Relational Data Models for Genetic VCF data

Mohamed Sabri Hafidi
Free University of Bozen-Bolzano
hmohamedsabri@unibz.it

Ozan Kahramanoğulları
Free University of Bozen-Bolzano
okahramanogullari@unibz.it

Anton Dignös
Free University of Bozen-Bolzano
anton.dignoes@unibz.it

Johann Gamper
Free University of Bozen-Bolzano
johann.gamper@unibz.it

ABSTRACT

The Variant Call Format (VCF) and its binary counterpart (BCF) are commonly used in bioinformatics for storing gene sequence data. While VCF files provide compact storage, they require specific tools and scripts for querying, thereby missing the rich functionality arsenal of database management systems and their potential for integration in multiomics pipelines. In this paper, we leverage Relational Database Management Systems (RDBMS) to enhance efficiency and flexibility in storing and querying large-scale genetic datasets. We map the VCF file structure to narrow, wide, and array-based data models that are further refined using JSON data structures, resulting in eight data models. Our experimental evaluation shows that RDBMS provide competitive performance in comparison with specialized state-of-the-art tools while making full-fledged database capabilities available for genetic data analysis.

PVLDB Reference Format:
Mohamed Sabri Hafidi, Ozan Kahramanoğulları, Anton Dignös, and Johann Gamper. Relational Data Models for Genetic VCF data. PVLDB, 18(11): 4045 - 4053, 2025.
doi:10.14778/3749646.3749674

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://gitlab.inf.unibz.it/dbs/vcf_to_rdbms.

1 INTRODUCTION

The increasing volume of data in bioinformatics, especially in genomics, presents new challenges for data storage and analysis. As the cost of collecting genetic data continues to decrease, the number and scale of sequencing studies rises [25, 30, 33]. Over the next decade, it is estimated that the amount of genetic data will reach petabytes [16]. This surge in genomic data is also leading to a new standard of healthcare, where treatments are tailored to an individual's genetic makeup [11, 23]. This trajectory on multiple fronts is creating a pressing need for scalable solutions to store and query massive volumes of genetic data.

Genetic data is typically stored in Variant Call Format (VCF) files [8, 12, 16], which provide a standardized way to represent genetic variants across samples. A VCF file usually contains information on millions of gene variants for thousands of individuals.

Existing tools, such as BCF Tools [29], excel at handling smaller genetic datasets of VCF files. However, their performance is less effective when dealing with massive cohorts with millions of samples. This limitation stems from bottlenecks in input/output operations, memory usage, and functionality. BCF Tools is well suited for variant analysis tasks [8] but cannot independently perform analysis tasks, such as Genome-Wide Association Studies (GWAS) [34]. For such complex analyses, BCF Tools works in conjunction with other specialized tools. In response to the challenges posed by the text-based structure of VCF files and the limitations of existing tools like BCF Tools, various approaches were proposed, e.g., [7, 9, 12, 14].

RDBMSs provide a promising setting for facilitating complex analytical queries, data integration with different datasets, and enhanced data management capabilities, including indexing and access control. To this end, in this paper, we provide a systematic comparison of standard RDBMS data models, exploring a range of mapping strategies, from more direct translations to more RDBMS-centric reorganizations. We set out by mapping the VCF file structure to three standard data models, namely, wide and narrow table-based and array-based data models. We refine our analysis by introducing JSON data structures into our data models. This results in a total of eight different data models that optimize performance for various query types. We discuss the design decisions for each model as well as our implementation of the models in PostgreSQL. In a detailed performance evaluation, we compare our models to state-of-the-art competitors, i.e., BCF Tools and TileDB-VCF, for storage and query runtime, and query selectivity.

Our results show that each model presents distinct trade-offs in terms of storage requirements, query performance, and flexibility. Overall, our Array and JSON models provide competitive performance compared to specialized tools while bringing about a rich set of relational database capabilities to genetic data analysis. We show the impact of query selectivity and dataset size on each model's performance, thereby exposing the model's suitability for specific workloads. Our findings offer insights into the trade-offs between various data models, enabling researchers to make informed decisions based on their specific requirements, whether they prioritize speed, storage, or the ability to handle complex queries.

The main contributions can be summarized as follows:

- We present eight novel models to store VCF data in RDBMSs, aiming to optimize performance for various query types.
- We implement the proposed models in PostgreSQL.
- We experimentally compare our proposed models to existing state-of-the-art competitors, revealing the need for

Proceedings of the VLDB Endowment, Vol. 18, No. 11 ISSN 2150-8097.
doi:10.14778/3749646.3749674

scalable solutions for processing genetic data and the rich functionalities offered by RDBMSs.

The remainder of this paper is structured as follows: Section 2 discusses related work. Section 3 provides an overview of the VCF file structure. Our proposed RDBMS data models for genomic data are introduced in Section 4 and empirically evaluated in Section 5. Section 6 explores trade-offs between data models and query performance, relating our findings to existing genomic data strategies. Section 7 summarizes our findings and outlines future work.

2 RELATED WORK

Tools and data models used in genomic research can be categorized into command-line utilities, libraries, and database systems.

Command-line utilities provide a direct way to work with VCF files. Examples include *vcf2fhir*, an open-source utility that converts VCF files into HL7 FHIR format for genomics-EHR integration [9], *VCF-kit*, a collection of utilities that analyze and annotate VCF files [7], and *bio-vcf*, a tool for filtering and converting VCF files into different formats [12]. While these tools are convenient for basic tasks, they lack advanced customization capabilities and require other tools for analyses with multiple filtering steps and data integration. Furthermore, the limited automation features of these tools hinder workflow repeatability on different datasets.

Despite these limitations, command-line utilities like *BCF Tools* and *Tabix* are state-of-the-art in this category as they balance performance and ease of use. *BCF Tools* is a command-line suite specifically designed for working with VCF and BCF (Binary VCF) files, offering a set of functionalities for various processing tasks. It allows the retrieval of specific samples or variants, e.g., based on location and variant type. BCF Tools also supports calculating summary statistics as well as manipulating files in analysis workflows [29]. *Tabix*, which complements BCF Tools, is a generic indexer for TAB-delimited genome position files as VCF files. It creates an index file for a given input data file. After indexing, Tabix can retrieve data lines overlapping regions [17]. This allows BCF Tools to efficiently extract and analyze data from large VCF files.

Libraries offer more flexibility than command-line utilities as they can be integrated into custom scripts and applications. They allow for more complex operations and analyses on VCF files. However, they tend to specialize in a specific set of features, rather than providing a general platform for programmatic data analysis, which limits their applicability. Examples of such libraries include: *vcflib*, a C++ library that provides a variety of functionalities for parsing and manipulating VCF files [7]; *cyvcf2*, a Python library for fast parsing and querying of VCF files [26]; *hts-nim*, a library for reading and writing high-throughput sequencing data [27].

Database systems offer powerful functionalities for storing, retrieving, and analyzing large volumes of data. Among these systems, *TileDB-VCF* is notable for its custom design tailored specifically for genetic data [24, 32]. It is an open-source C++ library that leverages the *TileDB Embedded* platform, a multi-dimensional array database system with high performance and the ability to scale for large volumes of genetic data due to its database roots. The core functionality of TileDB-VCF is its use of 3D sparse arrays for storing samples in a compressed and lossless manner by applying different compressors to VCF fields based on the data types and characteristics. It includes features for ingesting VCF files, performing genomic interval intersections, and slicing variant records by genomic regions across an arbitrary number of samples.

