

Demonstration of GenoMetric Query Language

Stefano Ceri, Arif Canakoglu, Andrea Gulino, Abdulrahman Kaitoua*

Marco Masseroli, Luca Nanni, Pietro Pinoli

DEIB, Politecnico di Milano

first.last@polimi.it

ABSTRACT

In the last ten years, genomic computing has made gigantic steps due to Next Generation Sequencing (NGS), a high-throughput, massively parallel technology; the cost of producing a complete human sequence dropped to 1000 US\$ in 2015 and is expected to drop below 100 US\$ by 2020. Several new methods have recently become available for extracting heterogeneous datasets from the genome, revealing data signals such as variations from a reference sequence, levels of expression of coding regions, or protein binding enrichments ('peaks') with their statistical or geometric properties. Huge collections of such datasets are made available by large international consortia.

In this new context, we developed GenoMetric Query Language (GMQL), a new data extraction and integration language. GMQL supports queries over thousands of heterogeneous datasets; as such, it is key to genomic data analysis. GMQL queries are executed on the cloud, after being translated and optimized; our best deployment uses Spark over Hadoop. Datasets are described by the Genomic Data Model (GDM), which provides interoperability between many data formats; GDM combines abstractions for genomic region data with the associated experimental, biological and clinical metadata.

GMQL is targeted to the bio-informatics community for facilitating data exploration and for integrating data extraction and data analysis; this demonstration highlights its usability and expressive power. We show GMQL at work from a Web-based user interface and from a language embedding (Python).

CCS CONCEPTS

• **Applied computing** → **Computational genomics**; • **Theory of computation** → **Data modeling**;

KEYWORDS

Genomic Data Management, Query Language

ACM Reference Format:

Stefano Ceri, Arif Canakoglu, Andrea Gulino, Abdulrahman Kaitoua, Marco Masseroli, Luca Nanni, Pietro Pinoli. 2018. Demonstration of GenoMetric Query Language. In *The 27th ACM International Conference on Information and Knowledge Management (CIKM '18)*, October 22–26, 2018, Torino, Italy. ACM, New York, NY, USA, 4 pages. <https://doi.org/10.1145/3269206.3269217>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CIKM '18, October 22–26, 2018, Torino, Italy

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6014-2/18/10...\$15.00
<https://doi.org/10.1145/3269206.3269217>

ID	ATTRIBUTE	VALUE
1	antibody_target	H3K4me1
1	cell	K562
1	data_type	ChIP-seq
1	treatment	none
2	antibody_target	CTCF
2	cell	K562
2	data_type	ChIP-seq

ID	(CHR, LEFT, RIGHT, STRAND)	P_VALUE
1	(chr1, 21070, 22375, *)	0.00025
1	(chr1, 22700, 24300, *)	0.00057
1	(chr2, 51050, 52903, *)	0.01500
2	(chr1, 20550, 21900, *)	0.01204
2	(chr2, 51700, 53140, *)	0.00020

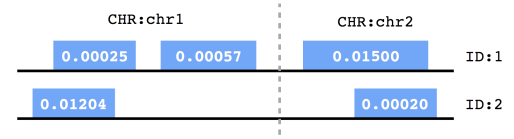


Figure 1: Dataset with two samples in GDM format.

1 BACKGROUND

We summarize the features of Genomic Data Model, from [6]. A sample s is a triple $\langle id, R, M \rangle$ where:

- id is the sample identifier, unique within each dataset.
- R is the set of genomic regions of the sample, built as pairs $\langle c, f \rangle$ of coordinates c and features f ; coordinates are arrays of four fixed attributes chr , $left$, $right$, $strand$; features are arrays of typed attributes; we assume attribute names of features to be different. The *region schema* of s is the list of attributes used for the identifier, the coordinates, and the features.
- M is the set of metadata of the sample, built as attribute-value pairs $\langle a, v \rangle$, where we assume the type of each value v to be *string*; the use of attribute-value pairs provides a generic solution that mediates among very different metadata formats of genomic sources.

A *dataset* is a collection of samples with the same region schema; Fig. 1 presents a dataset consisting of two samples.

