TASK 8: FINAL PROJECT

Diseño y Gestión de Sistemas de Información Genómicos (Spring 2024)

Julie Lervaag Breivik

Table of Contents

Technical report about SnpEff

SnpEff (“SNP Effects”) is a genetic variant annotation tool that efficiently categorizes the effects of variants on genes (such as amino acid changes) and uses a variety of genomic data sources and prediction algorithms. SnpEff annotate and predict the effects of genetic variations, particularly single nucleotide polymorphisms (SNPs), insertions, deletions, and other genomic variants. It analyzes genomic data to predict the impact of these variations on genes, proteins, and regulatory elements [1].

Whenever you have a huge file that describes the differences between your sample and the reference genome, but you think it is good to know more about these variants than just their genetic coordinates. For instance if they are in a spesific gene? In an exon? If they change protein coding? Do they cause premature stop codons? Then SnpEff is the perfect tool to use, the process of adding this information about the variants is called "Annotation".

SnpEff provides several degrees of annotations, from simple (e.g. which gene is each variant affecting) to extremely complex annotations (e.g. will this non-coding variant affect the expression of a gene?). It should be noted that the more complex the annotations, the more it relies in computational predictions. Because such computational predictions can potentially be incorrect, this can lead to the results from SnpEff (or any prediction algorithm) can therefore not be trusted blindly. It is therefore important that they are analyzed and independently validated.

My project makes use of SnpEff which plays a major role in annotating the genomic data. This will be geared towards determining the functional impacts of genomic variants found in the VCF (Variant Call Format) files. The inclusion of SnpEff into the pipeline by means of a Python script I have customized allows me not only to download and prepare the genomic data but also to fully annotate it. This in turn adds to its increasing level of scientific and clinical importance. The automated pipeline in place significantly reduces the effort required in handling genomic data, this simplifies the process and boosts the efficiency of the genomic data management.

To run SnpEff, the environment was set up with the following:

Java Runtime Environment (JRE): SnpEff runs on Java, requiring at least Java 8.

SnpEff.jar: The main executable JAR file for SnpEff.

The implementation involves setting up a Python script to invoke SnpEff using the subprocess module. This approach provides a simple way to integrate genomic annotation within my data pipeline. Upon a successful execution, the script outputs an VCF file that is now annotated. But what does that mean? It includes more Information.

When a VCF (Variant Call Format) file is annotated successfully using SnpEff, it means that additional information has been added to the file to describe the effects and implications of the genetic variants it lists. This annotation enhances the original VCF file by providing insights into how each variant might impact the genome, the genes, and potentially the phenotype of an organism.

What Does Annotation Add?

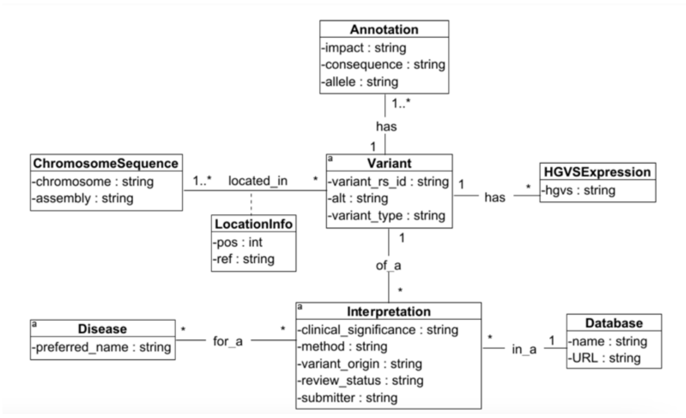
Annotation enriches a VCF file with the following types of information:

