MT	-	Null, no information is stored	-
	If INS:MT	M_Start	Mitochondrial start coordinate of inserted sequence	Signed 32 Bit Integer
		M_End	Mitochondrial end coordinate of inserted sequence	Signed 32 Bit Integer
	EInfo_Type		Existence of Extra Info (higher 4 bits) and Type of the field L_EInfo (Unsigned Byte=1, Unsigned 16 Bit Integer=2)	Unsigned Byte
	If EInfo=1	L_EInfo	Length of the Extra_Info	Appropriate type
		Extra_Info	Extra Information for the Variation (colon separated <property>=<value> pairs)	Char[L_EInfo]

The fields of the format designed to hold the information of five structural variations and their characteristics are as follows: The first part of the table is divided into two according to the types of the variations. If the type of variation is Cnv, Del or Dup, the first part consists of two fields. These fields are “nCNA_Type” and “CNA”. The "nCNA_Type" is of type unsigned byte and the first four bits of this field are used to hold the number of CNAs. On the other hand, the last four bits represent the type of variation. The field “CNA” is an array of type signed byte and holds the copy numbers. If the type of the variation is Ins:Mt or Inv, the first section consists of a single field, and the name of this single field is "U_Type". "U Type" is of type unsigned byte and the last four bits of this field are used to hold the type of the variation; the first four bits are not used.

The first five fields of the second section are standard, and these fields are "L_ID", "ID", "L_CS", "CS" and "END", respectively. From these fields, "L_ID" can be handled together with "ID". The same situation applies to "L_CS" and "CS". The "L_ID" field is used to keep the length of the variation-id represented by "ID". Similarly, the "L_CS" field is used to hold the length of the Source Call Set represented by "CS". The last field existing as the standard of the second section is "END" and holds the end coordinate of the variation. Apart from these five standard fields, if the type of the variation is Ins:Mt, there are two extra fields in this section. These fields, whose names are "M_Start" and "M_End" denotes mitochondrial start and end coordinates of the inserted sequence. Accordingly, the type of these fields is Signed 32 Bit Integer.

The third section, that is, the last section, is basically designed to hold extra information concerning the variation. The first field of this section is standard. The first four bits of this field, whose name is "EInfo_Type", indicate whether extra information is available. The fact that the first four bits are 1 means that extra information is available. If the first four bits are 1, there are two other fields at the end of this section to keep the other information associated with the variation. The names of these fields are "L_EInfo" and "Extra_Info". The "L_EInfo" field represents the length of the "Extra_Info" field. On the other hand, the type of the "L_EInfo" field is determined according to the last four bits of the "EInfo_Type" field. If the last four bits of "EInfo_Type" are 1, the type of "L_EInfo" is unsigned byte; on the contrary, if this value is 2, the type of "L_EInfo" is unsigned 16 Bit Integer. The last field of this last section, the "Extra_Info" field, is the actual field that holds the other information associated with the variation. The information in this field is stored in the form of colon separated <property>=<value> pairs.

The common format developed for the second group of structural variations (INS:ME:ALU, INS:ME:LINE1, INS:ME:SVA) appears in Table 3. As can be seen from the table, this common format, developed for three structural variations, consists of three sections and these three sections are separated from each other by two thick lines. The fields of each section and their characteristics are as follows: In the first section there are eight different fields and the names of these fields are "MEINFO_Type", "SVLEN", "L_ID", "ID", "L_CS", "CS", "L_TSD" and "TSD", respectively. The field "MEINFO_Type" is of type unsigned byte and the first four bits of this field determine whether mobile element information exists or not. If the first four bits are 1, the second section stores the information about the mobile element, but, if the first four bits are 0, then the second section stores nothing, in other words, the second section doesn't exist. On the other hand, the last four bits of "MEINFO_Type" represent the type of the variation. The second field of the first section, "SVLEN", denotes the length of the structural variation and the type of this field is Signed 32 Bit Integer. From the remaining fields of the first section, "L_ID", "L_CS" and "L_TSD" are used to store the lengths of the fields "ID", "CS" and "TSD", respectively. The "ID" field represents the variation-id and, accordingly, its type is a char array. On the other hand, while "CS" holds the Source Call Set, "TSD" holds Precise Target Site Duplication for bases. Likewise, the types of these fields are also char arrays.

