Advanced guide to analyzing variants using BigQuery

This page describes advanced methods for how to use BigQuery (https://cloud.google.com/bigquery) to analyze variants.

The data in this tutorial comes from the Illumina Platinum Genomes project (https://cloud.google.com/life-sciences/docs/public-datasets/illumina-platinum-genomes). The data was loaded into a BigQuery table that uses the BigQuery variants schema (https://cloud.google.com/life-sciences/docs/how-tos/bigquery-variants-schema). The name of the table is `platinum_genomes_deepvariant_variants_20180823`.

If your variant data is in a BigQuery table that uses the BigQuery variants schema, it's straightforward to apply the queries in this tutorial to your data. For information on how to load variant data into BigQuery, see the documentation on using the transform pipeline (https://cloud.google.com/life-sciences/docs/how-tos/load-variants#transform-pipeline).

Objectives

After completing this tutorial, you'll know how to:

- Get an overview of the data
- Find out how non-variant segments are represented
- Find out how variant calls are represented
- Find out how variant call quality filters are represented
- Aggregate hierarchical columns
- Condense queries
- Count distinct rows
- Group rows
- Write user-defined functions

This tutorial also shows how to find:

- The number of rows in the table
- The number of variant calls
- Variants called for each sample
- The number of samples
- Variants per chromosome
- High quality variants per sample

Costs

This tutorial uses billable components of Google Cloud, including:

- BigQuery

Use the Pricing Calculator (https://cloud.google.com/products/calculator#tab=bigquery) to generate a cost estimate based on your projected usage. New Cloud Platform users might be eligible for a free trial (https://cloud.google.com/free-trial).

Before you begin

1. Sign in (https://accounts.google.com/Login) to your Google Account.

   If you don't already have one, sign up for a new account (https://accounts.google.com/SignUp).

2. In the Cloud Console, on the project selector page, select or create a Cloud project.

   ★ **Note**: If you don't plan to keep the resources that you create in this procedure, create a project instead of selecting an existing project. After you finish these steps, you can delete the project, removing all resources associated with the project.

   GO TO THE PROJECT SELECTOR PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/PROJECTSELECT

3. Make sure that billing is enabled for your Google Cloud project. Learn how to confirm billing is enabled for your project (https://cloud.google.com/billing/docs/how-to/modify-project).

4. You should be familiar with the BigQuery variants schema (https://cloud.google.com/life-sciences/docs/how-tos/bigquery-variants-schema).

Viewing the table schema and data

Access the table and view the schema

The Illumina Platinum Genomes `platinum_genomes_deepvariant_variants_20180823` table is publicly available through this link (https://bigquery.cloud.google.com/table/bigquery-public-data:human_genome_variants.platinum_genomes_deepvariant_variants_20180823?tab=schema)
.

Variants and non-variants in the table

The Illumina Platinum Genomes data uses the gVCF (https://sites.google.com/site/gvcftools/home/about-gvcf) format, which means that there are rows in the table that include non-variants. These non-variants are also known as "reference calls."

In the table, the non-variant segments are generally represented in the following ways:

- With a zero-length `alternate_bases` value
- With the text string `<NON_REF>` as an `alternate_bases.alt` value
- With the text string `<*>` as an `alternate_bases.alt` value

The way that non-variant segments are represented typically depends on the variant caller that generated the source data. The variants in the `platinum_genomes_deepvariant_variants_20180823` table have been called using DeepVariant (https://cloud.google.com/life-sciences/deepvariant), which uses the `<*>` notation.

The following tables show some rows containing values that represent non-variant segments. The segments show a reference block of `10` bases on chromosome `1`. The reference block starts at position `1000`. The reference base at position `1000` is an `A`. The reference bases at the other positions of the block are not shown.

In the following table, the `alternate_bases` `REPEATED RECORD` column contains no values, meaning that it is an `ARRAY` of length 0.

reference_name	start_position	end_position	reference_bases	alternate_bases.alt
1	1000	1010	A	

In the following table, the `alternate_bases` `REPEATED RECORD` column is length 1, and it contains the literal text string `<*>`.

reference_name	start_position	end_position	reference_bases	alternate_bases.alt
1	1000	1010	A	<*>

The queries used in this guide use the three representations shown above.

