Introduction to Profile Areas Data Science Project Week 4

by Eva Aßmann, Paul Vogler

1 Scientific Background

Whole-genome sequencing provides the possibility to characterise the human genome more fully and accurately and enables clinicians to improve the diagnosis process and personalised patient treatment. In Next Generation Sequencing (NGS), fragment DNA is amplified and sequenced in parallel. The sequenced DNA reads are aligned against a reference genome to then identify differences or 'variants' between sample and reference sequence. Variant calling is clinically used for identifying genetic susceptibilities of a patient to specific disorders and can help guiding therapeutics or speeding up diagnosis using diagnostic gene panels.

2 Goal

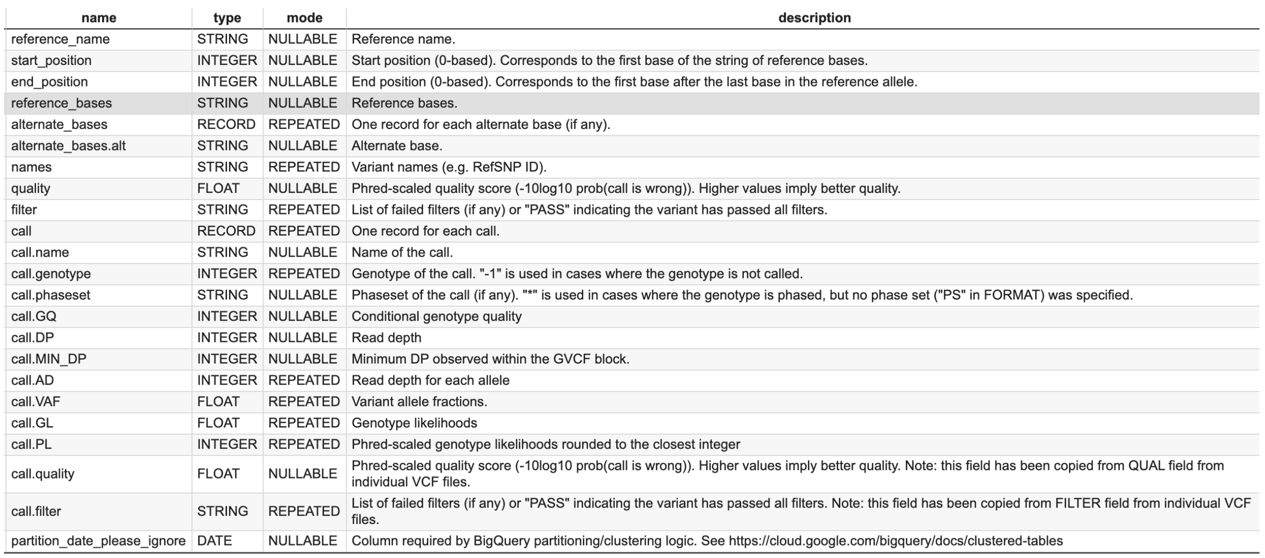
DNA sequencing data computationally requires large amounts of memory and runtime. This issue especially concerns clinical analysis tools that are based on access to large sequencing data, where time is crucial for diagnosis and further treatment strategies. Writing targeted, efficient queries and estimating the complexity of the required memory space thus are important abilities when working with SQL databases. The Google Data Analytics Product BigQuery is a server-less cloud data warehouse with built in in-memory BI engine and machine learning that allows real-time data analysis using SQL without the need for a database administrator. BigQuery also provides access to multiple publicly available data sets from the Google Healthcare and Life Sciences Products.

3 Data

In the Illumina Platinum Genomes Project, the 17 member CEPH pedigree 1463 was sequenced using Illumina HiSeq® systems and aligned to a human reference genome. Cloud Life Sciences retrieved the output binary alignment map file (BAM) for the 6 member CEPH pedigree 1463 from the European Nucleotide Archive (ENA) accession PRJEB3381, and variants were called using DeepVariant v0.7.0.
The data resulting from variant calling was stored in Genomic Variant Call Format (gVCF) files, transformed and loaded into BigQuery. gVCF files store information on all called variant segments and reference segments, i.e., DNA sequencing segments that matched the reference genome. After the transformation step, the variant calling data was stored under the BigQuery variants schema (Fig.1). The schema provides information on each variant and call, i.e., an identified occurrence of a variant or non-variant segment for an individual sample. The top-level records of a variants table can be both variants and non-variant segments. Each variants table record will contain one or more Calls. The data set comprised 105.923.159 entries and 19.6 GB of data. Non-variant segments are represented by a zero-length alternate_bases value and the text string <NON_REF>or <*>as an alternate_bases.alt value.