The second section is basically designed to store information regarding the mobile element. As we have already mentioned, the presence of this section depends on the first four bits of the first field of the first section. In accordance with the purpose of this section, there are five standard fields. The first field of this section, "L_MEN", is used to keep the length of the mobile element name represented by the field "MEN". The names of the two subsequent fields are "ME_SPos" and "ME_EPos". These fields are used to hold the start and end positions of the mobile element, respectively, and the types of both fields are Signed 32 Bit Integer. The last field of this section, whose name is "Polarity", denotes the polarity of the mobile element. Accordingly, the value of this field can be either '+' or'-'. Because one byte (or one character) is sufficient to represent these values, the type of the "Polarity" field was determined as char.

The third section is basically designed to hold extra information regarding the variation. The first field of this section is standard. The first four bits of this field, whose name is "EInfo_Type", indicate whether extra information is available. The fact that the first four bits are 1 means that extra information is available. If the first four bits are 1, there are two other fields at the end of this section to keep the other information associated with the variation. The names of these fields are "L_EInfo" and "Extra_Info". The "L_EInfo" field represents the length of the "Extra_Info" field. On the other hand, the type of the "L_EInfo" field is determined according to the last four bits of the "EInfo_Type" field. If the last four bits of "EInfo_Type" are 1, the type of "L_EInfo" is unsigned byte; on the contrary, if this value is 2, the type of "L_EInfo" is unsigned 16 Bit Integer. The last field of this last section, the "Extra_Info" field, is the actual field that holds the other information associated with the variation. The information in this field is stored in the form of colon separated <property>=<value> pairs.

Table 3. Structural Variation Formats-2

Variation Type	Field		Description	Type
INS:ME:ALU & INS:ME:LINE1 & INS:ME:SVA	MEINFO_Type		Mobile element info (Higher 4 bits) and Type of the Variation(Lower 4 bits)	Unsigned Byte
	SVLEN		Structural Variation Length	Signed 32 Bit Integer
	L_ID		Length of the Variation ID	Unsigned Byte
	ID		Variation ID	Char[L_ID]
	L_CS		Length of the Source Call Set	Unsigned Byte
	CS		Source Call Set	Char[L_CS]
	L_TSD		Length of TSD	Unsigned Byte
	TSD		Precise Target Site Duplication for bases (if unknown, value will be null)	Char[L_TSD]
	If MEINFO=0	-	Null, no information is stored	-
	If MEINFO=1	L_MEN	Length of MEN	Unsigned Byte
		MEN	Mobile element name	Char[L_MEN]
		ME_SPos	Mobile element Start Position	Signed 32 Bit Integer
		ME_EPos	Mobile element End Position	Signed 32 Bit Integer
		Polarity	Polarity ('+' or '-')	Char
	EInfo_Type		Existence of Extra Info (higher 4 bits) and Type of the field L_EInfo (Unsigned Byte=1, Unsigned 16 Bit Integer=2)	Unsigned Byte
	If EInfo=1	L_EInfo	Length of the Extra_Info	Appropriate type
		Extra_Info	Extra Information for the Variation (colon separated <property>=<value> pairs)	Char[L_EInfo]

4.4. General Format

The general format, which was designed for the storage of the personal genotypes or haplotypes (depending on the sex of the individual and/or chromosome), can be seen from Table 4. As it can be seen from the table, the general format was devised to store all or a portion of the personal genotypes/haplotypes for the variations detected on any chromosome of an individual. In short, the general format can be defined as: chromosomes are composed of regions; each region is made up of records.

The details of the format are as follows: Since the variations are detected by aligning a sequence to the reference sequence, the assembly of the human reference genome should be specified. Accordingly, the first two fields of the general format are used for that purpose. The names of these fields are "L_Hga" and "Hga", respectively. "L_Hga" holds the character length of the human genome assembly, represented by the field "Hga". Additionally, the types of these fields are unsigned byte and char array. The third field of the format is "N_Regions". "N_Regions" is used to hold the number of regions. After that field, the part comes where the indices of the regions are held.