See the VCF specification (https://samtools.github.io/hts-specs/VCFv4.3.pdf) for more information on representing non-variant positions in the genome.

Viewing the table data

To view the data in the `platinum_genomes_deepvariant_variants_20180823` table:

1. Go to the **Details** page in the BigQuery UI (https://bigquery.cloud.google.com/table/bigquery-public-data:human_genome_variants.platinum_genomes_deepvariant_variants_20180823?tab=details).

   Information about the table appears. You can see that it contains 19.6 GB of data and has over 105,000,000 rows.
2. Click **Preview** to view some of the rows in the table.

Querying the table

After viewing the table schema and some of its rows, you can now start issuing queries and analyzing data. Before continuing, make sure that you're familiar with the Standard SQL Query Syntax (https://cloud.google.com/bigquery/docs/reference/standard-sql/query-syntax) that BigQuery uses.

Counting total rows in the table

To view the number of rows in the table:

1. Go to the BigQuery UI.

   GO TO THE BIGQUERY UI (HTTPS://BIGQUERY.CLOUD.GOOGLE.COM)

2. Click **Compose query**.
3. Copy and paste the following query into the **New Query** text area:

```
#standardSQL
SELECT
  COUNT(1) AS number_of_rows
FROM
  `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_var
```

4. Click **Run query**. Running the query returns:

Row	number_of_rows
1	105923159

Counting variant calls in the table

Each row in the table has a genomic position that is either a variant or non-variant segment.

Each row also contains a `call` column, which is an `ARRAY` of variant calls. Each `call` column includes the `name` and other values, such as the genotype, quality columns, read depth, and others typically found in a VCF file.

To count the number of variant calls, query the number of elements inside the `ARRAY` columns. You can do this in several ways which are shown below. Each query returns the value 182,104,652, which means that there is an average of 1.7 variant calls per row in the dataset.

Summing the lengths of `call` arrays

One way to count the total number of variant calls across all samples is to sum the length of each `call` array:

```
#standardSQL
SELECT
```

```
  SUM(ARRAY_LENGTH(call)) AS number_of_calls
FROM
  `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_variants_
```

Running the query returns the correct value (182,104,652):

Row	number_of_calls
1	182104652

JOINing each row

A second way to count the total number of variant calls across all samples is to `JOIN` (https://cloud.google.com/bigquery/docs/reference/standard-sql/query-syntax#join_types) each row with the `call` column. Note the use of the comma (`,`) operator, which is a shorthand notation used for `JOIN`. Also note that the join to the `call` column makes an implicit `UNNEST` (https://cloud.google.com/bigquery/docs/reference/standard-sql/query-syntax#unnest) operation on the `call` column:

```
#standardSQL
SELECT
  COUNT(call) AS number_of_calls
FROM
  `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_variants_
```

Counting name in a call column

A third way to count the total number of variant calls across all samples is to count the `name` values in the `call` column. Each `call` column must have a single `name` value, so you can run the following query:

```
#standardSQL
SELECT
  COUNT(call.name) AS number_of_calls
FROM
  `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_variants_
```

Running the query returns the correct value (182,104,652):

Row	number_of_calls
1	182104652

Counting variant and non-variant segments

To count the number of variant and non-variant segments in the table, first run a query to filter out the non-variant segments:

```
#standardSQL
SELECT
  COUNT(1) AS number_of_real_variants
FROM
  `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_variants_
WHERE
  EXISTS (SELECT 1
            FROM UNNEST(v.alternate_bases) AS alt
          WHERE
            alt.alt NOT IN ("<NON_REF>", "<*>"))
```

Running the query returns:

Row	number_of_real_variants
1	38549388

As noted in Counting variant calls (#counting_variant_calls_in_the_table), the total number of variant calls in the table is 182,104,652, so this result shows that the vast majority of rows in the table are non-variant segments.

As shown in the section on Variants and non-variants in the table (#variants_and_non-variants_in_the_table_variants), there are at least three ways to classify a variant row as a non-variant segment. In the query above, the `WHERE` clause includes rows where the `alternate_bases` column has a value that is a true variant (meaning that it is not a special marker value such as `<*>` or `<NON_REF>`).