Other examples of database systems include *TheSNPpit* [14], a high-performance database system for managing large-scale SNP data, and *Breedbase* [20], an open-source web database that implements a novel data model for genotyping data.

Another work presented in [18] addresses similar challenges by exploring relational databases in an array model, resembling one of our eight models. The authors suggest that relational databases are effective for large genotypic datasets, balancing performance and storage, but do not compare them to BCF Tools. Our work takes a broader approach by exploring standard and extended RDBMS data models, including JSON structures, while preserving variant information. This emphasis on VCF fidelity sets our work apart, enabling richer analyses and greater flexibility for researchers. Finally, we thoroughly compare all our models to state-of-the-art tools, i.e., BCF Tools and TileDB VCF, using comprehensive workloads.

The study in [15] explored the use of relational databases for genomic data by storing individual genotypes in a so-called mapping table – a concept similar to our narrow models, where each variant-sample genotype is a separate row. Our study advances this line of research by providing a more extensive, large-scale evaluation, investigating eight different relational models and offering direct performance comparisons with state-of-the-art tools. While evaluation in [15] is limited to datasets of up to 100 million records, our work assesses scalability using over 1.2 billion genotype records.

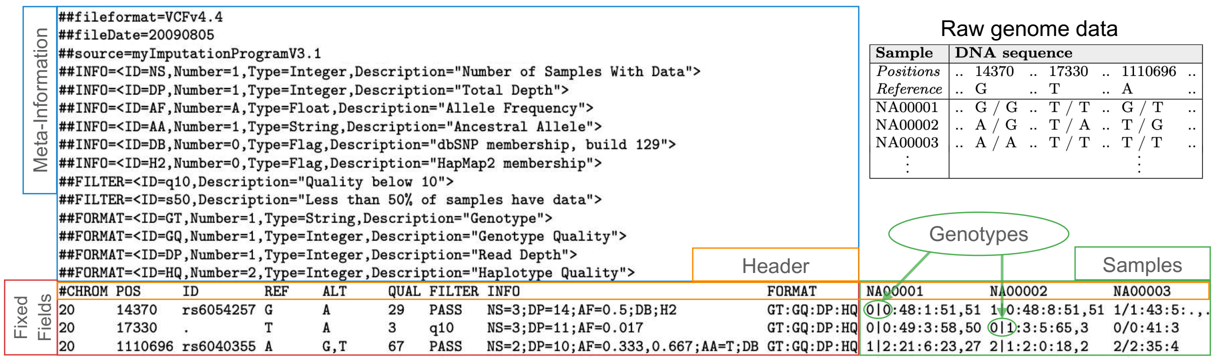
The work in [19] introduces a document-based approach that maps VCF files to JSON to integrate genetic variants with OWL-based ontologies in MongoDB, leveraging JSON's flexibility for handling semi-structured data. In a similar vein, our Full JSON and Full JSON ID models also use JSON, but within relational databases. However, our study takes a broader perspective by evaluating how various relational data models, including those incorporating JSON, can efficiently manage and query genomic variant data.

3 THE VCF FILE STRUCTURE

Genetic data is represented as sequences of letters denoting the bases. For simplicity, we focus on DNA, where these bases are A, T, C, and G. Other forms of genetic data follow the same scheme. The human genome has a pair of 23 chromosomes with about 3 billion bases. However, these numbers vary between species.

The *Variant Call Format* (VCF) has become the standard for storing large-scale gene sequence data from genotyping and DNA sequencing projects [10, 21, 22, 31]. VCF files are widely used in public genetic datasets, including the 1000 Genomes Project [6], as well as datasets for animals [2], plants [20], or microorganisms [13]. VCF files store genetic data in a compact form by exposing only positions that differ from a reference genome, which is a complete sequence of a particular genome assembly.

Figure 1 shows an example of raw genomic data and the corresponding VCF file. The Raw genome data table shows the raw DNA sequence of three samples: NA00001, NA00002, and NA00003, together with a reference sequence. In five positions, at least one sample differs from the reference genome. The parts that overlap with the reference in all samples are abbreviated with "..". Because



```
##fileformat=VCFv4.4
##fileDate=20090805
##source=myImputationProgramV3.1
##INFO=<ID=NS,Number=1,Type=Integer,Description="Number of Samples With Data">
##INFO=<ID=DP,Number=1,Type=Integer,Description="Total Depth">
##INFO=<ID=AF,Number=A,Type=Float,Description="Allele Frequency">
##INFO=<ID=AA,Number=1,Type=String,Description="Ancestral Allele">
##INFO=<ID=DB,Number=0,Type=Flag,Description="dbSNP membership, build 129">
##INFO=<ID=H2,Number=0,Type=Flag,Description="HapMap2 membership">
##FILTER=<ID=q10,Description="Quality below 10">
##FILTER=<ID=s50,Description="Less than 50% of samples have data">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
##FORMAT=<ID=HQ,Number=2,Type=Integer,Description="Haplotype Quality">
#CHROM POS     ID        REF    ALT     QUAL FILTER INFO                              FORMAT      NA00001        NA00002        NA00003
20     14370   rs6054257 G      A       29   PASS   NS=3;DP=14;AF=0.5;DB;H2           GT:GQ:DP:HQ 0|0:48:1:51,51 1|0:48:8:51,51 1/1:43:5:.,.
20     17330   .         T      A       3    q10    NS=3;DP=11;AF=0.017               GT:GQ:DP:HQ 0|0:49:3:58,50 0|1:3:5:65,3   0/0:41:3
20     1110696 rs6040355 A      G,T     67   PASS   NS=2;DP=10;AF=0.333,0.667;AA=T;DB GT:GQ:DP:HQ 1|2:21:6:23,27 2|1:2:0:18,2   2/2:35:4
```

Sample	DNA sequence						
Positions	..	14370	..	17330	..	1110696	..
Reference	..	G	..	T	..	A	..
NA00001	..	G / G	..	T / T	..	G / T	..
NA00002	..	A / G	..	T / A	..	T / G	..
NA00003	..	A / A	..	T / T	..	T / T	..
⋮						⋮	

Figure 1: Raw genome data and VCF file structure

sequences come in pairs, there are two variants at each position, e.g., G/G, one maternal and the other paternal. In the corresponding VCF file, we can distinguish three different blocks of information: meta-information, fixed fields, and samples.

The *meta-information* block provides essential context for interpreting the variant data, including file format version, reference genome, sample information, and detailed specifications for fields like `FILTER`, `INFO`, `FORMAT`, and others.

The *fixed fields* block contains general information about each variant, including its chromosome (`CHROM`), position (`POS`), and optional identifier (`ID`). The column `REF` stores the reference base/s, whereas `ALT` stores alternative alleles (bases). `QUAL`, `FILTER`, and `INFO` provide information about the quality score, filter status, and additional information, respectively. The optional `FORMAT` field is present if genotype/sample columns are included and defines the format used to store genotypes in the sample block.