4 Data Analysis

Notes:
One call corresponds to one sample (one human). Each call comprises variant call information on all chromosomes (22 + X,Y). A call comprises non-variant segments (reference segments?) and variant segments. A called variant comprises more than one genotype: no-call variants only have -1 or 0 (reference) gt, true variants have only gt greater than zero. Variants can be non-variant segment, variant segment, reference segment, true variant, no-variant, high-quality variant, high-quality true variant.



name	type	mode	description
reference_name	STRING	NULLABLE	Reference name.
start_position	INTEGER	NULLABLE	Start position (0-based). Corresponds to the first base of the string of reference bases.
end_position	INTEGER	NULLABLE	End position (0-based). Corresponds to the first base after the last base in the reference allele.
reference_bases	STRING	NULLABLE	Reference bases.
alternate_bases	RECORD	REPEATED	One record for each alternate base (if any).
alternate_bases.alt	STRING	NULLABLE	Alternate base.
names	STRING	REPEATED	Variant names (e.g. RefSNP ID).
quality	FLOAT	NULLABLE	Phred-scaled quality score (-10log10 prob(call is wrong)). Higher values imply better quality.
filter	STRING	REPEATED	List of failed filters (if any) or "PASS" indicating the variant has passed all filters.
call	RECORD	REPEATED	One record for each call.
call.name	STRING	NULLABLE	Name of the call.
call.genotype	INTEGER	REPEATED	Genotype of the call. "-1" is used in cases where the genotype is not called.
call.phaseset	STRING	NULLABLE	Phaseset of the call (if any). "*" is used in cases where the genotype is phased, but no phase set ("PS" in FORMAT) was specified.
call.GQ	INTEGER	NULLABLE	Conditional genotype quality
call.DP	INTEGER	NULLABLE	Read depth
call.MIN_DP	INTEGER	NULLABLE	Minimum DP observed within the GVCF block.
call.AD	INTEGER	REPEATED	Read depth for each allele
call.VAF	FLOAT	REPEATED	Variant allele fractions.
call.GL	FLOAT	REPEATED	Genotype likelihoods
call.PL	INTEGER	REPEATED	Phred-scaled genotype likelihoods rounded to the closest integer
call.quality	FLOAT	NULLABLE	Phred-scaled quality score (-10log10 prob(call is wrong)). Higher values imply better quality. Note: this field has been copied from QUAL field from individual VCF files.
call.filter	STRING	REPEATED	List of failed filters (if any) or "PASS" indicating the variant has passed all filters. Note: this field has been copied from FILTER field from individual VCF files.
partition_date_please_ignore	DATE	NULLABLE	Column required by BigQuery partitioning/clustering logic. See https://cloud.google.com/bigquery/docs/clustered-tables

Figure 1: Reference name corresponds to chromosome id.

4.1 Counting total rows in the table

The query counts the total number of rows as the variable number_of_rows from the platinum genome deep-variants table and returns only this number.
Query size: 0.0 GB
IT view: Returns the total number of rows in the data table.
Biological view: Counts the number of reference regions in the dataset.

4.2 Counting variant calls in the table

4.2.1 Summing the lengths of call arrays

This query counts the number of variant calls (stored in an array in the data table), by summing up the individual call-array lengths as number_of_calls from the data table. The query returns only this total count. **Query size:** 0.0 GB **IT view:** Returns the summed length of all arrays in the call column. **Biological view:** Returns the number of calls that the variant caller made, including reference and non-variant calls.

4.2.2 Joining each row

This query also counts the number of variant calls, but uses a join of the call array to the table to unnest the call array and returns the row count of the joined table as number_of_calls. **Query size:** 0.0 GB **IT view:** Returns the number of entries in all arrays in the call column. **Biological view:** Returns the number of calls that the variant caller made, including reference and non-variant calls.

4.2.3 Counting name in a call column

This query, similarly to the previous one, counts the number of variant calls by joining the call array, but then counts the call names and returns the number as number_of_calls. **Query size:** 1.5264 GB **IT view:** Returns the total number of names in the call column. **Biological view:** Returns the number of calls that the variant caller made, including reference and non-variant calls.

4.3 Counting variant and non-variant segments

4.3.1 Counting variant segments

The query counts all variant calls like the queries before, but only selects the ones, where the unnested alt-bases array does not contain "¡NON_REF¿" or "¡*¿". The row count of this filtered table is returned as number_of_real_variants. **Query size:** 0.534 GB **IT view:** Returns the number of rows in the table, dropping the ones where the filter on alternate_bases does hold. **Biological view:** Returns the number of true variant calls from all reference positions.

4.3.2 Counting non-variant segments

The query is the same as the one above to count all variant calls, but reverses the row filter, to only include the rows with "¡NON_REF¿" or "¡*¿". The row count of this filtered table is returned as number_of_non_variants. **Query size:** 0.534 GB **IT view:** Returns the number of rows in the table, dropping the ones where the filter on alternate_bases does not hold. **Biological view:** Returns the number of non-variant calls from all reference positions.

