Parquet format data types

Back to the Parquet glossary is an open source file format available for any project in the Hadoop ecosystem. Apache Parquet is designed for an efficient flat columnar storage format and performing data compared to files based on rows such as CSV or TSV files. Parquet uses the algorithm of shredding and assembly of registers that is superior to the simple flattening of nested namespaces. The parquet is optimized to work with complex data in bulk and presents different modes of efficient data compression and coding type. This approach is best especially for those queries that need to read certain columns from a large table. Parquet can only read the necessary columns, therefore, greatly minimize the IO. Advantages of storing data in Columnar format: Columnar storage such as Apache Parquet is designed to bring efficiency compared to row-based files such as CSV. When querying, columnar storage can be skipped on non-relevant data very quickly. As a result, aggregation queries consume less time compared to row-oriented databases. This form of storage has resulted in hardware savings and minimizes latency to access data. Apache Parquet is built from the ground upwards. It is therefore capable of supporting advanced nesting data structures. The layout of parquet data files is optimized for queries that process large volumes of data, in the gigabyte range for each individual file. The parquet flooring is built to support flexible compression options and efficient coding schemes. Because the data type for each column is quite similar, the compression of each column is simple (which makes queries even faster). Data can be compressed using one of several available codecs; as a result, different data files can be compressed differently. Apache Parquet works best with interactive, server-free technologies like AWS Athena, Amazon Redshift Spectrum, Google BigQuery and Google Dataproc. Difference Between Parquet and CSV CSV is a simple and widely extended format that uses many tools such as Excel, Google Sheets, and many others can generate CSV files. While CSV files are the default format for data processing pipes, it has some drawbacks: Amazon Athena and Spectrum will charge based on the amount of data scanned per query. Google and Amazon will charge you based on the amount of data stored on GS/S3. Google Dataproc charges are time-based. Parquet has helped its users to reduce storage requirements by at least a third in large data sets, in addition, it has greatly improved the time of scanning and de-marketing, hence the costs The following table compares the savings, as well as the speed obtained by converting data into Parquet from CSV. Data Set Size on Amazon S3 Query Runtime Data Scanned Cost Data Stored as CSV Files 1 TB 236 seconds 1.15 TB $5.75 Data Stored in Apache Parquet Format 130GB 6 .6 78 seconds 2.51GB $0.01 Savings 87% less when using 34x parquet faster 99% less scanned data 99.7% savings Extra resources Back to glossary Apache Parquet Flooring the following features: The self-description of the Columnar format Self-description data independent of the language embeds the schema or structure with the data ids themselves. Hadoop's use cases drive the growth of self-defed data formats, such as Parquet and JSON, and NoSQL databases, such as HBase. These formats and databases are very suitable for the agile and iterative development cycle of hadoop and BI/analytics applications. Optimized to work with large files, Parquet has data in columns, putting related values in close proximity to each other to optimize query performance, minimize I/O, and facilitate compression. The parquet detects and encodes the same or similar data using a technique that retains resources. Apache Drill includes the following support for Parquet: Consultation of self-dec data description in files or NoSQL databases without having to define and manage definitions of overlap of schemes in centralized metastores Creation of parquet files of other file formats, such as JSON, without any configuration generating parquet files that have evolved or changing schemes and consulting the data on the fly Handling of parquet data types Reading parquet files When a data reading of Parquet occurs, Drill only loads the necessary data columns, which reduces the I/O. Reading only a small part of the parquet data from a file or data table, Drill can examine and analyze all the values of a file or table of data. You can create a Drilling table from one format and store the data in another format, including parquet floors. Writing CREA TAULA COM (CTAS) parquet files can use any data source provided by the storage plugin. To write Parquet data using the CTAS command, set the store.session format option as shown in The Parquet Storage Format Settings. Alternatively, configure the storage plugin to point to the directory containing the parquet files. Although the data resides in a single table, parquet output generally consists of several files that resemble mapreduce output with numbered filenames, such as 0_0_0.parquet in a directory. Starting with Drill 1.10, Drill writes standard parquet date values. Drill also has an AutoCorrect feature that automatically detects and corrects the damaged date values that Drill wrote to Parquet