Each GDM dataset is implemented by two data collections pairing identifiers respectively to metadata and regions. The upper part of Fig. 1 shows metadata, with the attributes: *ID*, *ATTRIBUTE*, *VALUE*. The intermediate part shows regions, with the attributes *ID*, *CHR*, *LEFT*, *RIGHT*, *STRAND* followed by an arbitrary number of attributes representing features; here, we illustrate a simple case with

*Current affiliation: German research center for artificial intelligence (DFKI)/ Technical University of Berlin (TU-Berlin), Berlin, Germany

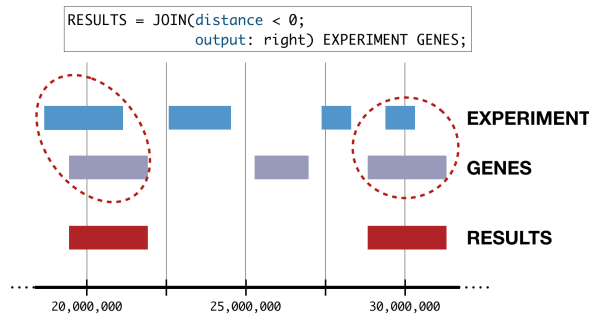


Figure 2: Genometric join in GMQL.

only one attribute, the region's *P. VALUE*. The lower part shows an abstract representation of the five regions, that belong to the first two chromosomes, aligned along the genome.

GMQL is a high-level language inspired by Codd's relational algebra; it extends conventional algebraic operations with domain-specific operations specifically designed for genomics [5]. GMQL operations include classic algebraic operations (SELECT, PROJECT, UNION, DIFFERENCE, JOIN, SORT, EXTEND, GROUP) and domain-specific operations (e.g., COVER deals with replicas of the same experiment; MAP refers regions of experiments to user-selected reference regions; GENOMETRIC JOIN selects region pairs based upon distance properties); the full description and language specification of GMQL is provided at the GMQL website¹.

A simple example of genometric join operation is shown in Fig. 2. Two samples, denoted as *EXPERIMENT* and *GENES*, are joined based on a simple distance predicate that is satisfied by overlapping regions (their distance must be less than zero). The *RESULT* is projected over the *RIGHT* operand; it consists of all the genes that intersect with the experiment regions.

A typical GMQL query starts with a SELECT operation, which applies to input datasets. The operation assigns to a GMQL variable those samples which are filtered by the selection predicate; these samples are loaded from a repository of the private and public dataset (later discussed). Then, the query continues with an arbitrary number of operations; eventually, it ends with a MATERIALIZE operation for the result variable. GMQL exploits a *batch and lazy execution model*: the actual execution starts with the MATERIALIZE operation. All operations producing the result variable are modeled as an *Operation DAG*, which is then logically and physically optimized.

In the join operation of Fig. 2, metadata of *RESULT* are copied from metadata of *GENES*; in general, GMQL queries extract in the result, operation after operation, the metadata content which is suitably composed from the metadata of the operands, thereby tracing the input samples which contributed to the results. Tracing metadata provenance during query processing is a unique aspect of our approach, that distinguishes GMQL from other genomic data management systems [1, 8, 9]; more comparisons can be found in [4, 5].

The GMQL system is organized as a four-layer architecture [4]: the *Access Layer*, supporting web interface / web services, a shell command line interface and several APIs (Scala, Python, R); the *Engine Layer*, including the compiler and managing the Operation DAG; the *Server Management Layer*, enabling the execution over heterogeneous implementations and environments (local vs. distributed); and the *Implementation Layer*, mapping the operation DAG to a specific implementation (Spark, Flink, SciDB).

Moreover, the Server Manager is supported by a Repository Manager, for accessing the data repository on heterogeneous file systems (local file system, Hadoop Distributed File System, and Google Cloud Storage).

The publicly accessible version of our system uses the Spark implementation and is deployed on a cluster provided by the CINECA supercomputing site². The cluster, made up of three core nodes and equipped, in total, with 120 vCPU and 375 GB of RAM, is driven by a master node providing Web services and a user-friendly Web interface.

2 DEMO HIGHLIGHTS

The demonstration is based on a specific query which is paradigmatic of the capabilities of our system; it is executed in two sections, from the GMQL Web-based user interface and from within a Python notebook, respectively; the former demo section highlights the declarative style of the language and the data browsing and visualization capabilities of the GMQL system, the latter demo session highlights the friendliness of a language embedding for data exploration and analysis.