4.4 Counting the variants called by each sample

4.4.1 With non-variants and variants

The query lists all the call names, as well as the count for each call name, and then groups and orders the table around the call names. The unique call names and each corresponding count is returned. **Query size:** 1.526 GB **IT view:** Returns the unique id and counts of the variable call.name. **Biological view:** Returns the number of calls for each sample, containing both non-variant and variant calls.

4.4.2 only with true variants

This query is similar to the previous one, but only includes rows, that do not contain "¡NON_REF¿" or "¡*¿" in the alternate_bases array. The returned values are the unique call names with their reduced counts. **Query size:** 2.06 GB **IT view:** Returns the unique id and counts of the variable call.name, but only using rows, where the filter on alternate_bases holds. **Biological view:** Returns the number of calls for each sample, only containing the true variant calls.

4.4.3 Filtering true variants by genotype

The query only includes the calls where at least one of the values in the genotype array is larger than 0, and no value is smaller than 0. Then the call names are counted and grouped again, to return the unique call names with their respective counts. **Query size:** 4.24 GB **IT view:** Returns the unique id and counts of the variable call.name, but only using rows, where the first filter on call.genotype holds and the second filter on call.genotype does not hold. **Biological view:** Returns the number of calls for each sample, where the genotype of the call contains at least one true variant and no no-calls.

4.5 Counting samples in the table

The query counts the number of distinct names in the call column arrays and returns that number. **Query size:** 1.526 GB **IT view:** Returns the count of the unique values of the call.name array entries **Biological view:** Returns the number of samples.

4.6 Counting variants per chromosome

The query counts all rows in which there is at least one variant call with at least one genotype greater than 0, then counts each group, groups the variant rows by chromosome and uses a case to order the chromosome strings numerically. **Query size:** 3.341 GB **IT view:** Returns the unique values of reference_name and the count of reference_name for all rows where the call.genotype filter holds **Biological view:** Returns the name of the chromosome and the number of variant rows for that chromosome.

4.7 Counting high-quality variants per sample

4.7.1 Querying calls with multiple FILTER values

The query extracts all filters from the call column and returns the filter name as well as their count. **Query size:** 0.248 GB **IT view:** Returns all unique values from the call.filter array as well as their counts. **Biological view:** Returns all filters that a variant call didn't pass or the word 'PASS' if all filters were passed.

4.7.2 FILTERing for high quality variant calls

The query only includes rows, where the call filter value is 'PASS' and the filter array contains more than one value. Multiple top level columns as well as the call filters an their counts are returned. The returned dataframe was empty. **Query size:** 4.281 GB **IT view:** Returns all rows that contain multiple call.filter values if the call.filter array contains 'PASS'. **Biological view:** Returns the columns that passed all filters after variant calling but erroneously have non-passed filters in the list.

4.7.3 Counting all high quality calls for each sample

The query only includes rows, where the call filter value is 'PASS' and then all returns the call grouped names as well as their counts. **Query size:** 1.774 GB **IT view:** Returns all unique call.name values as well as their count, for all rows where the call.filter array contains 'PASS'. **Biological view:** Returns the number of high quality calls, that passed all filters, per sample.

4.7.4 Counting all high quality true variant calls for each sample

The query is the similar to the previous one, but additionally only includes rows, where the call genotype is larger than 0. **Query size:** 4.488 GB **IT view:** Returns all unique call.name values as well as their count, for all rows where the call.filter array contains 'PASS' and the call.genotype array contains a value larger than 0. **Biological view:** Returns the count of all calls (variants and non-variants) for each sample, omitting any call with a non-PASS filter, and only includes calls with at least one true variant.

4.8 Best practices

4.8.1 Condensing the 'Counting variants per chromosome' query

1. Unnest the call array an the genotype array to process them on each row step
2. Join the call column with the call.genotype column to shorten the nested EXIST clause
3. Remove the "chr" prefix from the reference_name column using a regex replace function and give it the alias chromosome, to sort the output stringlike
4. cast the chromosome column from a string to an integer to sort numerically (erroneous)
5. insert a CASE selection to remove the "chr" prefix and to add a leading 0 to all numerically compatible values smaller 10 and keep the column as a string indexer
6. display the reference_name again and move the CASE selection to the ORDER BY function

4.8.2 Writing user-defined functions

Using the 'Counting variants per chromosome' like in the previous section, the CASE function that was moved to the ORDER BY segment is now moved to a user-defined function that is called in the ORDER BY segment. The first user-defined function is a SQL native function that returns the regex-transformed reference_name indexer.
The second user-defined function work the same way as the first one, but is written in java script.

5 Discussion

This was a typical project for data scientists, because we handled and processed large amounts of data from an SQL database. We needed to handle limited memory space by using cloud and server based data connections and in addition to this distributed computing, needed to query the data as efficient as possible to stay within the limits of the cloud provider.