Click Here

BigQuery supports a wide range of data types including string, integer, float, boolean, date, datetime, geography, interval, JSON, and numeric types. The numeric types include INT64, NUMERIC, BIGNUMERIC, FLOAT64, and BIGDECIMAL. The data type list includes arrays, boolean, bytes, date, datetime, geometry, interval, JSON, numeric types, range, string, struct, time, and timestamp. When storing and querying data, it's essential to keep in mind the nullable and orderable properties of each data type. When sorting data, zeros and negative infinity values are considered equal. Groupable data types can appear after GROUP BY or DISTINCT commands, except for floating point numbers. Floating point numbers with special values like NULL or NaN are grouped together. Arrays are groupable if their elements are groupable, and two arrays belong to the same group if they have the same number of elements and corresponding elements are in the same groups. Grouping structs is also possible if their field types are groupable. Two structs are in the same group if both are NULL or all corresponding field values are in the same groups. When comparing data, values with the same type can be compared directly. However, comparisons involving ranges handle lower and upper bounds differently, with NULL bounds being sorted accordingly. Data types supporting comparisons can be used in JOIN conditions, such as geography values which require ST_Equals for comparison. Data Types Explained: Join Conditions, Collatable Data Types, and More Join conditions provide an explanation for collatable data types that support collation, determining how to sort and compare strings. These data types support collation: String fields in a struct String elements in an array Data Type Sizes: | Data Type | Size (logical bytes) | | --- | --- | | ARRAY | Sum of element sizes | | BIGNUMERIC | 32 | | BOOL | 1 | | BYTES | 2 + value size | | DATE | 8 | | DATETIME | 8 | | FLOAT64 | 8 | | GEOGRAPHY | 16 + 24 x number of vertices | | INT64 | 8 | | INTERVAL | 16 | | JSON | UTF-8 encoded string size | | NUMERIC | 16 | | RANGE | 16 | | STRING | 2 + UTF-8 encoded string size | | STRUCT | 0 + contained field sizes | | TIME | 8 | | TIMESTAMP | 8 | NULL values are calculated as 0 logical bytes. Repeated columns store data as arrays, with sizes based on the column type and number of values. Parameterized Data Types: Syntax: DATA_TYPE(param[, ...]) Use parameters to specify constraints for STRING, BYTES, NUMERIC, and BIGNUMERIC data types. Declare a parameterized data type with the DATA_TYPE syntax. Enforce constraints when writing to a parameterized column or assigning to a script variable. Example: DECLARE x STRING(10); SET x = "hello"; DECLARE x NUMERIC(10) DEFAULT 12345; DECLARE y NUMERIC(5, 2) DEFAULT 123.45; An array in GoogleSQL is an ordered list of zero or more non-array values, with all elements sharing the same type. Arrays cannot contain other arrays directly; instead, a struct must be used to separate them. NULL arrays are treated as empty arrays, and writing a NULL array to a table converts it to an empty array. However, queries that produce arrays containing NULL elements raise an error. Array types can have complex element types, except for arrays themselves, which cannot contain other arrays. Examples of array types include simple arrays, parameterized bytes, and arrays of structs. SELECT [a, b, c] FROM (SELECT CAST(5 AS INT64) AS a, CAST(37 AS FLOAT64) AS b, 406 AS c); SELECT ARRAY[1, 2, 3] AS floats; SELECT GENERATE_ARRAY(11, 33, 2) AS odds; SELECT GENERATE_DATE_ARRAY('2017-11-21', '2017-12-31', INTERVAL 1 WEEK) AS date_array; The BYTES type represents variable-length binary data that cannot be used interchangeably with strings. Most string functions are also applicable to bytes, but they operate on raw bytes instead of Unicode characters. Casting between strings and bytes enforces UTF-8 encoding. A parameterized bytes type, BYTES(L), is a sequence of bytes with a maximum of L bytes allowed in the binary string, where L is a positive INT64 value. If the sequence exceeds L bytes, it throws an OUT_OF_RANGE error. The DATE type represents a Gregorian calendar date, independent of time zone, ranging from 0001-01-01 to 9999-12-31. A date value doesn't represent a specific 24-hour time period and may vary across time zones during daylight saving time (DST) transitions. The DATETIME type represents a Gregorian date and time, including the year, month, day, hour, minute, second, and subsecond, ranging