1. Gene Names and Locations: This shows the position of the disruptive genes and the actual changes on their specific locations due to the variants.
2. Type of Variation: This also consists of information whether the variant is a SNP, INS, DELETION , or one that has other changes of genetic structure.
3. Effect on the Gene: Thus, this variant creates an effect on a gene function by either synonymous (no change in amino acid), non-synonomous (change of amino acid), or it creates a premature Stop Codon (nonsense) or a frameshift together with the reading frame.
4. Impact Prediction: This signifies ability of the variant to disrupt protein function overwhelmingly high (completely change protein function), somewhat high (change protein effect at large extent), low (may change protein function slightly), or moderate (modifies protein function in some manner).
5. Conservation Scores: It supplies scores - the more they are conserved in different species, the higher the level of evolutionary conservation is. The most conserved domains are more likely just as functionally imperative.
6. Protein Changes: Glcout which are the particular alterations of amino acid sequences of proteins if any are present.
7. Allele Frequency: Shows pig population-specific allele frequent distributions, this gives information how popular the variant is in the different pig populations.
8. Regulatory Annotations: Determines whether or not the variants occurring in the regulator regions (like promoters or enhancers) would change the gene expression.

Technical report about the mapping

This report depicts the VCF file (Variant Call Format) mapping to a relational database model which uses the CSV files as a bridge. This step involved parsing the annotated VCF file, extracting relevant data, and structuring it into several CSV files that align with our database schema. The output is written in several CSV files that correspond to my database tables.

My goal here is to get the VCF data in the form that is acceptable to store in a relational database and allows performing of sophisticated data querying and management. Every data from VCF is managed to be stored in tables of the database. According to the figure 1 below you can see the Relational Data model. The CSV files was made according to that! Also for relation that are 1 to many or many to many needed to be stored in a separate csv file.



Picture 1. Relational Data model

In Python, I used a standard library packages called `csv` for file operations and `itertools.count` for generating unique identifiers. CSV files corresponding to each table in the database are opened for writing. Headers are written to each file based on the database schema. The script reads through the VCF file, and ignores or skip all the header lines (those starting with `#`) in all files. This transform.py script performs the functions of transformation of a annotated VCF file into multiple CSV files, following the exemplary formulation of schema. It does file writers initialization and unique ID generator alongside the main program. Then it starts to process each VCF record to accumulate and supply data concerning chromosomes, variants, and annotations to the respective CSV files. Furthermore, where required, it manages the avoidance of duplicate entries. The core tasks entail: looking for genomic elements inserted to the VCF, data saving the relational schema in CSV files, and normalizing information in all of lines with redundant data accordingly. At the last stage, it also runs all the processes to finish the file and, subsequently, assures the consistency of data.

Technical report about the process of how to Store CSV Files

This report is how I went from the Transformed, annotated vcf file to store them in a Database. The whole process is illustrated in Figure 2.