The *samples* block stores genotype data for each sample at the positions listed in the fixed fields block. Each column is a sample, and each row a position. The corresponding data format is specified in the `FORMAT` field. Each cell contains two reads (one per parental chromosome): `0` matches the reference in the `REF` column, `1` the first alternative in the `ALT` column, `2` the second, and so on. For example, `0|0` means both alleles are reference, `0|1` is one reference and one alternative, and `1|1` means both are the first alternative.

4 RELATIONAL DATA MODELS FOR VCF

In this section, we introduce several data models we designed for VCF data: Wide, Narrow, Array Plain, and Full JSON. Additionally, we explored hybrid models, which we refer to as Wide JSON, Narrow JSON, Array JSON, and Full JSON ID.

VCF data are typically used for read-heavy analysis, and modifications of the data often require file regeneration from upstream pipelines [12]. Consequently, our data models prioritize read queries.

4.1 Meta-Information and Fixed Fields

Across all data models, we extract the meta-information from the VCF files and map it to a relational model by creating dedicated tables for each type of metadata. This structured approach ensures that the metadata is organized and easily accessible.

vcf_fixed_fields

ln	chrom	pos	id	ref	alt	qual	filter	info	format
1	20	14370	rs6054257	G	A	29	PASS	NS=3;DP=...	GT:GQ:DP:HQ
2	20	17330	.	T	A	3	q10	NS=3;DP=...	GT:GQ:DP:HQ
3	20	1110696	rs6040355	A	G,T	67	PASS	NS=2;DP=...	GT:GQ:DP:HQ

Figure 2: Fixed fields table

The fixed fields block of the VCF format is stored in a table named `fixed_fields` as shown in Figure 2. Each tuple represents a row in the fixed fields block, where each field is stored as an attribute: chromosome `chrom`, position `pos`, identifier `id`, reference base(s) `ref`, alternative base(s) `alt`, quality `qual`, filter status `filter`, additional information `info`, and format `format`.

Additionally, we add a column `ln`, which serves as a unique identifier. It corresponds to the line number in the VCF file. This field is crucial for maintaining the variants' order and cross-referencing between tables. We cannot use the `ID` field of the VCF standard because it is not guaranteed to be unique for all variants. Some VCF files may contain multiple variants with the same `ID`, especially in cases where the `ID` field is left blank or filled with a generic placeholder. Similarly, the combination of the `chrom` and `pos` fields from the VCF file might not be unique. In some cases, multiple distinct variants can occur at the same position on the same chromosome. These are often referred to as multi-allelic sites [28].

4.2 Samples Data Block

4.2.1 Wide Models. The *Wide model* organizes genomic samples relationally using a 2D matrix (variants as rows, samples as columns). Due to column limits, samples are split into chunks and stored in tables named `vcf_wide_samples_chunk_i`, where each row holds a position's genotype data. Figure 3a shows an example where each chunk can store at most two samples.

The *Wide JSON model* shown in Figure 3b extends the Wide model by storing genotypes as a JSON object. While straightforward in design, the Wide models do not scale well with large numbers of samples. As the sample count grows, additional `vcf_wide_samples_chunk_i` tables are needed, leading to increasingly complex and slower joins. Moreover, the model handles varying sample counts per genotype inefficiently.

4.2.2 Narrow Models. In the *Narrow model*, all sample data is stored in a single table, `vcf_narrow_samples` (Figure 4a), where each row vcf_wide_samples_chunk_1

ln	na00001	na00002
1	0|0:48:1:51,51	1|0:48:8:51,51
2	0|0:49:3:58,50	0|1:3:5:65,3
3	1|2:21:6:23,27	2|1:2:0:18,2

vcf_wide_samples_chunk_2

ln	na00003
1	1/1:43:5:.
2	0/0:41:3
3	2/2:35:4

(a) VCF Wide model

vcf_wide_json_samples_chunk_1

ln	na00001	na00002
1	{"DP": 1, "GQ": 48, "GT": 0/0, "HQ": 51,51}	· · ·
2	{"DP": 3, "GQ": 49, "GT": 0/0, "HQ": 58,50}	· · ·
3	{"DP": 6, "GQ": 21, "GT": 1/2, "HQ": 23,27}	· · ·

vcf_wide_json_samples_chunk_2

ln	na00003
1	{"DP": 5, "GQ": 43, "GT": 1/1}
2	{"DP": 3, "GQ": 41, "GT": 0/0}
3	{"DP": 4, "GQ": 35, "GT": 2/2}

(b) VCF Wide JSON model

Figure 3: VCF Wide and VCF Wide JSON models

represents a variant-sample pair, identified by the `ln` and the `s_id` attributes, with the genotype. This approach, known as "flattening" or "unpivoting," transforms wide-format data into a narrow format, with each row capturing one data point and its attributes. The Narrow model avoids the need for many joins in large datasets; however, the table storing the data can be very large.

The *Narrow JSON model* extends the Narrow model by encapsulating each combination of genotype and sample ID as a singular observation in JSON format, as in Figure 4b.

vcf_narrow_samples

ln	s_id	genotypes
1	NA00001	0|0:48:1:51,51
1	NA00002	1|0:48:8:51,51
1	NA00003	1/1:43:5:.
2	NA00001	0|0:49:3:58,50
2	NA00002	0|1:3:5:65,3
2	NA00003	0/0:41:3
3	NA00001	1|2:21:6:23,27
3	NA00002	2|1:2:0:18,2
3	NA00003	2/2:35:4

(a) VCF Narrow

vcf_narrow_json_samples

ln	s_id	genotypes
1	NA00001	{"DP": 1, "GQ": 48, "GT": 0/0, "HQ": 51,51}
1	NA00002	{"DP": 8, "GQ": 48, "GT": 1/0, "HQ": 51,51}
1	NA00003	{"DP": 5, "GQ": 43, "GT": 1/1}
2	NA00001	{"DP": 3, "GQ": 49, "GT": 0/0, "HQ": 58,50}
2	NA00002	{"DP": 5, "GQ": 3, "GT": 0/1, "HQ": 65,3}
2	NA00003	{"DP": 3, "GQ": 41, "GT": 0/0}
3	NA00001	{"DP": 6, "GQ": 21, "GT": 1/2, "HQ": 23,27}
3	NA00002	{"DP": 0, "GQ": 2, "GT": 2/1, "HQ": 18,2}
3	NA00003	{"DP": 4, "GQ": 35, "GT": 2/2}

(b) VCF Narrow JSON

Figure 4: VCF Narrow and VCF Narrow JSON models

4.2.3 *Array Models.*

The *Array Plain model* tackles the Narrow model's size issue by storing each VCF row as tuple in a table `vcf_array_genotypes`, where an array is used to store all samples' genotypes (cf. Figure 5a). To enable efficient access, a separate table `vcf_array_indices` is created that maps each sample ID to its index in the array.

vcf_array_genotypes

ln	genotypes
1	["0|0:48:1:51,51", "1|0:48:8:51,51", "1/1:43:5:."]
2	["0|0:49:3:58,50", "0|1:3:5:65,3", "0/0:41:3"]
3	["1|2:21:6:23,27", "2|1:2:0:18,2", "2/2:35:4"]

vcf_array_indices

s_id	e_id
NA00001	1
NA00002	2
NA00003	3

(a) VCF Array Plain

vcf_array_json_genotypes

ln	genotypes
1	[{"DP": 1, "GQ": 48, "GT": "0/0", "HQ": "51,51"}, . . .]
2	[{"DP": 3, "GQ": 49, "GT": "0/0", "HQ": "58,50"}, . . .]
3	[{"DP": 6, "GQ": 21, "GT": "1/2", "HQ": "23,27"}, . . .]

vcf_array_json_indices

s_id	e_id
NA00001	1
NA00002	2
NA00003	3

(b) VCF Array JSON

Figure 5: VCF Array Plain and VCF Array JSON models

The *Array JSON model* extends the array model by storing genotypes as an array of JSON objects, each encapsulating a distinct genotype of a sample (cf. Figure 5b).