For each row in the table, a subquery is issued over the `alternate_bases` column of that row, which returns the value `1` for each value of `alternate_bases` that is not `<NON_REF>` or `<*>`. The number of rows that the subquery returns is the number of variant segments.

The following query shows how to get the count of non-variant segments:

```
#standardSQL
SELECT
  COUNT(1) AS number_of_non_variants
FROM
  `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_variants_
WHERE
  NOT EXISTS (SELECT 1
                FROM UNNEST(v.alternate_bases) AS alt
              WHERE
                alt.alt NOT IN ("<NON_REF>", "<*>"))
```

Running the query returns:

Row	number_of_non_variants
1	143555264

Adding the number of real variants (38,549,388) to the number of non-variant segments (143,555,264) equals the total number of variant calls.

Counting the variants called by each sample

After examining the top-level rows in the table, you can start querying for child rows. These rows include data such as the individual samples that have had calls made against the variants.

Each variant in the table has zero or more values for `call.name`. A particular `call.name` value can appear in multiple rows.

To count the number of rows in which each call set appears, run the following query:

```
#standardSQL
SELECT
  call.name AS call_name,
  COUNT(call.name) AS call_count_for_call_set
FROM
  `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_variants_
GROUP BY
  call_name
```

```
ORDER BY
  call_name
```

Running the query returns six rows. Each `call_name` corresponds to a sequenced individual human:

Row	call_name	call_count_for_call_set
1	NA12877	31592135
2	NA12878	28012646
3	NA12889	31028550
4	NA12890	30636087
5	NA12891	33487348
6	NA12892	27347886

Humans typically don't have the 30 million variants shown in the values for `call_count_for_call_set`. Filter out the non-variant segments to count just the variant rows:

```
#standardSQL
SELECT
  call.name AS call_name,
  COUNT(call.name) AS call_count_for_call_set
FROM
  `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_variants_
WHERE
  EXISTS (SELECT 1
            FROM UNNEST(v.alternate_bases) AS alt
          WHERE
            alt.alt NOT IN ("<NON_REF>", "<*>"))
GROUP BY
  call_name
ORDER BY
  call_name
```

Running the query returns:

Row	call_name	call_count_for_call_set

Row	call_name	call_count_for_call_set
1	NA12877	6284275
2	NA12878	6397315
3	NA12889	6407532
4	NA12890	6448600
5	NA12891	6516669
6	NA12892	6494997

The number of variants is now closer to 6 million, which is more typical for a human. Continue to the next section to filter true variants by genotype.

Filtering true variants by genotype

The variants in the table include no-calls, which are represented by a `genotype` value of -1. These variants are not considered true variants for individuals, so filter them out. True variants can only include calls with genotypes greater than zero. If a call includes only genotypes that are no-calls (-1) or reference (0), then they are not true variants.

To filter the variants by genotype:

```
#standardSQL
SELECT
  call.name AS call_name,
  COUNT(call.name) AS call_count_for_call_set
FROM
  `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_variants_
WHERE
  EXISTS (SELECT 1 FROM UNNEST(call.genotype) AS gt WHERE gt > 0)
  AND NOT EXISTS (SELECT 1 FROM UNNEST(call.genotype) AS gt WHERE gt < 0)
GROUP BY
  call_name
ORDER BY
  call_name
```

Running the query returns:

Row	call_name	call_count_for_call_set
1	NA12877	4486610
2	NA12878	4502017
3	NA12889	4422706
4	NA12890	4528725
5	NA12891	4424094
6	NA12892	4495753

Counting samples in the table

In Counting the variants called by each sample (#counting_the_variants_called_by_each_sample), each query returned six rows with values for **`call_name`**. To instead query for and get the value for that number of rows, you can run the following query:

```
#standardSQL
SELECT
  COUNT(DISTINCT call.name) AS number_of_callsets
FROM
  `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_variants_
```

Running the query returns:

Row	number_of_callsets
1	6

Counting variants per chromosome

To count the number of variants per chromosome, you can run the following query, which:

- Counts all rows in which there is at least one variant call with at least one genotype greater than 0
- Groups the variant rows by chromosome and counts each group

```
#standardSQL
SELECT
  reference_name,
  COUNT(reference_name) AS number_of_variant_rows
FROM
  `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_variants_
WHERE
  EXISTS (SELECT 1
            FROM UNNEST(v.call) AS call, UNNEST(call.genotype) AS gt
          WHERE gt > 0)
GROUP BY
  reference_name
ORDER BY
  CASE
    WHEN SAFE_CAST(REGEXP_REPLACE(reference_name, '^chr', '') AS INT64) < 10
      THEN CONCAT('0', REGEXP_REPLACE(reference_name, '^chr', ''))
      ELSE REGEXP_REPLACE(reference_name, '^chr', '')
  END
```

Running the query returns the name of the chromosome (`reference_name`) and the number of variant rows for each chromosome:

Row	reference_name	number_of_variant_rows
1	chr1	615000
2	chr2	646401
3	chr3	542315
4	chr4	578600
5	chr5	496202
...	...	...

Counting high-quality variants per sample

Querying calls with multiple `FILTER` values

The VCF specification (https://samtools.github.io/hts-specs/VCFv4.3.pdf) describes the `FILTER` column which can be used to label variant calls of differing qualities.

The following query shows how to view the per-variant-call `FILTER` values for the dataset:

```
#standardSQL
SELECT
  call_filter,
  COUNT(call_filter) AS number_of_calls
FROM
  `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_variants_
  v.call,
  UNNEST(call.FILTER) AS call_filter
GROUP BY
  call_filter
ORDER BY
  number_of_calls
```

Running the query returns:

Row	call_filter	number_of_calls
1	RefCall	11681534
2	PASS	26867854

The `PASS` value signifies that a variant call is of a high quality.

`FILTER`ing for high quality variant calls

When analyzing variants, you might want to filter out lower quality variants. If the `FILTER` column contains the value `PASS`, it is expected that the column will contain no other values. You can verify this by running the following query. The query also omits any calls that do not contain a `PASS` value under `FILTER`.

```
#standardSQL
SELECT
  reference_name,
  start_position,
  end_position,
  reference_bases,
  call.name AS call_name,
  (SELECT STRING_AGG(call_filter) FROM UNNEST(call.FILTER) AS call_filter) AS filter
  ARRAY_LENGTH(call.FILTER) AS filter_count
FROM
```

```
  `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_variants_
WHERE
  EXISTS (SELECT 1 FROM UNNEST(call.FILTER) AS call_filter WHERE call_filter = 'PASS
  AND ARRAY_LENGTH(call.FILTER) > 1
ORDER BY
  filter_count DESC, reference_name, start_position, end_position, reference_bases,
LIMIT
  10
```

As expected, running the query returns zero results.

Counting all high quality calls for each sample

The following query shows how to count all calls (variants and non-variants) for each call set, and omits any call with a non-`PASS` filter:

```
#standardSQL
SELECT
  call.name AS call_name,
  COUNT(1) AS number_of_calls
FROM
  `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_variants_
WHERE
  NOT EXISTS (SELECT 1 FROM UNNEST(call.FILTER) AS call_filter WHERE call_filter !=
GROUP BY
  call_name
ORDER BY
  call_name
```

Running the query returns:

Row	call_name	number_of_calls
1	NA12877	29795946
2	NA12878	26118774
3	NA12889	29044992
4	NA12890	28717437
5	NA12891	31395995

Row	call_name	number_of_calls
6	NA12892	25349974

Counting all high quality true variant calls for each sample

The following query shows how to count all calls (variants and non-variants) for each sample. It omits any call with a non-`PASS` filter, and only includes calls with at least one true variant, meaning that `genotype` > 0:

```
#standardSQL
SELECT
  call.name AS call_name,
  COUNT(1) AS number_of_calls
FROM
  `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_variants_
WHERE
  NOT EXISTS (SELECT 1 FROM UNNEST(call.FILTER) AS call_filter WHERE call_filter !=
  AND EXISTS (SELECT 1 FROM UNNEST(call.genotype) as gt WHERE gt > 0)
GROUP BY
  call_name
ORDER BY
  call_name
```

Running the query returns:

Row	call_name	number_of_calls
1	NA12877	4486610
2	NA12878	4502017
3	NA12889	4422706
4	NA12890	4528725
5	NA12891	4424094
6	NA12892	4495753

Best practices

Condensing queries

As your queries become more complex, it's important to keep them concise to ensure that their logic is correct and simple to follow.

