COLUMN STORE METHOD, DEVICE, COMPUTER PROGRAM PRODUCT, AND STORAGE MEDIUM

The present disclosure claims priority to Chinese Patent Application No. 202410232858.4, filed with the China National Intellectual Property Administration on February 29, 2024 and entitled "COLUMN STORE METHOD, DEVICE, COMPUTER PROGRAM PRODUCT, AND STORAGE MEDIUM", which is incorporated herein by reference in its entirety.

TECHNICAL FIELD

The present disclosure relates to the field of database technologies, and in particular, to a column store method, a device, a computer program product, and a storage medium.

BACKGROUND

Currently, a data storage manner in a database is usually row store. However, in a data analysis and query scenario, column store is more advantageous. This is because the column store organizes data based on a column dimension, and a quantity of times of reading and writing of a storage medium can be effectively reduced during data analysis and query, thereby improving performance.

Therefore, an increasing number of database products support conversion from the row store to the column store. However, an online service of the row store usually needs to be blocked in an existing conversion solution. Conversion is completed in the blocked state, and the online service of the row store can be restored only after the conversion is completed. As a result, the database products cannot provide the online service based on the row store within a period of time, and the more stock data in the row store, the greater the time cost, which causes significant inconvenience to users.

SUMMARY

Various aspects of the present disclosure provide a column store method, a device, a computer program product, and a storage medium, to implement column store without affecting provision of an online service based on row store.

Embodiments of the present disclosure provide a column store method, including:

pulling, in response to a column store triggering event, data rows from a target data table in sequence to perform column store, to generate a column store file and a first primary key index corresponding to the column store file;

synchronizing a data operation performed on the target data table and monitored during the pulling, to generate an incremental file for storing an incremental data row and a second primary key index corresponding to the incremental file;

recording, based on the data operation monitored during the pulling, a to-be-deleted primary key for the first primary key index; and

combining, in response to an end of the pulling, the first primary key index and the second primary key index into a target primary key index, and correcting the target primary key index and the column store file based on the to-be-deleted primary key.

Embodiments of the present disclosure further provide an electronic device, including a memory, a processor, and a communication component, where

the memory is configured to store one or more computer instructions; and

the processor is coupled to the memory and the communication component, and is configured to execute the one or more computer instructions, to perform the foregoing column store method.

Embodiments of the present disclosure further provide a computer-readable storage medium storing a computer program. The computer program, when executed by one or more processors, causes the one or more processors to perform the foregoing column store method.

Embodiments of the present disclosure further provide a computer program product, including a computer program. The computer program, when executed by one or more processors, causes the one or more processors to perform the foregoing column store method.

In the embodiments of the present disclosure, on the premise that normal provision of the online service based on the row store is maintained, the data rows can be pulled from the target data table in sequence to perform the column store in response to the triggering event, to generate the column store file and populate the first primary key index for the column store file based on the actually pulled data rows. In addition, the data operation performed on the target data table and monitored during the pulling can be further synchronized, to generate the incremental file for existing the incremental data row and the second primary key index corresponding to the incremental file. This can ensure consistency between column store and row store in terms of the increment. Furthermore, based on the data operation monitored during the pulling, the to-be-deleted primary key can be recorded for the first primary key index. In response to the end of the pulling, the first primary key index and the second primary key index are combined into the target primary key index, and the target primary key index and the column store file are corrected based on the to-be-deleted primary key. In this way, the data operation performed during the pulling on the stock aspect can be replicated to the target primary key index and the column store file through the recorded to-be-deleted primary key, to ensure consistency between the column store and the row store in terms of the stock.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings described herein are used to provide a further understanding of

the present disclosure and constitute a part of the present disclosure. Exemplary embodiments of the present disclosure and descriptions thereof are used to explain the present disclosure, and do not constitute any improper limitation on the present disclosure. In the accompanying drawings,

FIG. 1 is a schematic flowchart of a column store method according to an exemplary embodiment of the present disclosure;

FIG. 2 is a schematic diagram of logic of a column store method according to an exemplary embodiment of the present disclosure;

FIG. 3 is a schematic diagram of an architecture of an exemplary database system according to an exemplary embodiment of the present disclosure;

FIG. 4 is a schematic diagram of an internal storage structure of a column store file according to an exemplary embodiment of the present disclosure;

FIG. 5 is a schematic diagram of an exemplary processing solution for correcting a column store file according to the present disclosure; and

FIG. 6 is a schematic diagram of a structure of an electronic device according to another exemplary embodiment of the present disclosure.