4.2.4 *JSON Models.*

Besides flat (wide and narrow) and array-based models, we explored JSON-based data models for encoding genotypes. JSON's flexibility within a relational schema allows complex queries via database JSON functions and avoids the sparsity and storage overhead of dedicated relational columns for annotations. The alternative of defining a compressed custom data type would require additional decompression steps for retrieval.

The *Full JSON model* stores both sample IDs and genotypes together in a JSON object (see Figure 6a). Each row of the `vcf_json_samples` table contains a JSON array, where each element is a JSON object with a sample ID and its genotype data. Unlike the Array JSON model, which stores genotypes as JSON objects in an array without sample IDs, this model explicitly includes the sample ID within each genotype entry.

In the *Full JSON ID model* in Figure 6b, each genotype is a single JSON object in the `vcf_js_id_samples` table with a separate table for sample ID mapping, similar to the Array models. This additional table, `vcf_js_id_indices`, provides a mapping between the sample IDs and their indices in the genotypes array.

vcf_json_samples

ln	sample_genotype
1	{{"s_id": "NA00001", "genotype": {"DP": "1", "GT": "0|0", "GQ": "48", "HQ": "51,51"}}, . . .}
2	{{"s_id": "NA00001", "genotype": {"DP": "3", "GT": "0|0", "GQ": "49", "HQ": "58,50"}}, . . .}
3	{{"s_id": "NA00001", "genotype": {"DP": "6", "GT": "1|2", "GQ": "21", "HQ": "23,27"}}, . . .}

(a) VCF Full JSON model

vcf_js_id_samples

ln	genotypes
1	{{"DP": "1", "GT": "0|0", "GQ": "48", "HQ": "51,51"}, . . .}
2	{{"DP": "3", "GT": "0|0", "GQ": "49", "HQ": "58,50"}, . . .}
3	{{"DP": "6", "GT": "1|2", "GQ": "21", "HQ": "23,27"}, . . .}

vcf_js_id_indices

s_id	e_id
NA00001	1
NA00002	2
NA00003	3

(b) VCF Full JSON ID model

Figure 6: VCF Full JSON and VCF Full JSON ID models

4.3 Summary of Data Models

Table 1 summarizes the different genotype storage models above. It categorizes each model based on the data type used to represent genotypes (strings or JSON objects) and the method for associating genotypes with their Sample ID (same table or separate table). The models vary from direct translations (Wide models) and more RDBMS-centric reorganizations (Narrow models) to models using arrays and JSON. Moreover, different ways of storing sample IDs are considered, namely, in the same table as column names or column values, and in a separate table.

Table 1: Summary of the proposed models

	Sample ID in separate table	**Sample ID in same table**
Genotype as string	Array Plain	Narrow, Wide
Genotype as JSON	Array JSON, Full JSON ID	Narrow/Wide/Full JSON

5 EXPERIMENTAL EVALUATION

5.1 Setup

5.1.1 *Environment.*

We conducted the experiments on a server equipped with a 64-core Intel® Xeon® Gold 6246R CPU @ 3.40GHz. The server's configuration included 92 GiB of main memory and high-capacity hard disk drives (HDDs) for data storage, offering several terabytes of capacity.

The system was running on Ubuntu 22.04.4 LTS (Jammy). PostgreSQL 13.2 served as the primary database management system configured using PGTune (https://pgtune.leopard.in.ua). We used Python 3.10.14 for the scripts and tests.

5.1.2 Dataset. In the experiments, we used the chromosome 22 VCF file of the 1000 Genomes Project [6]. This VCF file contains a detailed record of human genetic variation with extensive coverage across various populations. We expanded the number of samples in this dataset by 400%, resulting in 12.808 samples, aligning with the sample size considered in the CHRIS study [25]. We did not modify the number of variants in the dataset. The variant distribution and allele frequencies present in the original dataset were maintained.

5.1.3 Workloads. We used three different *workload patterns*: *Access* retrieves all VCF entries for specified sample IDs and/or variant ranges, with "%" indicating the selected amount and "All" meaning no filter. *Variant Filtering* builds on *Access* by also filtering variants based on fixed fields, returning only those that meet the condition. *Sample Filtering* extends *Access* by filtering on genotype values, returning, for each variant, only the sample IDs that satisfy the condition. These are the core operations directly applicable to raw VCF data for downstream analyses, simulating various real-world scenarios, as supported by common bioinformatics practices [1, 8]. For our experiments, we set a timeout of one hour, repeated each query ten times, and calculated the truncated mean execution time.

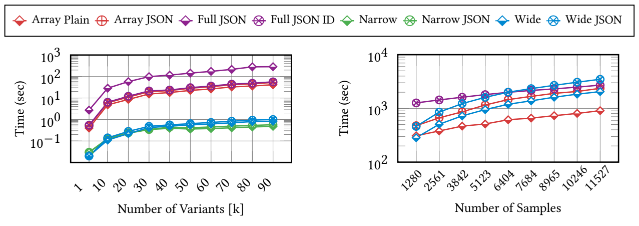
5.1.4 Implementation Details. We benchmarked TileDB-VCF 0.35.0 (with TileDB Embedded 2.24.0) and BCF Tools 1.20 with Tabix (htslib) 1.20. We use the following column data types: genotype strings use `VARCHAR` to accommodate the significant variations in their length, and identifiers such as sample ID use `INT` to ensure a sufficient range. Most tables use `line_nb` as the primary key (PK), while Narrow models use a composite PK (`line_nb`, `sample_id`), and Array/Full JSON models use `element_id`. PostgreSQL automatically indices PKs; additionally, we created a multicolumn index on `chrom` and `pos` in `fixed_fields` to optimize filtering. For JSON models, we used `JSONB` for its compression and advanced features. We refer to the source code for further details.

5.2 Overall Query Runtime Performance

In the first set of experiments, we evaluate which model in Section 4 performs best in general and is the most robust for query runtime using four workload settings (see Figure 7).

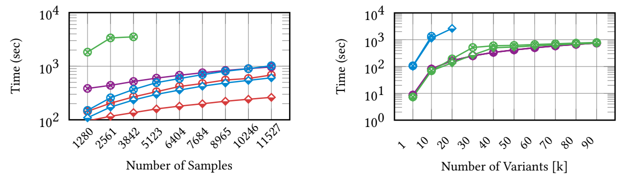
Figure 7a shows the results for selecting a single sample across increasing variant ranges (data access pattern). The Array and Full JSON ID models have low initial runtimes with moderate growth. The Narrow and Wide models perform exceptionally well, with only modest runtime increases. This efficiency is due to the Narrow model's storing each variant-sample pair as a tuple (cf. Figure 4) and the Wide models' columnar structure (cf. Figures 3a and 3b), where retrieving data for one specific sample involves directly accessing the corresponding column. Both models enable fast retrieval for a specific sample across variants.