The following example demonstrates how to start from a query that counts the number of variants per chromosome and, step by step, condense it using SQL syntax and user-defined functions.

As explained in the section on counting variants per chromosome (#counting_variants_per_chromosome), the query has the following requirements:

- Counts all rows in which there is at least one variant call with at least one genotype greater than 0
- Groups the variant rows by chromosome and counts each group

Writing this query can be complicated because, to complete the first task, you need to look into an `ARRAY` (`genotype`) within an `ARRAY` (`call`) while keeping the execution context of the query at the row level. This is because you want to produce a per-variant result, rather than a per-`call` or per-`genotype` result.

The `UNNEST` function allows you to query over an `ARRAY` column as if the column were a table. The function returns one row for each element of an `ARRAY`. It also doesn't change the query context. Therefore, you can start with using an `UNNEST` function in an `EXISTS` subquery in a `WHERE` clause:

```
#standardSQL
SELECT
  reference_name,
  COUNT(reference_name) AS number_of_variant_rows
FROM
  `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_variants_
WHERE
  EXISTS (SELECT 1
            FROM UNNEST(v.call) AS call
          WHERE EXISTS (SELECT 1
                          FROM UNNEST(call.genotype) AS gt
                        WHERE gt > 0))
GROUP BY
```

```
  reference_name
ORDER BY
  reference_name
```

The query returns the same results as the example in counting variants per chromosome (#counting_variants_per_chromosome):

Row	reference_name	number_of_variant_rows
1	chr1	615000
2	chr10	396773
3	chr11	391260
4	chr12	382841
5	chr13	298044
...	...	...

The query can be more concise by changing the `EXISTS` clause into a `JOIN` of the `call` column with the `call.genotype` column. Recall that the comma operator is a shorthand notation used for `JOIN`:

```
#standardSQL
SELECT
  reference_name,
  COUNT(reference_name) AS number_of_variant_rows
FROM
  `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_variants_
WHERE
  EXISTS (SELECT 1
            FROM UNNEST(v.call) AS call, UNNEST(call.genotype) AS gt
          WHERE gt > 0)
GROUP BY
  reference_name
ORDER BY
  reference_name
```

The query works, and is concise, but it doesn't allow you to sort the output in ascending numerical order of chromosomes (`reference_name`). This is because the values in `reference_name` are string types, and each value contains the prefix "chr."

To sort the output numerically, first remove the "chr" prefix from the `reference_name` column and give it the alias `chromosome`:

```
#standardSQL
SELECT
  REGEXP_REPLACE(reference_name, '^chr', '') AS chromosome,
  COUNT(reference_name) AS number_of_variant_rows
FROM
  `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_variants_
WHERE
  EXISTS (SELECT 1
            FROM UNNEST(v.call) AS call, UNNEST(call.genotype) AS gt
          WHERE gt > 0)
GROUP BY
  chromosome
ORDER BY
  chromosome
```

The query uses the [`REGEXP_REPLACE`](https://cloud.google.com/bigquery/docs/reference/standard-sql/functions-and-operators#regexp_replace)
 (https://cloud.google.com/bigquery/docs/reference/standard-sql/functions-and-operators#regexp_replace)
function to replace the "chr" prefix string with an empty string. It then changes the `GROUP BY` and `ORDER BY` functions to use the computed `chromosome` alias. However, the output still sorts by string:

Row	chromosome	number_of_variant_rows
1	1	615000
2	10	396773
3	11	391260
4	12	382841
5	13	298044
...	...	...