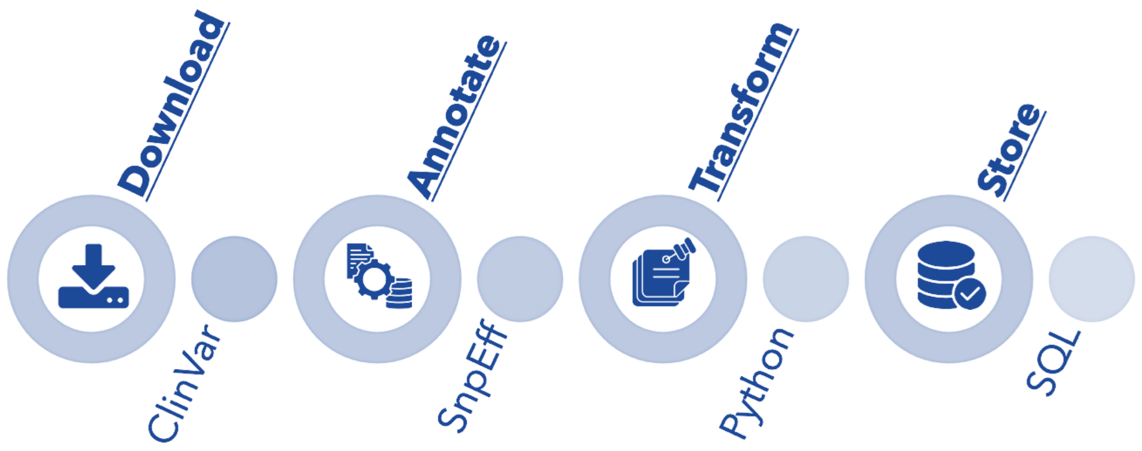


Figure 2. Project Flow

This process of storing structured CSV files into a MySQL database, by using Python to automate the data loading and table creation within a cloud-hosted environment. I first created a «MySQL on Clever Cloud» then I got the creadentials and everything I needed to connect from VScode to my database.

The process involves several steps implemented:

1.Environment Setup: Firstly the setup. The script begins by importing necessary libraries (`mysql.connector`, `pandas`) and setting up environment variables for database credentials to establish a connection to the MySQL server hosted on Clever Cloud.

2. Database Connection: Using `mysql.connector`, this establishes the connection to my MySQL database , by using my credentials. This connection facilitates all the operations on the database.

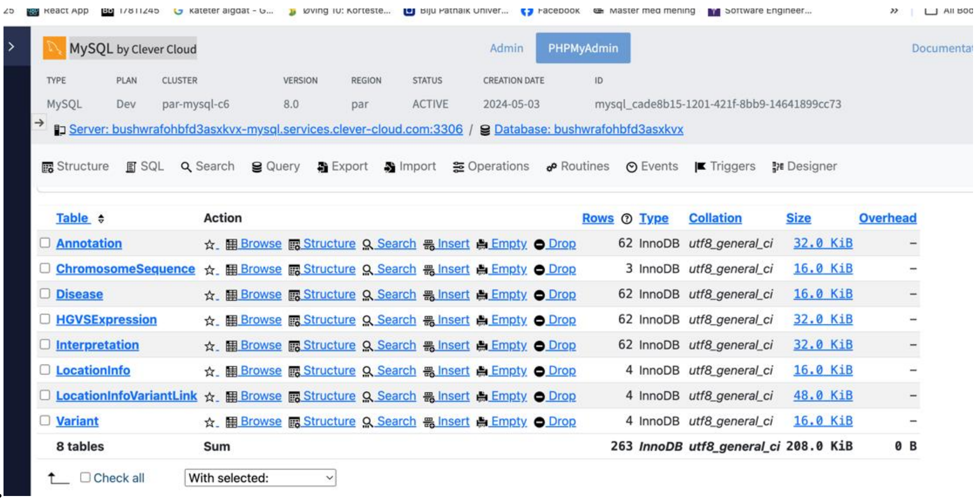
3. Table Creation: The script defines SQL commands for creating tables within the database. Each table is designed to hold specific data types related to genomic information, for instance variants, annotations, and links between diseases and genomic locations. SQL commands are executed to create tables such as `Variant`, `Annotation`, `Disease`, etc., with the appropriate fields and data types, this also includes relationships defined through foreign keys.

4. Data Loading: The script utilizes the `pandas` library to read data from CSV files located at specified paths. For each table, a cCSV file is read into a DataFrame. Data from each DataFrame is then inserted into the respective table in the database using a prepared SQL `INSERT` statement that iterates over each row of the DataFrame.

5. Transaction Management: All data insertions are wrapped within a transaction scope, this is done to ensure that any failure during the data loading process will not corrupt the database state. Commit operations is after that executed to finalize the insertions once each batch of data is successfully loaded.

6. Cleanup and Closure: After all data is loaded, the script closes the database connection and releases all resources, this ensures a clean and secure termination of the session.

Altogether this Python script is a method for transferring the genomic data from structured CSV files into the MySQL database. Figure 3. Shows my Database after inserting the testannotated.vcf file. And figure 4 Below shows my Environment variables.