Figure 7b shows model performance for retrieving increasing numbers of samples, without restricting the variant range. The



(a) Range of variants for a sample

(b) Set of samples across all variants



(c) All variants where $AF \geq 0.01$ for a set of samples

(d) All homozygous samples for a range of variants

Figure 7: Query runtime evaluation of the proposed models

Array models scale best, as they store all sample genotypes for a position in a single array (cf. Figure 5a), enabling efficient multi-sample retrieval. The Full JSON ID model, though starting with a higher initial runtime, also scales well due to its structural similarity to the Array models (cf. Figure 6b). In contrast, the Narrow and Full JSON models did not complete within the timeout for any sample size: the Narrow models perform poorly with more samples due to their long format, while the Full JSON model's nested structure makes sample access increasingly costly.

In the next experiment, we show the behavior for the variant filtering workload pattern in Figure 7c. We use a condition on the fixed fields (i.e., AF ≥ 0.01) with a selectivity of 24.36% for an increasing number of samples, and do not use a restriction on the range of variants. The Full JSON ID model shows an increasing runtime with increasing sample sizes. The Wide models maintain a lower runtime. The Array models demonstrate low and stable runtime, with the Array Plain model being the most efficient. The Narrow model timed out when retrieving 3,482 samples, highlighting a scalability problem. Additionally, the Full JSON and Narrow JSON models timed out across all sample sizes.

In the next experiment in Figure 7d, we explore the sample filtering workload pattern for an increasing range of variants, i.e., we select the sample IDs and variants for which a condition on the genotypes is met. Specifically, we select, for an increasing range of variants, all homozygous samples, where the selectivity on the genotype is 86.86%. The runtime of the Narrow models significantly increases for larger ranges of variants. Additionally, the Wide model times out at 10K variants, while the Wide JSON model times out at 20K variants. In contrast, the Array and JSON models show low and very similar runtimes.

In summary, we can observe that Array Plain is the overall best and most robust model in terms of runtime. For an increasing number of samples, Narrow models and JSON deteriorate substantially in performance. Similarly, Wide models suffer in performance and are competitive only if up to 10% of the samples are used. Full JSON ID is consistently worse than Array Plain. For multiple variants, the

Table 2: Runtime comparison in seconds without index (left column) and with index (right column); * indicates a timeout

Workload	Array Plain		Array JSON		Full JSON		Full JSON ID		Narrow		Narrow JSON		Wide		Wide JSON	
W1-A (All samples - 1 variant)	2.5	0.4	1.7	1.1	13.6	3.7	291.2	280.6	20.0	0.7	4.6	1.2	451.0	327.9	437.4	318.1
W1-B (1 sample - All variants)	336.6	333.6	526.2	518.0	*	*	*	*	2107.2	1054.8	4123.2	1088.4	*	*	*	*
W2-A (Sample filter - 1 variant)	7.0	2.3	4.9	2.8	0.6	0.2	306.9	0.4	2791.6	9.3	2835.6	6.9	442.8	0.2	429.6	0.2
W2-B (Sample filter - 10K variants)	322.5	8.2	*	10.7	9.6	9.5	322.5	8.2	*	150.6	*	177.7	*	*	*	*
W3-A (Variant filter - 1 sample)	87.6	49.8	151.2	114.0	310.2	282.0	514.2	326.5	40.8	31.2	96.0	47.0	675.6	17.4	655.3	19.7
W3-B (Variant filter - 1k samples)	118.8	65.2	141.6	53.0	*	*	236.1	129.4	376.8	294.3	*	*	725.2	17.9	719.0	41.1

difference between the approaches generally remains the same. For sample filtering workloads – unlike variant filtering – the flexibility of JSON and the faster parsing of complex annotations (as in Full JSON ID) outweigh the cost of increased storage. This is primarily because direct key-based access to values offers a significant advantage for sample filtering.

The Effect of Indexing on Data Access. Table 2 compares the query runtime without and with indexing across our different models and workloads. Indexing significantly reduces runtime for variant-centric access (*W1-A*) and single-variant sample filtering (*W2-A*) across most models, demonstrating its effectiveness for selective queries based on variant characteristics. This improvement is due to the multi-column index on `chrom` and `pos` in the `fixed_fields` table. Indexing also improves performance for variant filtering within a specified sample subset (*W3-A* and *W3-B*) by enabling direct access to relevant records. However, it offers less benefit for large range queries by sample ID (*W1-B*), emphasizing the importance of query selectivity. Narrow models benefit significantly from indexing due to their fine-grained structure, while Wide models see smaller gains because of their multi-table design. JSON-based models show variable performance due to JSON parsing overhead. In some workloads, indexing is essential for practical runtimes—as evidenced by timeouts in its absence (e.g., *W2-B*) – highlighting its necessity for handling complex filtering tasks efficiently.

5.3 Comparative Analysis with State-of-the-Art

We benchmarked the performance of our best-performing model, the Array Plain model, with the state-of-the-art tools, BCF Tools and TileDB-VCF. We evaluated query runtimes across all workload patterns with varying selectivities for variants and samples. The results are shown in Figures 8, 9, and 10. Each heatmap is color-coded, indicating the runtime of the best approach. Each cell contains the best-performing approach at the top and the second-best at the bottom, with the number in between indicating the speed-up of the first compared to the second. The axes show the selectivities for samples and variants (data access), where "All" indicates there is no restriction on that dimension and "One" indicates a single variant or sample. For sample filtering, the selectivity for the genotype is constant and 86.86%, whereas for variant filtering, the selectivity for the filter on the fixed fields is constant with 24.36%.

For completeness, we also evaluated the performance of other models (Full JSON ID and Wide models). These evaluations confirmed the findings of Section 5.2, i.e., Full JSON ID exhibited slightly faster performance than Array Plain for single-variant workloads. However, it was slower in all other cases. Wide models demonstrated superior performance for a very small number of samples (up to 10%) but were considerably outperformed by Array Plain for a larger number of samples.

Across all heatmaps, we observed a general trend of increased runtime with larger selectivities that generate larger outputs. TileDB-VCF only demonstrated competitive performance for workloads involving a single sample and is much slower in all other cases. For the data access workload pattern shown in Figure 8, the Array Plain model performs best when retrieving very few or all samples, while BCF Tools, an optimized tool for these workloads, performs slightly better for other selectivities. BCF Tools demonstrated a significant advantage when retrieving a single variant and specific samples, benefiting from its optimized Tabix indexing.

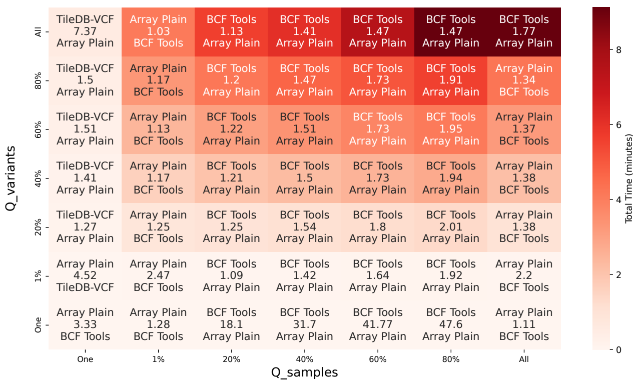


Figure 8: Data access by sample and variant

For the sample filtering workload pattern in Figure 9, we have a similar picture, where BCF Tools and Array Plain perform the best. For variant filtering in Figure 10, Array Plain consistently outperformed BCF Tools, particularly for small subsets of samples, due to its efficient filtering process using the fixed fields table prior to the join with the genotypes.