To instead sort the output numerically, cast the `chromosome` column from a string to an integer:

```
#standardSQL
SELECT
```

```
  CAST(REGEXP_REPLACE(reference_name, '^chr', '') AS INT64) AS chromosome,
  COUNT(reference_name) AS number_of_variant_rows
FROM
  `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_variants_
WHERE
  EXISTS (SELECT 1
            FROM UNNEST(v.call) AS call, UNNEST(call.genotype) AS gt
          WHERE gt > 0)
GROUP BY
  chromosome
ORDER BY
  chromosome
```

In this case, the query returns an error because not all chromosome names, such as "X," "Y," and "M" are numeric. Instead, use the **CASE** (https://cloud.google.com/bigquery/docs/reference/standard-sql/functions-and-operators#conditional-expressions) function to prepend a "0" to chromosomes 1 through 9 and remove the "chr" prefix:

```
#standardSQL
SELECT
  CASE
    WHEN SAFE_CAST(REGEXP_REPLACE(reference_name, '^chr', '') AS INT64) < 10
      THEN CONCAT('0', REGEXP_REPLACE(reference_name, '^chr', ''))
      ELSE REGEXP_REPLACE(reference_name, '^chr', '')
  END AS chromosome,
  COUNT(reference_name) AS number_of_variant_rows
FROM
  `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_variants_
WHERE
  EXISTS (SELECT 1
            FROM UNNEST(v.call) AS call, UNNEST(call.genotype) AS gt
          WHERE gt > 0)
GROUP BY
  chromosome
ORDER BY
  chromosome
```

The query returns the correct output:

Row	chromosome	number_of_variant_rows
1	01	615000

Row	chromosome	number_of_variant_rows
2	02	646401
3	03	542315
4	04	578600
5	05	496202
...	...	...

Note the use of the **SAFE_CAST** (https://cloud.google.com/bigquery/docs/reference/standard-sql/functions-and-operators#casting) function, which returns `NULL` for chromosomes X, Y, and M instead of returning an error.

As a last improvement on the output, display the `reference_name` column again instead of setting it to the `chromosome` alias. To do so, move the `CASE` clause to the `ORDER BY` function:

```
#standardSQL
SELECT
  reference_name,
  COUNT(reference_name) AS number_of_variant_rows
FROM
  `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_variants_
WHERE
  EXISTS (SELECT 1
            FROM UNNEST(v.call) AS call, UNNEST(call.genotype) AS gt
          WHERE gt > 0)
GROUP BY
  reference_name
ORDER BY
  CASE
    WHEN SAFE_CAST(REGEXP_REPLACE(reference_name, '^chr', '') AS INT64) < 10
      THEN CONCAT('0', REGEXP_REPLACE(reference_name, '^chr', ''))
      ELSE REGEXP_REPLACE(reference_name, '^chr', '')
  END
```

This final query is the same as the one shown in Counting variants per chromosome (#counting_variants_per_chromosome).

Writing user-defined functions

BigQuery supports User-defined functions (https://cloud.google.com/bigquery/docs/reference/standard-sql/user-defined-functions), which enable you to create a function using another SQL expression or another programming language, such as JavaScript.

The example in Condensing queries (#condensing_queries) shows how to build a complex query, but the query becomes overly verbose.

The following query demonstrates how to make the query more concise by moving the `CASE` logic into a function:

```
#standardSQL
CREATE TEMPORARY FUNCTION SortableChromosome(reference_name STRING)
  RETURNS STRING AS (
  -- Remove the leading "chr" (if any) in the reference_name
  -- If the chromosome is 1 - 9, prepend a "0" since
  -- "2" sorts after "10", but "02" sorts before "10".
  CASE
    WHEN SAFE_CAST(REGEXP_REPLACE(reference_name, '^chr', '') AS INT64) < 10
      THEN CONCAT('0', REGEXP_REPLACE(reference_name, '^chr', ''))
      ELSE REGEXP_REPLACE(reference_name, '^chr', '')
  END
);

SELECT
  reference_name,
  COUNT(reference_name) AS number_of_variant_rows
FROM
  `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_variants_
WHERE
  EXISTS (SELECT 1
            FROM UNNEST(v.call) AS call, UNNEST(call.genotype) AS gt
          WHERE gt > 0)
GROUP BY
  reference_name
ORDER BY SortableChromosome(reference_name)
```