The indices section is actually a list and its length equals the value of the "N_Regions" field. On the other hand, the fact that this section is a list is indicated with a rational visual approach in the table. Each element of the indices section is composed of three different fields. These fields are "C_SPos", "C_EPos" and "SO_CReg". The fields "C_SPos", "C_EPos" and "SO_CReg" hold the start position and end position of the region on the chromosome, and the start offset of the compressed region in the byte array, respectively. Regions are kept in order according to their start and end positions. This simple but significant approach gives us the possibility to make binary search. In this way, the region or regions that match the given position information can be determined in a very short time. Besides, since the initial offset (SO_CReg) in the

compressed byte array of any detected region is also held, there is also the opportunity to position directly in that position and bring the relevant region. Here, the corresponding region is fetched as a compressed byte array and then it is decompressed.

Table 4. General Format

Field			Description	Type
L_Hga			Length of the Human Genome Assembly	Unsigned Byte
Hga			Human Genome Assembly (ex. GRCh37)	Char[L_Hga]
N_Regions			Number of Regions	Unsigned 32 Bit Integer
List of Indexes (n=N_Regions)				
	C_SPos		Chromosomal Start Position of the Region	Unsigned 32 Bit Integer
	C_EPos		Chromosomal End Position of the Region	Unsigned 32 Bit Integer
	SO_CReg		Start offset of the compressed region	Unsigned 32 Bit Integer
List of Uncompressed/Compressed Regions (n=N_Regions)				
	N_Rec		Number of Records in the Region	Unsigned 32 Bit Integer
	List of Records (n=N_Rec)			
	If Haploid	V_Pos	Position of the Variation	Unsigned 32 Bit Integer
		Var	Variation	Variation
	If Diploid	V_Pos	Position of the Variation	Unsigned 32 Bit Integer
		I_Byte	For Diploid Genotypes, there are five situations: Only first allele has the variation, I_Byte=00000000 Only second allele has the variation, I_Byte=00000001 Both alleles have the same variation, I_Byte=00000010 The alleles have different variations, I_Byte=00000011 The alleles have the same Structural variations, but different CNA, I_Byte=00000100	Unsigned Byte
		Var	If the value of I_Byte is 00000011, two variations	Variation[]
			Else, one variation	Variation

The next section of the general format holds compressed regions. This section is a list of compressed regions, and the length of this list is, as you will guess, equal to the length of the index list. Although we express compressed regions and index region as a list, the whole of the general format is actually a byte array, and as we have already mentioned, we have the possibility to position on any offset of this array. When looking at the format of a compressed region in Table 4, basically two components are seen. The first of these, the field with the name "N_Rec" shows the number of records (personal genotype or haplotype) in the compressed region, and the type of this field is Unsigned 32 Bit Integer. The second component is a list of records. On the other hand, there are two different Record formats, one for haploid calls and the other for diploid calls. Haploid format consists of two fields ("V_Pos", "Var") whereas diploid format consists of three ("V_Pos", "I_Byte", "Var"). The extra field "I_Byte" of the diploid format is used to represent five situations whose details are given in Table 4. For haploid calls, e.g. on Y, male nonpseudoautosomal X, only one allele value should be given. Accordingly, one variation is stored in the "Var" field of Haploid format. For diploid calls, e.g. on chromosome 1, female nonpseudoautosomal X, two alleles' values should be given. Here, based on the value of the field "I_Byte", either one variation or two variations can be stored in the "Var" field of Diploid format.

5. THE RESULTS OF THE ANALYSIS MADE ON THE RELATIONAL DATABASE

In the process of recording variation-based personal genetic data into the relational database, the data are passed through many stages and/or undergoing transformation. These stages are the encoding of class objects according to the general format (conversion to byte array), the compression (in parts) of the data encoded in the form of byte array and storage of the compressed byte array into the database, respectively. On the other hand, in the process of reading the data stored in the database and transforming

it into class objects, these steps are implemented backward. Namely; first, the data held in the form of compressed byte array is fetched from the database, then, this data is decompressed, and finally, the decompressed data is transformed into class objects. Table 5 below shows the analysis results regarding time requirements of the operations carried out at the mentioned stages. In addition, the results of the analysis on the size of the area occupied by the data at each step are also given in Table 6. The values indicated on both tables are the average of 2504 people. Also, analyzes on time and space requirements were performed separately for each chromosome. At this point, since the sex chromosomes of females and males are different, analyzes of the X chromosome were made separately for females and males. Accordingly, the rows starting with "X) F" and "X) M" on both tables indicate the mean values of the females and males, respectively. The various physical characteristics of the test computer are as follows: (Asus K55VJ-SX077D, Windows 8.1 Pro 64 bit, Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz, 6 MB Intel® Smart Cache, 8 GB DDR3 1600 MHz, 750 GB 7200rpm). Also, Sql Server 2008 R2 was selected as the database management system.