files before Drill 1.10. By default, the AutoCorrect feature is turned on and works for dates up to 5,000 years in the future. In the unlikely event that Drill needs to write dates thousands of years in the future, turn off the AutoCorrect feature. To turn off AutoCorrect, go to storage plugin settings and change the in the parquet configuration to false, as shown in the following example: formats: { parquet: { type: parquet, autoCorrectCorruptDates: false } Alternatively, you can set the option to false when you issue a query, as shown in the following example: SELECT l_shipdate, l_commitdate l_commitdate (type => 'parquet', autoCorrectCorruptDates => false)) LIMIT 1; Configuring the parquet storage format To read or write Parquet data, you must include the Parquet format in the storage connector format definitions. The dfs connector definition includes the Parquet format. Use the store.format option to set the CTAS output format of a group of Parquet rows at session or system level. Use the ALTER command to set the store.format option. ALTER SYSTEM| SET SESSION 'store.format' = 'parquet'; Configuring the size of parquet files By configuring the size of parquet files by setting the store.parquet.block size can improve writing performance. The block size is the size of MFS, HDFS or the file system. The larger the block size, the more memory the drill needs for buffer data. Parquet files containing a single block maximize the amount of data Drill stores adantiguously on disk. Given a single row pack per file, Drill stores the entire Parquet file in the block, avoiding network I/O. To maximize performance, set the target size of a group of Parquet rows to the number of bytes less than or equal to the block size of MFS, HDFS, or the file system by using the store.parquet.block size option, as shown in: ALTER SYSTEM| SET SESSION 'store.parquet.block-size' = 536870912; The default block size is 536870912 bytes. The HDFS block size setting for the 1.11 parquet file drill introduces the store.parquet.writer.use_single_fs_block option, which allows Drill to write a Parquet file as a single block of the file system without changing the default block size of the file system. Query performance improves when Drill reads parquet files as a single block of the file system. When the store.parquet.writer.use_single_fs_block is enabled, the store.parquet.block option determines the block size of the parquet files created. The default option store.parquet.writer.use_single_fs_block option is 'false'. Use the SET command to enable or disable the option, as shown by ALTER SYSTEM| SET SESSION store.parquet.writer.use_single_fs_block = 'true|false'; Type mapping The high correlation between parquet and SQL data types means that the reading of Parquet files has no effort in Drill. Writing to Parquet files has more work to read. Because SQL doesn't support all types of Parquet data, to prevent Drill from inferring a type other than one you want, use the Drill cast feature offers more liberal casting capabilities than SQL for Parquet conversions if parquet data is of a logical type. The following general process converts a JSON file to Parquet: Create or use an existing storage plugin that specifies the storage location of the Parquet file, the mutability of the data supported file formats. Check out the JSON data. Create a table that selects the JSON file. In the CTAS command, it emits JSON string data to the corresponding SQL types. Example: Read JSON, Writing Parquet This example demonstrates a definition of the storage plugin, an example row of data from a JSON file, and a drilling query that the JSON entrance at the Parquet exit. Setting the storage plugin You can use the default dfs storage plugin installed on your system. The storage plugin must configure the writable option of the workspace to true, so that Drill can write the Parquet output. The dfs storage plugin defines the tmp recordable workspace, which you can use in the CTAS command to create a parquet table. JSON Data Sample Row A JSON file called sample.json contains data consisting of strings, typical of JSON data. The following example shows a row in the JSON file: {trans_id:0,date:2013-07-26,time:04:56:59,amount:80.5,user_info: {cust_id:28,device:WEARABLE2 ,state:mt},marketing_info: {camp_id:4,keywords: [go, [go, to,thing, watch, made, laugh, might, pay, in, your, hold] }, trans_info: {prod_id:[16], purch_flag:false } } CTAS Query The following example shows a CTAS query that creates a table from JSON data shown in the last example. The command emits date, time, and quantity strings to SQL DATE, TIME, and DOUBLE types. The string-to-VARCHAR shredding of the other strings occurs automatically. CREATE TABLE dfs.tmp.sampleparquet AS (SELECT trans_id, cast('date' AS date) transdate, cast('time' AS time) transtime, cast(amount AS double) amountm, user_info, marketing_info, trans_info FROM dfs.' /Users/drilluser/sample.json'); The CTAS query does not specify a file name extension for the output. Drill creates a default parquet file, as indicated by the file name on output: +------------+----------------------------+ | Fragment | Number of written records | +------------+----------------------------+ | 0_0 | May 10, 2018 +------------+----------------------------+ 1 