from 0001-01-01 00:00:00 to 9999-12-31 23:59:59.999999. It's independent of time zone and can be displayed as a civil date and time part. The GEOGRAPHY type is a collection of points, linestrings, and polygons represented as a point set or a subset of the Earth's surface, based on the OGC Simple Features specification (SFS). It includes geography objects such as points, linestrings, and polygons. Given article text here Polygon Representation in GeoJSON A polygon is represented as a planar surface defined by 1 exterior boundary and 0 or more interior boundaries. Each interior boundary defines a hole in the polygon, with boundary loops oriented to traverse them counterclockwise if you move around the vertices in order, resulting in the interior being on the left side. GeoJSON Polygon Syntax The syntax is POLYGON(interior_ring[, ...]), where interior_ring consists of (point[, ...]). Examples include: - POLYGON((0 0, 2 2, 2 0, 0 0), (2 2, 3 4, 2 4, 2 2)) - POLYGON EMPTY MultiPoint in GeoJSON A collection of points is represented by MULTIPOINT(point[, ...]). An example is: - MULTIPOINT(0 32, 123 9, 48 67) MULTIPOINT EMPTY MultiLineString in GeoJSON Represents a multilinestring, which is a collection of linestrings, using the syntax MULTILINESTRING((linestring)[, ...]). An example includes: - MULTILINESTRING((2 2, 3 4), (5 6, 7 7)) MULTILINESTRING EMPTY MultiPolygon in GeoJSON Represents a multipolygon, which is a collection of polygons, using the syntax MULTIPOLYGON((polygon)[, ...]). An example includes: - MULTIPOLYGON(((0 -1, 1 0, 1 1, 0 -1)), ((0 0, 2 2, 3 0, 0 0), (2 2, 3 4, 2 4, 1 9))) MULTIPOLYGON EMPTY GeometryCollection in GeoJSON Represents a geometry collection with elements of different dimensions or an empty geography, using the syntax GEOMETRYCOLLECTION(geography_object[, ...]). An example includes: - GEOMETRYCOLLECTION(MULTIPOINT(-1 2, 0 12), LINESTRING(-2 4, 0 6)) GEOMETRYCOLLECTION EMPTY Simple Arrangements and Empty Geometries The points, linestrings, and polygons in a geography value form a simple arrangement on the WGS84 reference ellipsoid. A simple arrangement is one where no point on the WGS84 surface is contained by multiple elements of the collection. If self-intersections exist, they are automatically removed. An empty geography isn't associated with a particular geometry shape. For example: - SELECT ST_GEOGFROMTEXT('POINT EMPTY') AS a, ST_GEOGFROMTEXT('GEOMETRYCOLLECTION EMPTY') AS b Result: | a | b | |----------------| | GEOMETRYCOLLECTION EMPTY | GEOMETRYCOLLECTION EMPTY | Compound Geometry Objects The structure of compound geometry objects isn't preserved if a simpler type can be produced. For example, in column b, GEOMETRYCOLLECTION with (POINT(1 1) and POINT(2 2) is converted into the simplest possible geometry, MULTIPOINT(1 1, 2 2). - SELECT ST_GEOGFROMTEXT('MULTIPOINT(1 1, 2 2)') AS a, ST_GEOGFROMTEXT('GEOMETRYCOLLECTION(POINT(1 1), POINT(2 2))') AS b Result: | a | b | |----------------| | MULTIPOINT(1 1, 2 2) | MULTIPOINT(1 1, 2 2)| An INTERVAL object represents a duration or time period without specifying a particular point in time. It can be created using various methods. One way is to use an interval literal that supports only one datetime part or a range of datetime parts. For example, to create an interval representing 1 year and 0 months, you can use the following syntax: INTERVAL [sign]Y-M [sign]D [sign]H:M:S[.F] Where Y is the year, M is the month, D is the day, H is the hour, M is the minute, S is the second, and [.F] represents up to six fractional digits (microsecond precision). Another way is to use an INT64 expression and one interval-supported datetime part, as shown in examples: INTERVAL int64_expression datetime_part For instance, "1-0 0:0:0" represents 1 year, 0 months, 0 days, 0 hours, 0 minutes, and 0 seconds. Alternatively, you can construct an INTERVAL object using a string that contains the datetime parts you want to include, a starting datetime part, and an ending datetime part. This format is represented as: INTERVAL datetime_parts_string starting_datetime_part TO ending_datetime_part For example, "2-11 YEAR TO MONTH" represents 0 years, 8 months, and so on. There are also specific formats for interval-supported datetime parts, such as Y-M D, Y-M D H, Y-M D H:M, etc. These formats allow you to specify a range of values within an INTERVAL object. The examples provided in the original text demonstrate various ways to construct an INTERVAL object, including using interval literals and