DETAILED DESCRIPTION

To make objectives, technical solutions, and advantages of the present disclosure clearer, the technical solutions of the present disclosure are clearly and completely described below with reference to specific embodiments of the present disclosure and corresponding accompanying drawings. Apparently, the described embodiments are merely some of the embodiments of the present disclosure, but not all of the embodiments. All other embodiments obtained by a person of ordinary skill in the art based on the embodiments of the present disclosure without creative efforts fall within the protection scope of the present disclosure.

Before the technical solutions provided in the embodiments of the present disclosure are described in detail, several technical concepts involved in the present disclosure are briefly explained below.

In row store, data storage is performed by using a row as a unit, and an entire row of data is used as a storage unit.

In column store, data storage is performed by using a column as a unit. In a process of converting the row store into the column store, a group of data rows may be stored as a column store file, and data storage is performed in the column store file by using a column as a unit. That is, a column of data may be used as a storage unit (which may be referred to as a column segment (column segment)) in the column store file.

A primary key, also referred to as a primary keyword (primary key), is one or more fields in

a data table, and a value of the primary key is used for uniquely identifying a record in the data table (that is, used for identifying a data row in the data table).

As described in the background section, currently, during conversion from the row store to the column store, the online service of the row store needs to be blocked, so as to ensure data consistency between the column store and the row store. However, such a conversion solution in which the online service needs to be suspended causes users considerable inconvenience.

Therefore, embodiments of the present disclosure provide a column store solution, which can ensure data consistency between the column store and the row store without affecting the online service provided by the row store.

The data consistency may be understood as that a data state of the column store can be consistent with a data state of the row store. It should be understood that the column store usually lags behind the row store. Therefore, an objective of the embodiments of the present disclosure is to ensure that a data state obtained after the column store solution is performed can be consistent with a data state of the row store at a historical moment.

The technical solutions provided in the embodiments of the present disclosure are described in detail below with reference to the accompanying drawings.

FIG. 1 is a schematic flowchart of a column store method according to an exemplary embodiment of the present disclosure. FIG. 2 is a schematic diagram of logic of a column store method according to an exemplary embodiment of the present disclosure. The method may be performed by a data processing apparatus. The data processing apparatus may be implemented as software, hardware, or a combination of software and hardware, and the data processing apparatus may be integrated in an electronic device. Referring to FIG. 1, the method may include the following steps.

Step 100: Pull, in response to a column store triggering event, data rows from a target data table in sequence to perform column store, to generate a column store file and a first primary key index corresponding to the column store file.

Step 101: Synchronize a data operation performed on the target data table and monitored during the pulling, to generate an incremental file for storing an incremental data row and a second primary key index corresponding to the incremental file.

Step 102: Record, based on the data operation monitored during the pulling, a to-be-deleted primary key for the first primary key index.

Step 103: Combine, in response to an end of the pulling, the first primary key index and the second primary key index into a target primary key index, and correct the target primary key index and the column store file based on the to-be-deleted primary key.

The column store method provided in this embodiment is applicable to databases of various

architectures. FIG. 3 is a schematic diagram of an architecture of an exemplary database system according to an exemplary embodiment of the present disclosure. Referring to FIG. 3, the column store method provided in this embodiment may be applicable to a distributed database system. The distributed database system may include one or more computing nodes and one or more data nodes. The data node is used to store a data table, and the computing node may be configured to process various processing transactions on a data table, for example, various data operations in an online transaction processing (Online Transaction Processing, OLTP) scenario and various query and analysis operations in an online analytical processing (Online Analytical Processing, OLAP) scenario. In such an exemplary database system, the column store method provided in this embodiment can convert a data table stored in a row store manner on a data node into a data table stored in a column store manner. It should be noted that in this embodiment, the data table stored in the row store manner is not disturbed, nor is the data table stored in the row store manner deleted. Instead, a copy stored in the column store manner is generated for the data table stored in the row store manner. In actual application, various data operations are still performed on the data table stored in the row store manner, and the data table stored in the column store manner generated in this embodiment may be used to support the foregoing OLAP scenario, to improve query and analysis performance. In other words, in this embodiment, coexistence of the row store and the column store is supported, and data consistency between the column store and the row store can also be ensured.