In summary, in the experiments in Figures 8 and 9, BCF Tools shows better performance in many cases due to specific optimizations for data access and sample filtering workloads, relying on tools like Tabix for efficient indexing and retrieval. A more detailed analysis reveals that Array Plain outperforms BCF Tools as the number of samples increases, which is increasingly common in newer studies due to cheaper and faster sequencing technologies [16]. In contrast, when the filtering condition is based on variants, Array Plain performs best in almost all cases, and BCF Tools is never the top performer (Figure 10).

Mixed Workload Analysis. In this experiment we compared Array Plain and BCF Tools on mixed workloads composed of data access, sample filtering, and variant filtering queries. A subset of

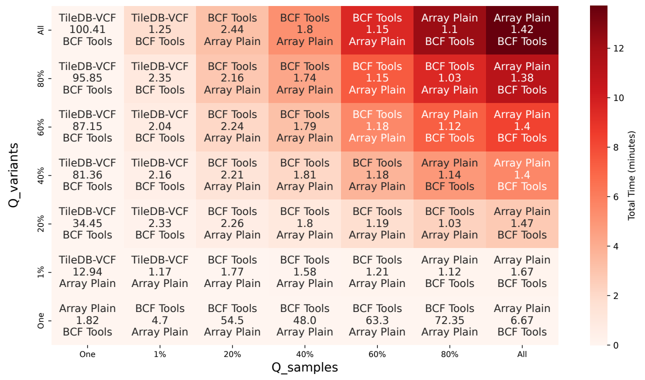


Figure 9: Sample filtering

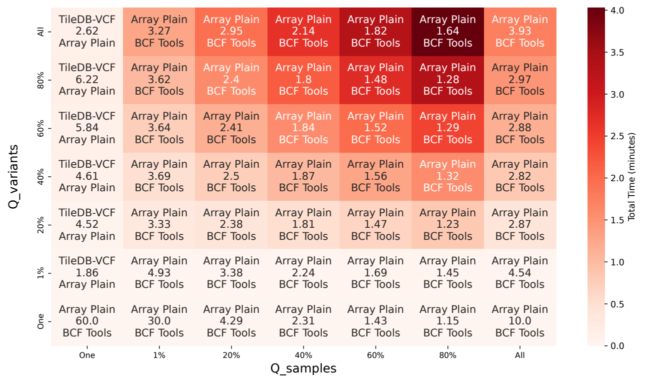


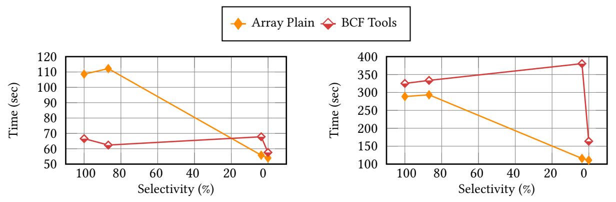
Figure 10: Variant filtering

the experimental results is presented in Table 3. Array Plain performs best when variant filtering is the dominant workload, thanks to its columnar-like storage of fixed fields, which is optimized for scanning variant-level annotations. It also performs well in sample filtering, particularly when filtering across all samples, as genotypes are stored contiguously within a single array. However, unnesting this array becomes a bottleneck when accessing specific sample genotypes. In contrast, BCF Tools excels in data access and sample filtering due to efficient Tabix indexing. Its performance degrades with increased variant filtering, as it requires parsing individual records, whereas Array Plain benefits from direct access to fixed fields, offering a clear speed advantage.

Table 3: Mixed workload performance of Array Plain and BCF Tools

Access	Sample	Variant	Array	BCF	Access	Sample	Variant	Array	BCF
80 %	10 %	10 %	**26**	106	33 %	0 %	67 %	81	**51**
67 %	33 %	0 %	**12**	120	10 %	80 %	10 %	**31**	101
67 %	0 %	33 %	**46**	86	10 %	10 %	80 %	121	**11**
33 %	67 %	0 %	**15**	117	0 %	67 %	33 %	**46**	86
33 %	33 %	34 %	**26**	106	0 %	33 %	67 %	122	**10**

Varying the Query Selectivity. We investigated how the selectivity of the genotype filter in sample filtering affects various approaches. We used two cells in Figure 9 for which the selectivity of the genotype filter was 86.86%. We varied this selectivity using different genotype filters. The results are shown in Figure 11. In Figure 11a, we see the difference between the Array Plain model and BCF Tools reduces for smaller selectivities, and in Figure 11b, we see that the Array Plain model is much more efficient with smaller selectivities compared to BCF Tools. This indicates that for smaller selectivities, the Array Plain model has an advantage over BCF Tools.



(a) 40% samples and 20% variants (b) 80% samples and 40% variants

Figure 11: Sample filtering with different genotype filters

5.4 Storage and Memory Scalability

We evaluated the storage requirements of our models and state-of-the-art approaches. The results, including the size of the compressed and uncompressed VCF file, are shown in Figure 12. TileDB 200 refers to a configuration with a sample batch size of 200 during ingestion. We found that 200 was the largest batch size that avoided excessive memory usage and system crashes while minimizing disk space. Wide and Narrow models have a very large storage requirement. Other DBMS approaches, including TileDB 200, are in the order of compressed VCF data, much smaller than the uncompressed VCF.

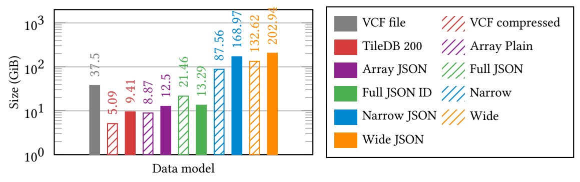


Figure 12: Storage size Comparison for different data models

We also measured the peak memory usage of all the approaches. DBMS approaches offer constant, very low main memory requirements similar to BCF tools. However, TileDB-VCF requires a very large amount of main memory. Whereas other approaches are in the order of 10 to 100 MiB, TileDB-VCF for the same workload and dataset requires tens of GiB.

In summary, the Array Plain model is overall the most efficient and robust approach in terms of query runtime among all RDBMS approaches. Similarly, it also requires the least amount of storage. In comparison to BCF Tools, a specialized tool optimized for these tasks, the Array Plain approach offers competitive performance and robustness, while making available all the features of RDBMS to genetic data analysis.

6 DISCUSSION

Our study provides a comprehensive comparison of different approaches for storing and querying VCF data. Our results highlight

Table 4: Comparison of RDBMS, VCF/BCF Tools, and TileDB-VCF for managing and storing genetics data.