Table 5, which shows the time requirements of the processes that need to be realized in the system, consists of a total of 12 columns, if we do not take the first column into account. Eight of these columns are dedicated to compression and decompression operations, two of them are to the operations of writing to the database and reading from the database, one of them is to encoding operation and one of them is to decoding operation. In the process of recording variation-based personal genetic data into the relational database, encoding operation (conversion to byte array) and the compression of the byte array lead to extra source usage (RAM) and delay. On the other hand, similar source usage and delay are valid for the decompression and the decoding operations in the process of reading the data stored in the database. The time losses caused by these four operations appear in Table 5.

Table 5. Time Table for the Operations Related to the Relational Database (in millisecond)

Chr No	**Encoding General format**	**Gzip Algorithm**				**Deflate Algorithm**				**Database**		**Decoding**
		Gzip Fastest		Gzip Optimal		Deflate Fastest		Deflate Optimal		Write	Read	
		Comp	Decomp	Comp	Decomp	Comp	Decomp	Comp	Decomp			
1)	117.04	79.37	47.08	173.81	42.19	62.73	40.05	165.84	35.61	54.32	26.85	278.5
2)	122.1	82.78	51.07	183.35	46.79	66.29	42.61	173.58	39.21	58.53	34.35	296.58
3)	106.07	76.23	42.12	156.5	38.07	53.07	34.6	147.57	31.36	57.3	41.55	246.44
4)	120.45	73.89	44.36	167.7	40.65	61.63	36.91	161.98	33.45	120.42	44.4	262.69
5)	101.12	69.64	37.74	138.11	32.71	47.83	31.67	133.62	28.7	150.52	42.54	217.56
6)	107.04	78.49	40.97	154.84	35.74	50.22	32.23	140.67	29.06	110.09	41.78	220.71
7)	96.08	70.06	35.3	132.07	31	46.36	29.6	128.2	26.29	124.45	42.5	201.61
8)	88.52	66.94	32.97	121.96	28.74	43.95	26.69	118.89	24.71	129.08	41.63	182.9
9)	70.46	52.98	25.32	97.58	22.61	33.86	21.32	92.41	18.98	89.15	38.99	145.07
10)	84.09	56.54	31.93	116.22	27.18	43.4	25.32	109.11	23	110.41	39.77	172.93
11)	83.19	59.33	32.4	114.88	27.79	43.91	26.86	113.79	24.41	143.48	40.36	179.57
12)	81.12	60.48	29.26	110.96	25.78	44.15	23.74	104.6	21.6	94.37	39.49	155.42
13)	65.58	45.52	23.56	85.74	22.01	36.49	19.57	88.92	18.02	95.04	33.17	126.62
14)	59.34	46.27	19.98	74.66	17.68	28.49	16.44	72.66	15.91	80.23	32.31	104.82
15)	51.83	39.96	18	70.28	16.19	26.22	14.7	63.97	13.39	59.5	30.27	93.12
16)	53.01	45.79	18.45	75.39	16.42	25.65	15.26	67.76	14.36	59.43	31.56	97.95
17)	45.42	37.67	17.13	65.67	15.58	23.07	13.86	63.05	12.57	34.74	28.88	82.26
18)	46.8	39.78	17.5	66.05	15.67	30.77	15.26	65.2	13.23	23.2	29.52	89.28
19)	37.97	39.95	14.65	55.5	12.92	22.37	11.68	52.16	10.56	30.99	29.05	73.96
20)	34.8	35.41	14.77	53.64	11.88	21.16	10.68	53.15	9.84	39.77	28.6	63.46
21)	25.25	23.38	9.83	44.15	8.25	14.44	7.93	33.6	6.92	15.99	22.88	39.96
22)	22.62	26.43	8.86	37.47	7.61	16.9	7.04	31.15	6.33	28.44	21.04	31.65
X)F	55.71	38.88	21.99	79.36	20.42	29.94	17.96	78.39	16.33	108.98	43.69	107.13
X)M	32.59	30.94	17.09	60.13	13.49	21.83	11.86	51.53	11.1	56.01	34.34	48.6
Y)	0.24	0.86	0.14	1.16	0.14	0.71	0.12	0.8	0.12	13.9	14.21	0.33
Female Total	1675.61	1245.77	635.24	2375.89	563.88	872.9	521.98	2260.27	473.84	1818.43	805.18	3470.19
Male Total	1652.73	1238.69	630.48	2357.82	557.09	865.5	516	2234.21	468.73	1779.36	810.04	3411.99