An execution subject of the column store method is not limited in this embodiment. In some possible technical concepts, an independent electronic device may be added to the database system to serve as an execution subject in this embodiment. For example, an independent node may be added to the distributed database system shown in FIG. 3, to implement the column store method. In some other possible technical concepts, an existing node in the database system may alternatively be used as an execution subject in this embodiment. For example, in the distributed database system shown in FIG. 3, the data processing apparatus provided in this embodiment may be distributed and installed on various data nodes, to implement the column store method. In this way, the data node can perform the column store on its data table stored in the row store manner.

In addition, a data table type under the row store is not limited in this embodiment, and the column store for various types of data tables such as a common table, a partitioned table, and a bucketed table is supported in this embodiment. For ease of description, in this embodiment, an example in which a target data table is stored in a row store manner is used for description of the technical solution. It should be understood that the target data table may be any common table, any partition in the partitioned table, any bucket in the bucketed table, or the like. Specification of the target data table matches the type of the data table in the row store.

Referring to FIG. 1, in step 100, the data rows may be pulled from the target data table in sequence in response to the column store triggering event to perform column store, to generate the column store file and the first primary key index corresponding to the column store file.

As mentioned above, in this embodiment, the row store may normally provide the online service. In other words, after the column store triggering event occurs, data operations may be normally performed on the target data table, and these data operations also normally take effect on the target data table. The column store triggering event may be an event such as receiving a column store start instruction or starting to push an operation log by a log node in the database system. The log node mentioned herein may be used for recording, in a form of an event, all data operations performed on the data table stored in the row store manner. An operation log generated in the log node may be a binary format file (binlog).

In a research process, the inventors find that data operations performed on the target data table may usually include an insertion operation, a deletion operation, and an update operation.

The insertion operation (insert) is used to insert one or more data rows into the target data table.

The deletion operation (delete) is used to delete one or more data rows from the target data table.

The update operation (update) is used to update one or more data rows on the target data table. It should be noted that the update operation may be broken down into a deletion operation and an insertion operation. That is, the update operation may be understood as deleting a data row from the target data table first and then inserting a data row. It may be learned that the data operation causes the data row on the target data table to be incremented or deleted.

In actual application, there are usually many data rows in the target data table. Therefore, the pulling process in step 100 takes a specific amount of time to complete. Consequently, during the pulling, the data operation(s) may be performed on the target data table. That is, the target data table dynamically changes during the pulling. In view of this, it is provided in this embodiment that, in step 100, after the column store is started, the data is pulled from the target data table in sequence. In this pulling solution, pulling is performed step by step based on an actual data state in the target data table without considering a change situation in the target data table. In actual application, paginated query may be performed on the target data table. A page is pulled each time, and a single page may include several data rows. In step 100, after the data rows are pulled, the data rows may be stored in a column store manner, to generate the column store file. In this way, in step 100, as the pulling operation is performed, a plurality of column store files are gradually generated for the target data table. Optionally, the column store file in this embodiment may use a format such as ORC, which is not limited herein.

For example, in step 100, after the column store is started, the first page may start to be pulled from the target data table, and the first page corresponds to data rows 1 to 10000. After this page is pulled, the column store may be performed on the data rows within this page, to obtain column store file 1. After the previous page is pulled, the second page may be continuously pulled, and the second page corresponds to data rows 10001 to 20000. Similarly, after the second page is pulled, column store file 2 may be obtained. In this way, pulling is performed in sequence, and a plurality of column store files are gradually generated for the target data table.

In addition, in step 100, after the column store is performed on the pulled data rows, corresponding primary key index entries may be constructed for the data rows and populated into the first primary key index. The primary key index in this embodiment is used for recording mapping relationships between primary keys and data positions, and each mapping relationship is recorded as a primary key index entry. In addition, two primary key indexes are provided in this embodiment. The first primary key index is first described herein, and the second primary key index is described below. The first primary key index in this embodiment is used for recording the primary keys of the data rows pulled in step 100 and the data positions thereof in the column store file. Continuing with the plurality of column store files generated for the target data table described above, an exemplary data position representation structure may be [an identifier of a column store file, a data position in the file]. FIG. 4 is a schematic diagram of an internal storage structure of a column store file according to an exemplary embodiment of the present disclosure. Referring to FIG. 4, the column store file includes a plurality of column segments (column segments), and one data position in the column store file may point to all elements (a plurality of black squares in FIG. 4) that belong to a same data row in the column segments. For example, for a data row whose primary key is name, if the data row is stored at data position pos1 in the column store file (identified as file1) shown in FIG. 4, the primary key index entry of the data row in the first primary key index can be recorded as: name—>>[file1, pos1].