2.1 Query Description

Informally, the query consists of loading three experimental data samples (*UPLOADED* dataset), then extracting the regions which are present in at least two out of the three experiments (using the *Cover* operation), then finding those resulting regions which overlap with genes (using the *Genometric Join* operation), and then finding those gene-overlapping regions which include at least one mutation (using the *Map* operation); genes are extracted from public annotations (*RefSeq* by UCSC), mutations are extracted from a public dataset from the *International Cancer Genome Consortium* (ICGC). The query can be spotted in panel (B) of Fig. 3; its step-by-step explanation is covered by videos available on the GeCo Website³.

2.2 Demo Objectives

- Demonstrate a complete user session, from input data upload to result in visualization.
- Explain three core operations of GMQL: Cover, Map, and Join. Explain their effect on genomic regions during GMQL execution.
- Show metadata browsing and in particular show region schemas, data profiles and metadata-based selections.
- Show query compilation/execution and data materialization within private workspaces.

¹<http://www.bioinformatics.deib.polimi.it/GMQL/>

²<https://www.cineca.it/>

³See: <http://www.bioinformatics.deib.polimi.it/geco/?video>

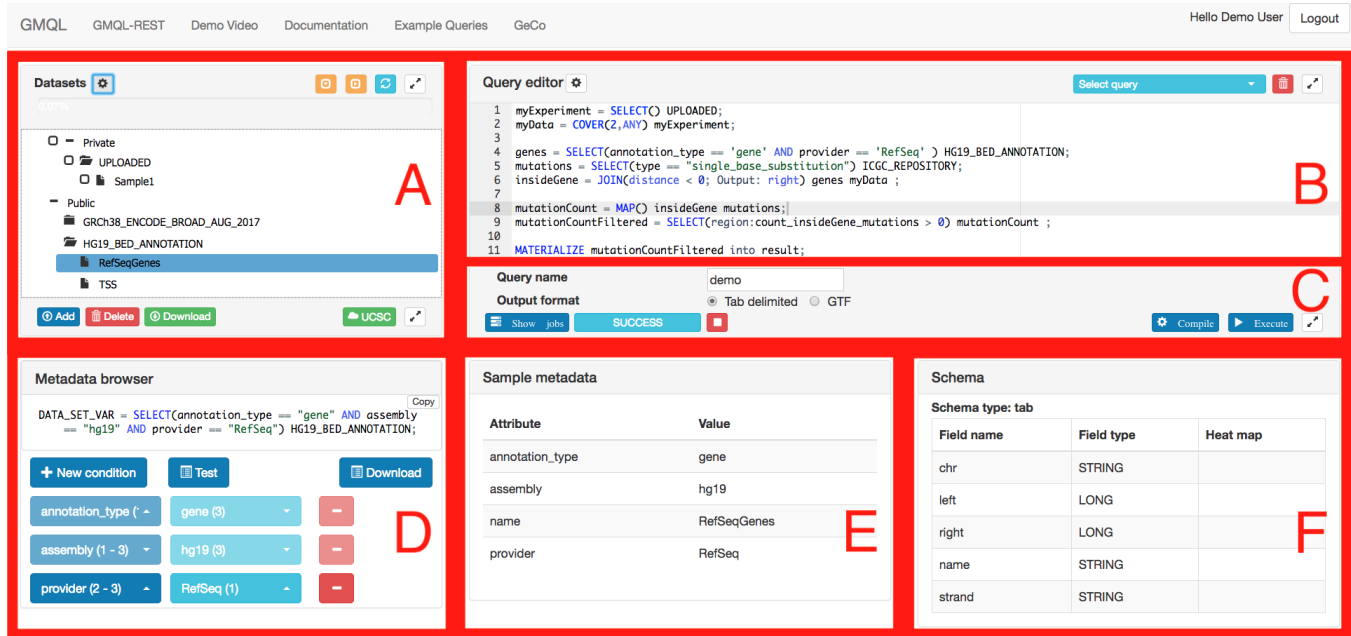


Figure 3: GMQL Web Interface, artificially divided into six panels representing (A) Public and private data resources, (B) Query editor, (C) Compilation/Execution monitor, (D) Metadata browser, (E) Metadata display, (F) Schema display.