For compression operation, basically two different classes (algorithms) in the "System.IO.Compression" library that ".net" provides were compared. The names of these classes are "GZipStream" and "DeflateStream". In addition, two different compression levels (“Fastest” and “Optimal”) of both algorithms were tested and these levels indicate whether to emphasize speed or compression efficiency when compressing the stream. Therefore, a total of 4

different compression methods are compared. The comparison results are as follows: Except for one or two results, the “Gzip” algorithm completes the compression process longer than the “Deflate” algorithm. This applies to both "Fastest" and "Optimal" levels. That is, the "fastest" level of the Deflate algorithm is the method that performs the compression process fastest. On the other hand, in the same way, the “Gzip” algorithm completes the decompression process longer than the “Deflate” algorithm. Moreover, surprisingly, the "optimal" levels of both algorithms perform decompression operation more quickly than their "fastest" levels. Therefore, the algorithm performing the fastest decompression is “Deflate-optimal”.

Table 6 shows the size of the area occupied by the data in the recording process. In this table, except for the last two rows in which the total values for the whole genome of females and males are shown, the values in the other rows are in bytes. The values on the last two rows of the table are in MB. As can be seen from the table, the algorithm and its' level with the best compression performance is Deflate-Optimal. Both the Optimal level of the Deflate algorithm and the other three alternatives achieved quite successful results. All of these four methods have reduced the size of the byte array encoded according to the general format by approximately 50% and there is little difference between the results of the methods.

Two criteria have been taken into consideration in determining the method to be used for the compression of the byte array. These criteria are the compression ratio and the duration of the decompression process. At this point, the question "why was the duration of the decompression process chosen as a criterion?" may come to mind. As the genetic data of any person is recorded only once into the database, naturally, the compression process is also performed once. On the other hand, since the process of reading from the database is repeated in each clinical application, the decompression process is also carried out many times. From this point of view, the duration of the decompression process has been selected as the criterion. Surprisingly, in terms of both the compression ratio and the duration of the decompression operation, the best algorithm is "Deflate-optimal". Therefore, this algorithm has been used as the compression method in our application as well. Accordingly, the variation-based genetic data for the whole genome of females occupies 15.270 MB in the database. This value is 15.086 MB for males.

Table 6. Size Requirement Table for the Operations Related to the Relational Database (in bytes)

Chr No	**General format**	**Gzip Algorithm**		**Deflate Algorithm**		**Number of Regions**
		Gzip Fastest	Gzip Optimal	Deflate Fastest	Deflate Optimal	
1)	2408112	1253248	1224781	1250774	1222307	137
2)	2541726	1320030	1289846	1317418	1287234	145
3)	2166207	1121046	1095114	1118818	1092886	123
4)	2290589	1178695	1150662	1176340	1148307	130
5)	1883022	977429	955463	975492	953527	107
6)	2020235	1034925	1010897	1032850	1008822	115
7)	1789965	923548	901590	921706	899748	102
8)	1653394	847277	826890	845575	825188	94
9)	1303419	673591	657558	672248	656214	74
10)	1551477	801238	782168	799641	780571	88
11)	1558968	805382	786268	803777	784664	89
12)	1475290	767316	749813	765797	748294	84
13)	1170156	607522	593367	606316	592161	66
14)	1006770	522732	510626	521693	509587	57
15)	906607	471686	460954	470749	460017	52
16)	946070	485095	473347	484117	472370	54
17)	839810	438596	428962	437727	428093	48
18)	884164	457790	446836	456876	445922	50
19)	707567	363672	355261	362939	354528	40
20)	652718	337391	329537	336714	328860	37
21)	465189	239537	233823	239051	233338	26
22)	420671	217256	212262	216817	211823	24
X)F	1103835	579597	568871	578456	567730	63
X)M	655681	373474	372391	372683	371600	43
Y)	4845	2959	3013	2941	2995	1
Female Total	30.275 MB	15.663 MB	15.301 MB	15.632 MB	15.270 MB	1805
Male Total	29.852 MB	15.469 MB	15.117 MB	15.439 MB	15.086 MB	1786