In the research process, the inventors find that, in the pulling solution provided in this embodiment, the data rows stored in the column store manner have redundancy compared with the target data table when the pulling ends. This is because during the pulling, an insertion operation performed on the target data table may occur before pulling a corresponding data row. In this case, the inserted data row is pulled in step 100. In addition, during the pulling, a deletion operation performed on the target data table may occur after pulling a corresponding data row. In this case, the data row is pulled in step 100, but the data row is already deleted from the target data table. Correspondingly, the first primary key index generated when the pulling ends also has redundancy compared with the target data table.

For this redundancy problem, in this embodiment, it is proposed that no processing is

performed during the pulling, and correction is performed after the pulling ends.

As mentioned above, an objective of the column store method provided in this embodiment is to ensure data consistency between the column store and the row store. However, the column store file and the first primary key index that are generated during the pulling in step 100 cannot meet the data consistency yet. Therefore, the column store file and the first primary key index that are generated during the pulling will not be delivered for use; instead, they are first stored as background processing results, and will be delivered for use at a suitable subsequent moment. The delivery for use herein means being allowed to serve as a data basis for query and analysis operations in the foregoing OLAP scenario.

In view of this, referring to FIG. 2, it is proposed in this embodiment that the pulling operation in step 100 is performed by using an independent thread/process. For ease of description, this thread/process is described as a stock processing module in FIG. 2 and hereinafter. Referring to FIG. 2, the stock processing module can be independently executed, to gradually generate the column store files and the first primary key indexes that correspond to the target data table.

Still referring to FIG. 1, in step 101, the data operation performed on the target data table and monitored during the pulling may be synchronized, to generate the incremental file for storing the incremental data row and the second primary key index corresponding to the incremental file. As mentioned above, the online service provided by the row store is not affected in this embodiment, but it is proposed in this embodiment that after the column store is started, the data operation performed on the target data table during the pulling is monitored.

In an actual application solution, the operation log provided by the foregoing log node may be subscribed to. Based on this, after the column store is started, the log node may actively push the operation log to the execution subject of this embodiment when generating the operation log on the target data table. In this way, in this embodiment, the data operations performed on the target data table during the pulling can be monitored in a timely and comprehensive manner. It should be understood that these data operations are a cause that affects data consistency between the column store and the row store.

Referring to FIG. 2, it is proposed in this embodiment that these data operations are monitored and processed by using an independent thread/process. For ease of description, this thread/process is described as an increment processing module in FIG. 2 and hereinafter. It may be understood that, referring to FIG. 2, the increment processing module and the stock processing module that are independent of each other are proposed in this embodiment. In addition, referring to FIG. 2, a stage from a moment when the column store starts to a moment when the pulling ends is described as a first stage. In the first stage, the increment processing module and the stock processing module may be executed in parallel, and do not interfere with each other.

Processing logic of the increment processing module in the first stage is described below.

Referring to FIG. 1 and FIG. 2, in step 101, the increment processing module may synchronize the data operation performed on the target data table and monitored during the pulling. The synchronization herein may be understood as replicating, on the column store side, the data operation performed on the row store side. In the research process, the inventors find that the data operation performed on the target data table may point to any data row in the target data table. However, in the first stage, the stock processing module does not complete pulling of the data rows. Therefore, the increment processing module does not have a global view of the target data table. Consequently, some data operations performed on the target data table and monitored during the pulling cannot be directly replicated to the column store side.

Therefore, it is proposed in this embodiment that monitored data operations are classified into two types: an increment data operation and a non-increment data operation. The increment data operation may include a data operation that causes an incremental data row to appear in the target data table during the pulling and a data operation performed on the incremental data row. The incremental data row herein refers to a data row inserted into the target data table during the pulling. The non-increment data operation may include a data operation on a stock data row that already exists in the target data table when the column store is started.