Feature	RDBMS	VCF/BCF Tools	TileDB-VCF
Data Structure	Tables with relations	Flat files (TSV)	Multi-dimensional dense arrays
Data Integrity	Constraints and data types	Prone to errors	Data integrity checks
Scalability	Efficient with large data	Bulky with large data	Scales well, fast retrieval
Querying	Powerful capabilities	Requires specialized tools	Efficient with specific subsets
Data Integration	Seamless with bio-data sources	Limited capabilities	Limited compared to RDBMS
Standardization	Standardized models (HGNC[3])	No inherent standardization	User-defined schemas
Security	Robust access control	Limited security features	Supports access control
Backup/ Recovery	Built-in mechanisms	Requires separate backup	Requires configuration
Collaboration	Multiple user access	Reliance on specific tools	Limited compared to RDBMS
Other Considerations (Downstream Analysis)			
GWAS/phenotyping	Efficient format conversion/join	Requires data conversion	Efficient for specific subsets
Filtering/Annotation	Built-in	Requires exporting/re-importing	Within the array structure

the trade-offs between simplicity, efficiency, and flexibility in genomic data representation. Table 4 collects these trade-offs, comparing key features of RDBMS, VCF/BCF Tools, and TileDB-VCF. This comparison underscores each approach's relative strengths and weaknesses in the context of genomic data handling.

RDBMSs offer significant advantages beyond read-optimized storage, enabling complex SQL queries for flexible filtering and aggregation across variant and sample attributes. Similar capabilities are cumbersome with file-based tools, particularly in scenarios involving the integration of external datasets such as phenotypes, clinical information, or other genomic annotations [4, 19]. RDBMSs also provide benefits for data integration pipelines in multiomics analysis, which is the new trend in studying biological processes, where genotypes are stored and queried together with other types of biomedical data such as proteomics, lipidomics, and others [3, 5]. Furthermore, for large-scale cohort studies, the robust indexing features of the RDBMS bring about efficiency in data retrieval, enhancing speed in read-heavy scenarios and providing performance advantages for querying specific data subsets. While the core variant and genotype information are typically static, updates can arise in associated metadata or when participants decide to withdraw from a cohort. Such updates are straightforward in RDBMS, while VCF files have to be created from scratch. Collectively, these benefits establish a full-fledged RDBMS as a robust and versatile platform for the diverse analytical demands of genomics and multiomics.

Our findings highlight unique requirements when selecting a data storage and query method. Although some models may outperform others in certain cases, the optimal choice depends on the specific features most crucial for the given task. Building on these insights, we examine the characteristics of various data models:

Simplicity vs Complexity. The Wide models, while straightforward, are not suitable for large datasets due to their simplistic two-dimensional approach. RDBMS limitations, i.e., the maximum number of columns and fixed page size, can pose challenges in the Wide models. Despite a more compact format, the Narrow models result in large tables as the number of genotype-sample pairs increases, posing challenges in handling large genetic datasets.

Performance Variance. BCF Tools, while occasionally demonstrating competitive execution times, show a high variance in performance, with certain queries taking la ong duration. TileDB-VCF, on the other hand, performs well with some of the lower execution times recorded. However, its performance is compromised by frequent timeouts during various queries, with scalability challenges or constraints when dealing with specific data configurations.

Balanced Performance. By extending the Array models and requiring a separate table for sample IDs, the Full JSON ID model provides consistent performance across a range of queries, with a good balance between efficiency and scalability. For tasks on large datasets, the Full JSON ID model offers easier data access and management without significantly increasing storage requirements.

Efficiency vs Flexibility. The Array models demonstrated significant scalability as the sample size increases. These models address the size issue of the Narrow model by storing genotypes in a single array, offering better performance at the cost of less efficient data access. The JSON model is simpler due to a compact encapsulation with a straightforward structure. However, with larger datasets, the JSON model does not scale as others do due to its inherent verbosity. This may lead to increased storage requirements and performance bottlenecks if individual genotype access is frequently required.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we utilize RDBMS to improve the efficiency and flexibility for storing and querying large-scale genetic data. We map the structure of VCF files into different models, exploring various RDBMS concepts (standard relational columns, arrays, and JSON) and considering the characteristics of typical query workloads. Our experimental results highlight the trade-offs between efficiency, scalability, and accessibility in genomic data management. RDBMS offers a promising approach to address scalability issues in genetic data analysis, leveraging their robust data management capabilities and extensive built-in query processing tailored for large-scale data.

Future research points in several directions. To improve runtime performance and scalability, we will investigate cloud technologies, distributed computing, and advanced index structures. Furthermore, we will investigate the seamless integration of our data models with existing pipelines by adhering to standards and APIs.

ACKNOWLEDGMENTS

We thank Christian Fuchsberger, David Emmert, and Michele Filosi at Eurac for valuable feedback. This work was funded in part by the Autonomous Province of Bozen-Bolzano, call Joint Projects South Tyrol – Germany, project DyHealthNet – CUP: I53C22002980003.

[3]https://www.genenames.org/about/

REFERENCES

[1] Amira Al-Aamri, Syafiq Kamarul Azman, Gihan Daw Elbait, Habiba Alsafar, and Andreas Henschel. 2023. Critical assessment of on-premise approaches to scalable genome analysis. *BMC bioinformatics* 24, 1 (2023), 354.

[2] Ellie E Armstrong, Anubhab Khan, Ryan W Taylor, Alexandre Gouy, Gili Greenbaum, Alexandre Thiéry, Jonathan T Kang, Sergio A Redondo, Stefan Prost, Gregory Barsh, et al. 2021. Recent evolutionary history of tigers highlights contrasting roles of genetic drift and selection. *Molecular Biology and Evolution* 38, 6 (2021), 2366–2379.

[3] Liuyang Cai, Jun Qiao, Ruixin Zhou, Xinyi Wang, Yelan Li, Lei Jiang, Qiangwei Zhou, Guoliang Li, Tao Xu, and Yuliang Feng. 2025. EXPRESSO: a multi-omics database to explore multi-layered 3D genomic organization. *Nucleic Acids Research* 53, D1 (2025), D79–D90.

[4] Yi-Ming Chen, Tzu-Hung Hsiao, Ching-Heng Lin, and Yang C Fann. 2025. Unlocking precision medicine: clinical applications of integrating health records, genetics, and immunology through artificial intelligence. *Journal of Biomedical Science* 32, 1 (2025), 16.

[5] Ana Conesa and Stephan Beck. 2019. Making multi-omics data accessible to researchers. *Scientific Data* 6, 1 (October 2019), 251. https://doi.org/10.1038/s41597-019-0258-4

[6] The 1000 Genomes Project Consortium. 2015. A global reference for human genetic variation. *Nature* 526 (2015), 68–74. https://doi.org/10.1038/nature15393

[7] Daniel E Cook and Erik C Andersen. 2017. VCF-kit: assorted utilities for the variant call format. *Bioinformatics* 33, 10 (2017), 1581–1582.

[8] Petr Danecek, James K Bonfield, Jennifer Liddle, John Marshall, Valeriu Ohan, Martin O Pollard, Andrew Whitwham, Thomas Keane, Shane A McCarthy, Robert M Davies, et al. 2021. Twelve years of SAMtools and BCFtools. *Gigascience* 10, 2 (2021), giab008.

[9] Robert H Dolin, Shaileshbhai R Gothi, Aziz Boxwala, Bret SE Heale, Ammar Husami, James Jones, Himanshu Khangar, Shubham Londhe, Frank Naeymi-Rad, Soujanya Rao, et al. 2021. vcf2fhir: a utility to convert VCF files into HL7 FHIR format for genomics-EHR integration. *BMC bioinformatics* 22 (2021), 1–11.