- Show the integration with the UCSC Genome Browser for result displaying.
- Repeat the user session, including data loading, extraction and analysis, using the PyGMQL library from inside a Python notebook.
- Show exploratory steps of data analysis from within Python (including integration with Pandas library⁴).

2.3 Session using the Web interface

The GMQL Web interface can be invoked from within the GeCo Website⁵; guest logins are supported in addition to registered users. The demo starts with the loading of experimental datasets, activated by an upload button (Add); as result, an *UPLOADED* dataset appears in the private user's repository (Fig. 3A). The public repository includes public datasets from sources such as ENCODE (the Encyclopedia of DNA Elements [2]) and TCGA (The Cancer Genome Atlas [10]); the GMQL public repository currently consists of about 20 datasets and 150,000 samples.

By selecting a sample of the repository, users can view its metadata (Fig. 3E) and schema (Fig. 3F); the specific situation illustrated in Fig. 3 shows the *RefSeq* genes (a specific sample of the *HG19_BED_ANNOTATION* dataset). Users can then use the Metadata browser to progressively create a GMQL *SELECT* statement on a repository dataset, by adding simple predicates (Fig. 3D); metadata are suitably profiled, hence the user sees the cardinality of result samples after applying conjunctive conditions. When the condition is executed, it generates a tabular view of the metadata of

The screenshot shows a window titled 'Search result for public.HG19_BED_ANNOTATION'. It contains a table with columns: 'Samples', 'annotation_type', 'assembly', 'name', and 'provider'. The table shows one row for 'RefSeqGenes' with values 'gene', 'hg19', 'RefSeqGenes', and 'RefSeq'. The window also includes search filters, a 'Show' button, and a 'Close' button.

Figure 4: Metadata browser results.

the samples which satisfy the selection; Fig. 4 shows the metadata attributes and their values of the *RefSeqGenes* sample.

The user then can write the demo query shown in Fig. 3B. It is possible to separately compile each GMQL operation (Fig. 3C) and check statements one by one. GMQL uses a lazy query execution initiated by the *MATERIALIZ*E operation. During execution, some information about the processing state is shared with the user through a query log panel.

Upon termination, a new dataset is loaded in the user's private workspace (Fig. 3A). Results are profiled so as to provide their statistical description; Fig. 5 shows the profile of the query result.

2.4 Session using the Python library

GMQL can be used also through PyGMQL, a Python interface which supports GMQL queries in an interactive and exploration-driven fashion. Fig. 6 shows an example of join operation expressed in Python style, for comparison with the algebraic operation shown in

⁴<https://pandas.pydata.org/>

⁵See: <http://www.bioinformatics.deib.polimi.it/geco/?try>

Info of demo_20180226_085146_result	
Attribute	Value
Average region length	1002.23
Created by	demo
Creation date	2018/02/26 08:52:41
Number of regions	155
Number of samples	12

Figure 5: Profile of query result.

```
myExpOverGenes = refSeqGenes.join(experiment=myConfirmExp, refName="gene",
                                   genomeric_predicate=[gl.DLE(0)],
                                   output="RIGHT_DISTINCT")
```

Figure 6: Join in PyGMQL.

result.regs.head()							
	chr	start	stop	strand	name	score	count_gene_mutation
S_00000.gdm	chr13	37583450	37633850	-	NM_017569	0.0	1.0
S_00000.gdm	chr1	231762560	231886365	+	NM_001164555	0.0	1.0
S_00000.gdm	chr18	3496029	3845358	-	NM_001242766	0.0	4.0
S_00000.gdm	chr5	125877532	125931082	-	NM_001201377	0.0	1.0
S_00000.gdm	chr14	80962820	81405884	-	NM_152446	0.0	11.0

result.meta[["GENE.HM_TF.type", "GENE.DNase.cell", "MUTATION.disease", "MUTATION.type"]].head(3)			
id_sample	GENE.HM_TF.type	GENE.DNase.cell	MUTATION.disease
S_00000.gdm	[broadPeak]	[ECC-1]	[Lung cancer]
S_00001.gdm	[broadPeak]	[ECC-1]	[Melanoma]
S_00002.gdm	[broadPeak]	[ECC-1]	[Thyroid cancer]

Figure 7: Dataframe for data exchange.