In clinical applications, the data to be processed in RAM are class objects. In the process of converting the personal genetic data held in the database into class objects, the data passes through multiple stages and the space occupied by

the data at each step must also be taken into consideration. Let's explain this with an example. The whole genome data of females takes up 15.27 MB in the database. When this data is stored in memory, it again occupies 15.27 MB in the same way. Because this data is a compressed data, it is decompressed primarily and as a result of the decompression process, a separate byte array that occupies 30.275 MB emerges. It is no longer necessary to keep the compressed data in memory at this point, and for this reason, the compressed data is discarded. Then, a byte array encoded according to the general format and occupying 30.275 MB in memory is passed through decode operation and the personal genotype objects are generated. After the personal genotype objects are generated, the byte array encoded according to the general format is discarded from the memory. Therefore, the areas used in these two intermediate stages are seen as temporary losses.

6. CONCLUSION

In this study, a relational database was utilized for the organization of variation-based personal genetic data, and the space requirements and query performances of this database was computed. Relational databases have been successfully used in many areas so far. But unfortunately, there are several challenges confronted when using relational databases for storing personal genetic data. In order to store all the variations existing in the genome of a person in relational database, normally, millions of rows are required. In practice this is almost impossible. However, based on our storage approach and the designed data formats, variations of each chromosome are stored in type of "varbinary (MAX)". In this way only 23 rows are used to store variation-based genetic data for the entire genome of females. In contrast to females, the number of rows required to store variation-based genetic data for the whole genome of males is 24. By using this proposed method, the space required to store all the variations in the genome of a person is approximately 0.26 % of the space required to store the raw sequence of this person. This means that, on average, our method provides a space saving of approximately 99.74%. In addition, our method yielded better results than the accomplished compression methods. The space need of our method is 0.015 GB as opposed to 1.2 GB obtained by the accomplished compression methods.

REFERENCES

[1] International Human Genome Sequencing Consortium, “Finishing the euchromatic sequence of the human genome”, *Nature*, 431(7011), 931-945, 2004.

[2] International HapMap Consortium, “A second generation human haplotype map of over 3.1 million SNPs”, *Nature*, 449, 851–861, 2007.

[3] 1000 Genomes Project Consortium, “A map of human genome variation from population-scale sequencing”, *Nature*, 467(7319), 1061-1073, 2010.

[4] 1000 Genomes Project Consortium, “An integrated map of genetic variation from 1,092 human genomes”, *Nature*, 491(7422), 56-65, 2012.

[5] 1000 Genomes Project Consortium, “A global reference for human genetic variation”, *Nature*, 526(7571), 68-74, 2015.

[6] P. H. Sudmant, et al., “An integrated map of structural variation in 2,504 human genomes”, *Nature*, 526(7571), 75-81, 2015.

[7] B. Alberts, et al., **Molecular Biology of the Cell**. Garland Science, New York, A.B.D., 2007.

[8] M. M. Alves, et al., “Contribution of rare and common variants determine complex diseases—Hirschsprung disease as a model”, *Developmental biology*, 382(1), 320-329, 2013.

[9] W. P. Gilks, J. K. Abbott, E. H. Morrow, “Sex differences in disease genetics: evidence, evolution, and detection”, *Trends in Genetics*, 30(10), 453-463, 2014.

[10] J. Hardy, A. Singleton, “Genomewide association studies and human disease”, *N. Engl. J. Med*, 360, 1759–1768, 2009.

[11] W. L. Lowe, T. E. Reddy, “Genomic approaches for understanding the genetics of complex disease”, *Genome research*, 25(10), 1432-1441, 2015.

[12] C. Katsios, D. H. Roukos, “Individual genomes and personalized medicine: life diversity and complexity”, *Personalized Medicine*, 7(4), 347-350, 2010.

[13] M. A. Hamburg, F. S. Collins, “The path to personalized medicine”, *New England Journal of Medicine*, 363(4), 301-304, 2010.

[14] G. S. Ginsburg, J. J. McCarthy, “Personalized medicine: revolutionizing drug discovery and patient care”, *TRENDS in Biotechnology*, 19(12), 491-496, 2001.

[15] N. J. Schork, “Personalized medicine: time for one-person trials”. *Nature*, 520(7549), 609-611, 2015.