[10] Ze-Zhen Du, Jia-Bao He, and Wen-Biao Jiao. 2024. A comprehensive benchmark of graph-based genetic variant genotyping algorithms on plant genomes for creating an accurate ensemble pipeline. *Genome Biology* 25, 1 (2024), 91.

[11] Holger Fröhlich, Rudi Balling, Niko Beerenwinkel, Oliver Kohlbacher, Santosh Kumar, Thomas Lengauer, Marloes H Maathuis, Yves Moreau, Susan A Murphy, Teresa M Przytycka, et al. 2018. From hype to reality: data science enabling personalized medicine. *BMC medicine* 16 (2018), 1–15.

[12] Erik Garrison, Zev N Kronenberg, Eric T Dawson, Brent S Pedersen, and Pjotr Prins. 2022. A spectrum of free software tools for processing the VCF variant call format: vcflib, bio-vcf, cyvcf2, hts-nim and slivar. *PLoS computational biology* 18, 5 (2022), e1009123.

[13] Jeffry M Gaston, Eric J Alm, and An-Ni Zhang. 2024. Fast and accurate variant identification tool for sequencing-based studies. *BMC biology* 22, 1 (2024), 90.

[14] Eildert Groeneveld and Helmut Lichtenberg. 2016. TheSNPpit—A High Performance Database System for Managing Large Scale SNP Data. *PLOS ONE* 11, 10 (2016), e0164043. https://journals.plos.org/plosone/article/file?id=10.1371/journal.pone.0164043&type=printable

[15] Steffen Janetzki, Magnús Rafn Tiedemann, and Hardik Balar. 2015. *Genome data management using RDBMSs*. Technical Report. Technical Report. https://www.researchgate.net/profile/Hardik-Balar/publication/280232082_Genome_Data_Management_using_RDBMSs/

[16] Kenneth Katz, Oleg Shutov, Richard Lapoint, Michael Kimelman, J Rodney Brister, and Christopher O'Sullivan. 2022. The Sequence Read Archive: a decade more of explosive growth. *Nucleic acids research* 50, D1 (2022), D387–D390.

[17] Heng Li. 2011. Tabix: fast retrieval of sequence features from generic TAB-delimited files. *Bioinformatics* 27, 5 (2011), 718–719.

[18] Ryan N Lichtenwalter, Katerina Zorina-Lichtenwalter, and Luda Diatchenko. 2017. Genotypic Data in Relational Databases: Efficient Storage and Rapid Retrieval. In *Advances in Databases and Information Systems: 21st European Conference, ADBIS 2017, Nicosia, Cyprus, September 24-27, 2017, Proceedings 21*. Springer, Nicosia, Cyprus, 408–421. https://link.springer.com/content/pdf/10.1007/978-3-319-66917-5_27.pdf

[19] Jian Liu, Zhi Qu, Mo Yang, Jialiang Sun, Shuhui Su, and Lei Zhang. 2019. Jointly integrating VCF-based variants and OWL-based biomedical ontologies in MongoDB. *IEEE/ACM transactions on computational biology and bioinformatics* 17, 5 (2019), 1504–1515.

[20] Nicolas Morales, Guillaume J Bauchet, Titima Tantikanjana, Adrian F Powell, Bryan J Ellerbrock, Isaak Y Tecle, and Lukas A Mueller. 2020. High density genotype storage for plant breeding in the Chado schema of Breedbase. *PLoS One* 15, 11 (2020), e0240059.

[21] Nature Genetics Nature and Nature Reviews Genetics. 2021. Milestones in Genomic Sequencing. https://www.nature.com/immersive/d42859-020-00099-0/

[22] Hilde Nybom and Gunārs Lācis. 2021. Recent large-scale genotyping and phenotyping of plant genetic resources of vegetatively propagated crops. *Plants* 10, 2 (2021), 415.

[23] Tolulope O Olorunsogo, Obe Destiny Balogun, Oluwatoyin Ayo-Farai, Oluwatosin Ogundairo, Chinedu Paschal Maduka, Chiamaka Chinaemelum Okongwu, and Chinyere Onwumere. 2024. Bioinformatics and personalized medicine in the US: A comprehensive review: Scrutinizing the advancements in genomics and their potential to revolutionize healthcare delivery. *World Journal of Advanced Research and Reviews* 21, 01 (2024), 335–351. https://doi.org/10.30574/wjarr.2024.21.1.0016

[24] Stavros Papadopoulos, Kushal Datta, Samuel Madden, and Timothy Mattson. 2016. The tiledb array data storage manager. *Proceedings of the VLDB Endowment* 10, 4 (2016), 349–360.

[25] Cristian Pattaro, Martin Gögele, Deborah Mascalzoni, Roberto Melotti, Christine Schwienbacher, Alessandro De Grandi, Luisa Foco, Yuri D'elia, Barbara Linder, Christian Fuchsberger, et al. 2015. The Cooperative Health Research in South Tyrol (CHRIS) study: rationale, objectives, and preliminary results. *Journal of translational medicine* 13 (2015), 1–16.

[26] Brent S Pedersen and Aaron R Quinlan. 2017. cyvcf2: fast, flexible variant analysis with Python. *Bioinformatics* 33, 12 (2017), 1867–1869.

[27] Brent S Pedersen and Aaron R Quinlan. 2018. hts-nim: scripting high-performance genomic analyses. *Bioinformatics* 34, 19 (2018), 3387.

[28] Benjamin A Pierce. 2017. *Genetics: A conceptual approach.* Macmillan Higher Education, New York, NY.

[29] SAMtools. 2024. BCFtools by SAMtools. https://github.com/samtools/bcftools. Accessed: 2025-06-21.

[30] Gary Saunders, Michael Baudis, Regina Becker, Sergi Beltran, Christophe Béroud, Ewan Birney, Cath Brooksbank, Søren Brunak, Marc Van den Bulcke, Rachel Drysdale, et al. 2019. Leveraging European infrastructures to access 1 million human genomes by 2022. *Nature Reviews Genetics* 20, 11 (2019), 693–701.

[31] Tomoya Tanjo, Yosuke Kawai, Katsushi Tokunaga, Osamu Ogasawara, and Masao Nagasaki. 2021. Practical guide for managing large-scale human genome data in research. *Journal of Human Genetics* 66, 1 (2021), 39–52.

[32] TileDB-Inc. 2024. TileDB-VCF: Efficient variant-call data storage and retrieval. https://github.com/TileDB-Inc/TileDB-VCF. Accessed: 2025-06-21.

[33] Clare Turnbull, Richard H Scott, Ellen Thomas, Louise Jones, Nirupa Murugaesu, Freya Boardman Pretty, Dina Halai, Emma Baple, Clare Craig, Angela Hamblin, et al. 2018. The 100 000 Genomes Project: bringing whole genome sequencing to the NHS. *Bmj* 361 (2018), k1687. https://doi.org/10.1136/bmj.k1687

[34] Emil Uffelmann, Qin Qin Huang, Nchangwi Syntia Munung, Jantina De Vries, Yukinori Okada, Alicia R Martin, Hilary C Martin, Tuuli Lappalainen, and Danielle Posthuma. 2021. Genome-wide association studies. *Nature Reviews Methods Primers* 1, 1 (2021), 59.