Fig. 2; one can note the functional style which is typical of Python, but the use of operation parameters in the two cases is identical.

PyGMQL can execute both locally, using a local GMQL backend embedded in the library, or remotely, by sending queries (in the form of a graph data structure) to a remote server where a GMQL engine is installed. PyGMQL can be installed by using the *pip* package manager.

The result of a query is automatically loaded in memory as a Python structure called GDataframe, which in turn is composed of two Pandas dataframes, illustrated in Fig. 7; the dataframes reflect the GDM organization shown in Fig. 1, except for metadata which is organized in a tabular format. In Fig. 7, regions include mutation counts, metadata include some attributes from GENE and MUTATION datasets.

Thanks to the Pandas implementation, the GDataframe can be used as a starting point for complex data manipulation using specific Python libraries for data analysis (like Pandas or Numpy), for machine learning (like Scikit-Learn or Scipy) and for deep learning

```
plt.figure(figsize=(25, 20))
result.regs[result.regs.count_gene_mutation <= 50] \
    .hist("count_gene_mutation", bins=50)
plt.show()
```

<matplotlib.figure.Figure at 0x23808243400>

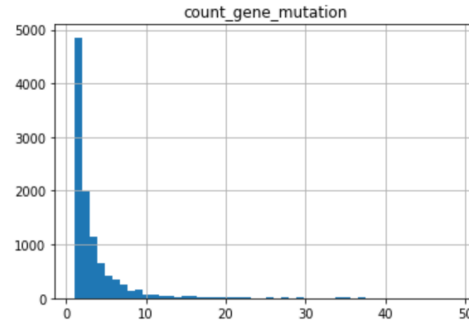


Figure 8: Data analysis using Pandas.

(like TensorFlow or Keras). Fig. 8 shows an example of simple descriptive statistics (the histogram of the genes with a given number of found mutations) computed upon the region dataframe of the result.

As illustrated in Fig. 8, PyGMQL code will be shown with the support of Jupyter notebook⁶, a programming environment for data exploration. The usage of Jupyter notebooks improves the reproducibility of the results, their sharing, and visualization.

The demo will be initiated by providing basic information about used concepts so that it will be understandable to a public with no background in genomics. We also developed a complex Jupiter notebook relative to an ongoing research project, which uses more sophisticated datasets (including: topologically associating domains, ChiaPet links, transcription factors and histone modifications), dedicated to interested users with a strong biological background.

ACKNOWLEDGMENT

This work is supported by the ERC Advanced Grant *GeCo (Data-Driven Genomic Computing)*.

REFERENCES

- [1] V. Bafna et al. Abstractions for genomics. *CACM*, 56(1):83-93 (2013).
- [2] ENCODE Project Consortium. An integrated encyclopedia of DNA elements in the human genome. *Nature*, 489(7414):57-74 (2012).
- [3] W.J. Kent et al. The Human Genome Browser at UCSC. *Genome Res*, 12(6):996-1006 (2002).
- [4] A. Kaitoua et al. Framework for supporting genomic operations *IEEE-TC*, 66(3):443-457 (2017).
- [5] M. Masseroli et al. GenoMetric Query Language: A novel approach to large-scale genomic data management. *Bioinformatics*, 31(12):1881-1888 (2015).
- [6] M. Masseroli, et al. Modeling and interoperability of heterogeneous genomic big data for integrative processing and querying. *Methods*, 1:111:3-11 (2016).
- [7] L.D. Stein. The case for cloud computing in genome informatics. *Genome Biol*, 11(5):207 (2010).
- [8] S. Tata et al. Declarative querying for biological sequences. In *Proc. IEEE ICDE* 87:99 (2006).
- [9] X.Zhu, et al. START: a system for flexible analysis of hundreds of genomic signal tracks in few lines of SQL-like queries. *BMC Genomics*, 18(1):749 (2017).
- [10] J.N. Weinstein et al. The Cancer Genome Atlas Pan-Cancer analysis project. *Nat Genet.*, 45(10):1113-1120 (2013).

⁶<http://jupyter.org/>