The following query also demonstrates how to make the query more concise, but it uses a function defined in JavaScript:

```
#standardSQL
CREATE TEMPORARY FUNCTION SortableChromosome(reference_name STRING)
  RETURNS STRING LANGUAGE js AS """
  // Remove the leading "chr" (if any) in the reference_name
  var chr = reference_name.replace(/^chr/, '');

  // If the chromosome is 1 - 9, prepend a "0" since
  // "2" sorts after "10", but "02" sorts before "10".
  if (chr.length == 1 && '123456789'.indexOf(chr) >= 0) {
    return '0' + chr;
  }

  return chr;
""";

SELECT
  reference_name,
  COUNT(reference_name) AS number_of_variant_rows
FROM
  `bigquery-public-data.human_genome_variants.platinum_genomes_deepvariant_variants_
WHERE
  EXISTS (SELECT 1
            FROM UNNEST(v.call) AS call, UNNEST(call.genotype) AS gt
          WHERE gt > 0)
GROUP BY
  reference_name
ORDER BY SortableChromosome(reference_name)
```

Both queries return the correct result, but their logic is more concise.

Improving query performance and reducing costs

BigQuery pricing (https://cloud.google.com/bigquery/pricing) is based on the number of bytes processed for a query. Query performance also improves when the amount of data processed is reduced. The BigQuery UI provides data on how many seconds have elapsed since a query started and how many bytes the query processed. See the BigQuery query plan explanation (https://cloud.google.com/bigquery/query-plan-explanation) for information on optimizing your queries.

Some of the examples in this page, such as Counting the variant calls in a table (#counting_variant_calls_in_the_table), demonstrate multiple ways to write a query. To determine which method of querying is best for you, examine the duration of different queries and see how many bytes of data they process.

Cleaning up

After you've finished the tutorial, you can clean up the resources you created on Google Cloud so you won't be billed for them in the future. The following sections describe how to delete or turn off these resources.

Deleting the project

The easiest way to eliminate billing is to delete the project you used for the tutorial.

To delete the project:

Warning: Deleting a project has the following consequences:

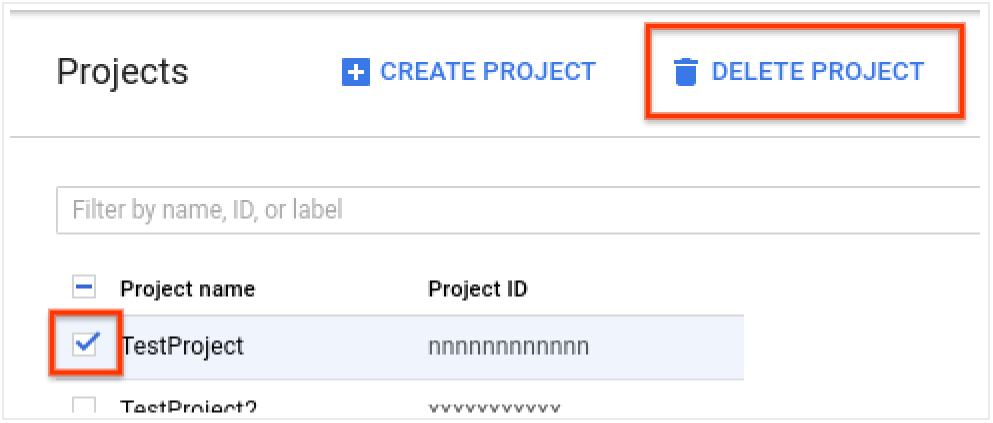
- If you used an existing project, you'll also delete any other work you've done in the project.
- You can't reuse the project ID of a deleted project. If you created a custom project ID that you plan to use in the future, delete the resources inside the project instead. This step ensures that URLs that use the project ID, such as an `appspot.com` URL, remain available.

If you are exploring multiple tutorials and quickstarts, reusing projects instead of deleting them prevents you from exceeding project quota limits.

1. In the Cloud Console, go to the Projects page.

   GO TO THE PROJECTS PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/IAM-ADMIN/PROJECTS)

2. In the project list, select the project you want to delete and click **Delete project**.



Project name	Project ID
TestProject	nnnnnnnnnnnn
TestProject2	xxxxxxxxxxxx

3. In the dialog, type the project ID, and then click **Shut down** to delete the project.

What's next

- Work through the other Cloud Life Sciences tutorials (https://cloud.google.com/life-sciences/docs/tutorials).
- Analyze variants in BigQuery using R, RMarkdown, or JavaScript (https://github.com/googlegenomics/getting-started-bigquery).