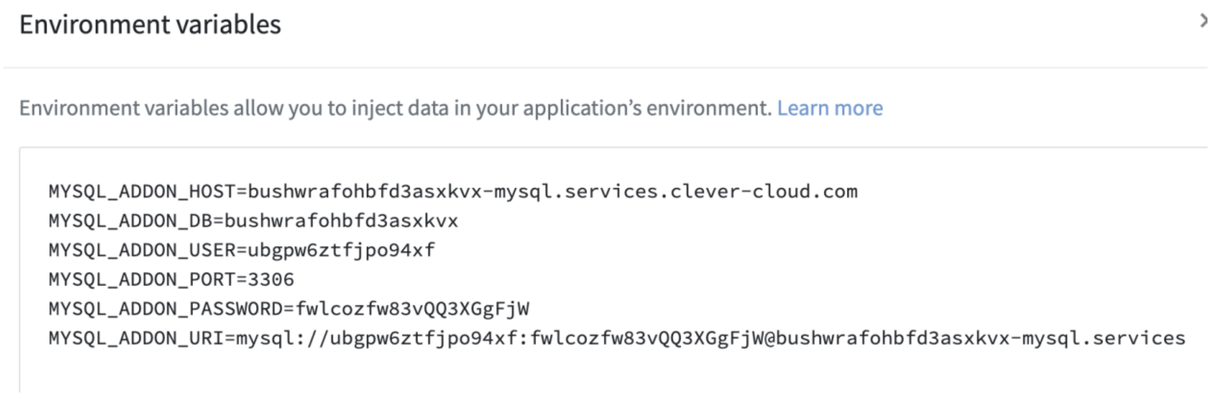


TYPE	PLAN	CLUSTER	VERSION	REGION	STATUS	CREATION DATE	ID
MySQL	Dev	par-mysql-c6	8.0	par	ACTIVE	2024-05-03	mysql_cade8b15-1201-421f-8bb9-14641899cc73

Server: bushwrafohbfd3asxkvx-mysql.services.clever-cloud.com:3306 / Database: bushwrafohbfd3asxkvx

Table	Action	Rows	Type	Collation	Size	Overhead
Annotation	Browse Structure Search Insert Empty Drop	62	InnoDB	utf8_general_ci	32.0 KiB	–
ChromosomeSequence	Browse Structure Search Insert Empty Drop	3	InnoDB	utf8_general_ci	16.0 KiB	–
Disease	Browse Structure Search Insert Empty Drop	62	InnoDB	utf8_general_ci	16.0 KiB	–
HGVSExpression	Browse Structure Search Insert Empty Drop	62	InnoDB	utf8_general_ci	32.0 KiB	–
Interpretation	Browse Structure Search Insert Empty Drop	62	InnoDB	utf8_general_ci	32.0 KiB	–
LocationInfo	Browse Structure Search Insert Empty Drop	4	InnoDB	utf8_general_ci	16.0 KiB	–
LocationInfoVariantLink	Browse Structure Search Insert Empty Drop	4	InnoDB	utf8_general_ci	48.0 KiB	–
Variant	Browse Structure Search Insert Empty Drop	4	InnoDB	utf8_general_ci	16.0 KiB	–
8 tables	Sum	263	InnoDB	utf8_general_ci	208.0 KiB	0 B

Figure. 3 MySQL Database


Environment variables

Environment variables allow you to inject data in your application's environment. Learn more

```
MYSQL_ADDON_HOST=bushwrafohbfd3asxkvx-mysql.services.clever-cloud.com
MYSQL_ADDON_DB=bushwrafohbfd3asxkvx
MYSQL_ADDON_USER=ubgpw6ztfjpo94xf
MYSQL_ADDON_PORT=3306
MYSQL_ADDON_PASSWORD=fwlcozfw83vQQ3XGgFjW
MYSQL_ADDON_URI=mysql://ubgpw6ztfjpo94xf:fwlcozfw83vQQ3XGgFjW@bushwrafohbfd3asxkvx-mysql.services
```

Figure 4. Environment variables

References

[1] https://pcingola.github.io/SnpEff/snpeff/introduction/