row selected (1.369 seconds) You can check the Parquet file to verify that Drill now interprets the converted string as a date. SELECT extract(year from transdat) AS Year, t.user_info.cust_id as dfs client.tmp.'sampleparquet't; +------------+------------+ | Year of the year | The customer | +------------+------------+ | 2013 | 28 | 2013 | 86623 | 2013 | 11 | 2013 | 666 | 2013 | 999 | +------------+------------+ 5 selected rows (0.039 seconds) For more examples and information on the use of Parquet data, see Evolution of parquet as an autodescracy data format - New paradigms for consuming Hadoop data. SQL data type for parquet The first table in this section assigns SQL data types to parquet data types, intentionally limited by parquet creators to minimize the impact on disk storage: SQL Parquet Type Description BIGINT INT64 8-byte signed enter BOOLEAN BOOLEAN TRUE (1) or FALSE (0) N / A BYTE_ARRAY Arbitrarily long byte array FLOAT FLOAT 4 byte single precision floating point number DOUBLE DOUBLE DOUBLE 8-byte double precision floating point number INTEGER INT32 4-byte signed integer INT96 12-byte signed int * Drill 1.10 and later can implicitly interpret the parquet type INT96 as TIMESTAMP (standard 8 byte / millisecond millisecond when store.parquet.reader.int96_as_timestamp is enabled. In earlier versions of Drill (1.2 to 1.9) or when the store.parquet.reader.int96_as_timestamp option is off, you must use the CONVERT_FROM function for Drill to correctly interpret the INT96 values as TIMESTAMP values. On INT96 support From Drill 1.10, Drill can implicitly interpret the INT96 timestling data type in Parquet files when the store.parquet.reader.int96_as_timestamp option is enabled. For earlier versions of Drill, or when the store.parquet.reader.int96_as_timestamp option is disabled, you must use the CONVERT_FROM function, the store.parquet.reader.int96_as_timestamp option is disabled by default. Use SYSTEM ALTER| SESSION SET command to enable the option. Unnecessarily enabling this option can cause queries to fail because the CONVERT_FROM(glue, 'TIMESTAMP_IMPALA') function does not work store.parquet.reader.int96_as_timestamp is not enabled. Using CONVERT_FROM to play INT96 In earlier versions of Drill (1.2 to 1.9), you must use the CONVERT_FROM function for Drill to interpret the type of Parquet INT96. For example, to decode a Beehive or Impala timestix, which is INT96 type, use the CONVERT_FROM function and TIMESTAMP_IMPALA type argument: SELECT CONVERT_FROM(timestamp_field, 'TIMESTAMP_IMPALA') as timestamp_field OF 'dfs.file_with_timestamp.parquet'; Because INT96 only supports readings, you cannot use the TIMESTAMP_IMPALA as a data type argument with CONVERT_TO. You can convert a SQL TIMESTAMP to VARBINARY using the CAST function, but the resulting VARBINARY is not the same as INT96. For example, create a Drilling table after reading INT96 and convert some data into a timestem mark. CREATE A TABLE T2(c1) AS SELECT CONVERT_FROM(created_ts, 'TIMESTAMP_IMPALA') FROM T1 COMMAND BY 1 LIMIT 1; t1.created_ts is an INT96 (or Hive/Impala) timest t2.created_ts; These types are not comparable. You cannot use a condition like t1.created_ts = t2.created_ts. Setting the Default time zone, the INT96 timestamp values represent the local date and time, which is similar to Hive. For INT96 timestage mark values in UTC, set the drill for UTC time. SQL type parquet logical types Parquet also supports logical types, fully described on the site of Parquet Apache. Embedded types, JSON and BSON, note a binary primitive type representing a JSON or BSON document. The logical types and their allocation to type SQL are: SQL Type Broken down parquet Logical type Parquet Description DATE Months and days in the form of date YYYY-MM-DD, not including the time of day. Use int32 annotation. It stores the number of days since Unix time, January 1, 1970. VARCHAR UTF8 character string variable length Write down the binary primitive type. The byte array is interpreted as a utf-8 encoded character string. No INT_8 8-bit, signed INT_16 16-bit, usigned 4-byte INT signed integer INT_32 32-bit, signed 8-byte double precision floating point number INT_64 64-bit, signed UINT_8 8-bit, unsigned UINT_16 16-bit, unsigned UINT_32 No 32-bit, unsigned UINT_64 64-bit, unsigned decimal* 38-digit decimal precision Decimal numbers signed of arbitrary precision of the unscaled formValora * 10 ^(scale) TIME Hours, minutes, seconds, milliseconds; Base 24 hours TIME_MILLIS logical time, not including the date. Write down int32. The number of milliseconds after midnight. TIMESTAMP Year, month, day and according TIMESTAMP_MILLIS logical date and time. Write down an int64 that stores the number of milliseconds from the Time of Unix, 00:00:00.000 on January 1, 1970, UTC. INTERVAL Integer fields that represent a time period depending on the INTERVAL range type A time range. Write down a fixed_len_byte_array length of 12. Months, days and ms in unending encoding. * From drill 1.14, the DECIMAL data type is enabled by default. Description of Language Support Parquet data supports the following data description languages: Apache Avro Apache Thrift Google Protocol Buffers Implement custom storage plugins to create readers/parquet writers for formats such as Thrift. Saving.