[16] E. L. Van Dijk, H. Auger, Y. Jaszczyszyn, C. Thermes, “Ten years of next-generation sequencing technology”, *Trends in genetics*, 30(9), 418-426, 2014.

[17] Internet: Fasta Format, https://en.wikipedia.org/wiki/FASTA_format, 20.04.2018.

[18] A. Löytynoja, N. Goldman, “An algorithm for progressive multiple alignment of sequences with insertions”, **Proceedings of the National academy**

of sciences of the United States of America, 102(30), 10557-10562, 2005.

[19] H. Li, N. Homer, "A survey of sequence alignment algorithms for next-generation sequencing", *Briefings in bioinformatics*, 11(5), 473-483, 2010.

[20] T. Lassmann, E. L. Sonnhammer, "Kalign–an accurate and fast multiple sequence alignment algorithm", *BMC bioinformatics*, 6(1), 2005.

[21] O. Çakırgöz, **Organization and Processing of Personal Genetic Data for Clinical Use**, Phd Thesis, Dokuz Eylül University, The Graduate School of Natural and Applied Sciences, 2017.

[22] S. Grümbach, F. Tahi, "Compression of DNAsequences", **Proceedings of the IEEE Data Compression Conference (DCC)**, 340–350, 1993.

[23] X. Chen, et al., "DNACompress: fast and effective DNA sequence compression", *Bioinformatics*, 18(12), 1696-1698, 2002.

[24] B. Behzadi, F. L. Fessant, "DNA compression challenge revisited: a dynamic programming approach", *CPM*, Springer, 190–200, 2005.

[25] M. D. Cao, et al., "A simple statistical algorithm for biological sequence compression", **Proceedings of the IEEE Data Compression Conference (DCC)**, 43–52, 2007.

[26] Internet: The Variation data as VCF files, ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/, 23.07.2016.

[27] Internet: The Variation data as BCF files, ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/supporting/bcf_files, 23.07.2016.

[28] Internet: The VCF File Format, https://github.com/samtools/hts-specs, 19.03.2016

[29] J. Hammer, M. Schneider, "Genomics Algebra: A New, Integrating Data Model, Language, and Tool for Processing and Querying Genomic Information", **Proceedings of the 2003 CIDR Conference**, 2003.

[30] S. Tata, **Declarative querying for biological sequences**, Phd Thesis, The University of Michigan, Michigan, 2007.

[31] V. Bafna, et al., "Abstractions for genomics", *Communications of the ACM*, 56(1), 83-93, 2013.

[32] T. J. Pemberton, Z. A. Szpiech, "Relationship between Deleterious Variation, Genomic Autozygosity, and Disease Risk: Insights from The 1000 Genomes Project", *The American Journal of Human Genetics*, 102(4), 658-675, 2018.

[33] J. S. A. Ramos, et al., "Unraveling CYP2E1 haplotypes in alcoholics from Central Brazil: a comparative study with 1000 genomes population", *Environmental Toxicology and Pharmacology*, 62, 30-39, 2018.

[34] K. Okamura, et al., "Lists of HumanMethylation450 BeadChip probes with nucleotide-variant information obtained from the Phase 3 data of the 1000 Genomes Project", *Genomics data*, 7, 67-69, 2016.

[35] K. Nunes, et al., "HLA imputation in an admixed population: An assessment of the 1000 Genomes data as a training set", *Human immunology*, 77(3), 307-312, 2016.

[36] S. Demircioğlu, S. Özdemir, "İlişkisel Veri Tabanlarında Anahtar Kelime Arama", *Bilişim Teknolojileri Dergisi*, 5(3), 51-56, 2012.

[37] S. Öztürk, H. Atmaca, "İlişkisel ve İlişkisel Olmayan (NoSQL) Veri Tabanı Sistemleri Mimari Performansının Yönetim Bilişim Sistemleri Kapsamında İncelenmesi", *Bilişim Teknolojileri Dergisi*, 10(2), 199-209, 2017, DOI: 10.17671/gazibtd.309303.

[38] A. Haltaş, A. Alkan, "Medlıne Veritabanı Üzerinde Bulunan Tıbbi Dökümanların Kanser Türlerine Göre Otomatik Sınıflandırılması", *Bilişim Teknolojileri Dergisi*, 9(2), 181-186, 2016.