string formats with specific syntax. The following date components can be used to create time intervals: YEAR (Y) representing the number of years, QUARTER (Q) converted to three months (M), MONTH (M) with each 12 months equating to one year. WEEK (W) and DAY (D) are also usable, with weeks being equivalent to seven days (D). For time components: HOUR (H), MINUTE (M), SECOND (S) with 60 seconds converting to one minute, and MICROSECOND (US) for up to six fractional digits of precision. JSON data can be represented using a lightweight format, preserving booleans, strings, and nulls exactly while ignoring whitespace characters. JSON values can store integers between -9.223372036854775808 and 18.446744073709551615 and floating-point numbers within the FLOAT64 domain. Array elements maintain their order, but object member order is not guaranteed or preserved, with up to 500 levels of nesting allowed. Numeric types include INT64 (with aliases INT, SMALLINT, INTEGER, BIGINT, TINYINT, BYTEINT) and DECIMAL types (NUMERIC, BIGNUMERIC). The former can represent integers from -9.223372036854775808 to 18.446744073709551615 while the latter allows decimal fractions with a maximum precision of 38 and scale of 9. BigQuery data types for numerical values. ####DECIMAL Type - Precision and scale limits: max(1, S) ≤ P ≤ S + 29. - Scale range: 0 ≤ S ≤ 9. - If a value has more than S decimal digits, it is rounded to S decimal digits. ####BIGNUMERIC Type - Maximum precision and scale limits: max(1, S) ≤ P ≤ S + 38. - Scale range: 0 ≤ S ≤ 38. - If a value has more than S decimal digits, the value is rounded to S decimal digits. ####Floating Point Type - Double precision numeric values with fractional components. - Special non-numeric values: NaN and +/-inf. - Arithmetic operators provide standard IEEE-754 behavior for all finite input values that produce finite output. - Function calls and operators return an overflow error if the input is finite but the output would be non-finite. - All such cases are noted explicitly in Mathematical functions. Floating Point Arithmetic Considerations and Range Type Usage for Deterministic Results Given article text here Strings in this system count characters, not bytes, and assign a numeric code point to each Unicode character. Lower code points mean lower characters. Most string functions operate on bytes rather than Unicode characters. Strings and bytes are separate types that can't be converted implicitly. Explicit casting between them does UTF-8 encoding and decoding; casting bytes to strings fails if the bytes aren't valid UTF-8. There's a parameterized string type STRING(L) with a maximum of L Unicode characters allowed, where L is an INT64 value. If more than L characters are assigned, it throws an OUT_OF_RANGE error. Struct types are declared using angle brackets and can contain ordered fields with arbitrary types. They're used in various scenarios such as declaring simple structs like STRUCT for a single integer field or complex structs with parameterized string fields like STRUCT. Structs can be constructed using tuple syntax (expr1, expr2 [, ...]) to create an anonymous struct type. This syntax is also used in comparisons and WHERE clauses. The output of these operations are STRUCT types that derive their field data types from the input expressions or column data types. String structures can be defined with an explicit data type using typed syntax. The output type is determined by the provided field type, and the input expression is coerced to match if the types are different. The number of expressions must equal the number of fields in the type, and the expression types must be coercible or literal-coercible to the field types. Examples of typed syntax include: * `STRUCT(5)` - A struct with an integer value of 5 * `STRUCT("2011-05-05")` - A struct with a date value of "2011-05-05" * `STRUCT(1, t.str_col)` - A struct with two fields: x as an integer and y as a string Structs can be directly compared using equality operators such as equal (=), not equal (!= or) and [NOT] IN. However, this comparison is done pairwise in ordinal order, ignoring any field names. Time types represent a time of day, independent of date and time zone. Timestamp values represent an absolute point in time, with microsecond precision. Canonical formats for time and timestamp literals include: * Time: `[H]H:[M]M:[S]S[.F]` * Timestamp: `civil_ date_part[time_part [time_zone]] | civil_date_part[time_part[time_zone_offset]]` Note that timestamps themselves do not have a time zone, but may be displayed with one for human readability. Given article text here 2023-04-01T12:00:00+02:00 When working with timestamps, it's essential to use a space between the time zone name and the rest of the timestamp. For example, "2014-09-27 12:30:00.45 America/Los_Angeles". However, not all time zones are interchangeable, even if they report the same time during certain periods. Leap seconds don't affect timestamp computations, which use Unix-style timestamps that don't account for leap seconds. Instead, these seconds can be observed through functions that measure real-world time. When it comes to data types, a classification governs how compilers and programming languages collect, store, and interpret data. Google BigQuery is a fully managed enterprise data warehouse with built-in machine learning and business intelligence capabilities. It enables users to process large datasets and supports various data types, including real-time and batch data in structured, semi-structured, or unstructured formats. BigQuery currently offers eight supported data types: NUMERIC (INT64, NUMERIC, BIGNUMERIC, FLOAT64), BOOLEAN, STRING, BYTES, GEOGRAPHY, ARRAY, STRUCT. These data types are crucial for manipulating and analyzing data operations. Numerical data types in BigQuery include INT64, NUMERIC DECIMAL, BIGNUMERIC, and FLOAT64. These types allow for various mathematical operations and are suitable for financial calculations. The main difference between these types lies in their precision, ranging from 38 decimal digits for NUMERIC to 76.76 for BIGNUMERIC. FLOAT64 supports double-precision numeric values with fractional components and includes unique non-numeric values like NaN, +inf, and -inf. STRING STRING is UTF-8 encoded variable-length character data, commonly used for storing user-generated values such as billing addresses, usernames, survey replies, tweets, and email addresses. It must be quoted with either single ('), double ("), or triple quotation marks. Given article text here 2022-12-12 10:59:13 is a TIMESTAMP value that represents an exact point in time with microsecond precision on a particular day in a time zone. It has the same meaning as the DATE data type but includes the letter T separating the date part and the time part of the values. The format of a TIMESTAMP is YYYY-[M]M-[D]D[|T][H]H:[M]M:[S]S[.DDDDDD]][time zone]. It ranges from 0001-01-01 00:00:00 to 9999-12-31 23:59:59.999999. When working with TIMESTAMP, the time zone is assumed to be in UTC by default and can be precise down to microseconds unless a specific time zone is specified. SELECT cast('2022-12-12 10:59:13.0245-2:00' AS TIMESTAMP) as timestamp The TIMESTAMP data type supports mathematical operations with date and time values, allowing users to answer business questions such as "How long did it take product A to be delivered after it was shipped?" The GEOGRAPHY data type represents a location on Earth using longitude/latitude values and can be used to build maps and routes. It is based on the Open Geospatial Consortium's Simple Features specification. SELECT cast(ST_GEOGFROMTEXT('POINT(6.4550575,3.3941795)') as GEOGRAPHY) geography The spatial functions of BigQuery support SQL/MM 3 specification. The ARRAY data type is an ordered list of zero or more elements of any non-ARRAY type, but the elements must share the same type. SELECT cast(['sync', 'store', 'access'] as ARRAY) STRUCTs are containers of ordered fields with required data types and optional field names, supporting NULL values. They can be declared using angle brackets and contain arbitrarily complex elements. SELECT cast((27,'Panoply') as STRUCT) struct Get instant access to all your business data with Panoply! Seamlessly integrate multiple sources of data into a single, cloud-based repository, effortlessly mapping out data types, analyzing trends, and gaining valuable insights using BigQuery. Take the first step by requesting a complimentary demo today and discover the power of unified data management. This article was crafted by Ifeanyi Benedict Iheagwara, a seasoned data analyst and Power Platform developer driven by technical writing, open-source contributions, and community building. With expertise in machine learning, data science, and DevOps, Ifeanyi shares knowledge on these topics while actively participating in global ecosystems.

Bigquery supported formats. Bigquery data types. Data types sql bigquery. Bigquery types. Bigtable supported data types. Bigquery date type.