Based on this, in step 101, when synchronizing the monitored increment data operation, the newly generated incremental data row may be stored into the incremental file. In this way, for the increment processing module, the incremental file generated by the increment processing module may include all incremental data rows that have appeared in the target data table during the pulling, so that the increment processing module has a global view of the incremental data rows. Therefore, the data operations for the incremental data rows can be replicated to the incremental file without an obstacle. In step 101, the second primary key index is proposed, to record a mapping relationship between a primary key of an incremental data row and a data position. Based on the second primary key index, it can be supported that the data positions in the incremental file can be accurately found for the increment data operations, thereby supporting replication of this type of data operations on the incremental file. It may be understood that, in the second primary key index, the data positions recorded for the incremental data rows are located in the incremental file proposed in this embodiment. It may be learned from the above that the data positions recorded for the data rows pulled from the target data table in the first primary key index are located in the column store file proposed in this embodiment. That is, in this embodiment, the data rows on the column store side are dispersedly stored in two types of files: the incremental file and the column store file.

In step 101, when synchronizing the monitored non-increment data operation, because an

operation object cannot be found in the incremental file and the second primary key index, such a data operation cannot be directly replicated to the incremental file. In step 101, it may be considered that synchronization of such a data operation fails. However, in the research process, the inventors find that impact of such a data operation on the data state of the target data table mainly relates to deletion of a data row. This is because insertion of a data row (that is, an incremental data row) into the target data table is accurately replicated in step 101 by using the incremental file and the second primary key index. Therefore, if accurate replication of the deletion of the data row involved on the target data table can be ensured in another manner, data consistency between the column store and the row store will not be affected when synchronization of such a data operation fails in step 101.

In view of this, referring to FIG. 1 and FIG. 2, it is proposed in this embodiment that in step 102, the to-be-deleted primary key may be recorded for the first primary key index based on the data operation monitored during the pulling.

Preferably, to ensure that deletion of data rows involved in the target data table during the pulling can be comprehensively replicated on the column store side (which may be specifically on the column store file and the first primary key index that are obtained when the pulling process ends) subsequently, primary keys to which deletion operations or update operations performed on the target data table and monitored during the pulling may be recorded as to-be-deleted primary keys in the first primary key index. In the research process, the inventors find that such a manner of comprehensively recording deletion may cause a primary key that is not included in the first primary key index to exist in the generated to-be-deleted primary keys (for example, an incremental data row is not pulled, and the primary key of this incremental data row is not included in the first primary key index, but if it is monitored that this incremental data row is deleted, the primary key of this incremental data row will also be recorded as the to-be-deleted primary key). Although this may lead to some invalid deletion operations, it can be effectively ensured that the deletion of the data rows on the target data table during the pulling is accurately and comprehensively replicated, thereby avoiding a problem of missing deletion on the column store side.

Referring to FIG. 2, in the first stage, through cooperation among steps 100 to 102 described above, data states between the column store and the row store can achieve consistency at the end of the first stage. In this case, the data state of the column store is jointly represented by the following processing results: the column store file and the first primary key index corresponding to the column store file, the incremental file and the corresponding primary key index, and the to-be-deleted primary keys recorded for the first primary key index. However, it should be understood that, at the end of the first stage, the data state jointly represented by these processing results is

consistent with the data state of the row store.

Based on this, referring to FIG. 1 and FIG. 2, this embodiment proceeds to a second stage. The second stage may be understood as a stage from the end of pulling to the delivery for use. In the second stage, the increment processing module and the stock processing module can still be executed in parallel, and do not interfere with each other.

Referring to FIG. 1, step 103 corresponds to the second stage. In step 103, in response to the end of the pulling, the first primary key index and the second primary key index may be combined into the target primary key index, and the target primary key index and the column store file may be corrected based on the to-be-deleted primary key. Referring to FIG. 2, the increment processing module may be responsible for performing two aspects of operations: combining the first primary key index and the second primary key index into the target primary key index, and correcting the target primary key index based on the to-be-deleted primary key. The stock processing module may be responsible for performing the operation of marking deletion for the column store file based on the to-be-deleted primary key.

In view of this, in step 103, the stock processing module may provide the generated first primary key index to the increment processing module in response to the end of the pulling. The increment processing module may combine, in response to the end of the pulling, the second primary key index generated by the increment processing module and the first primary key index provided by the stock processing module into the target primary key index. In actual application, the combination herein does not need copying and moving on memory, and instead, two types of primary key indexes can be uniformly maintained through a carrier of the list. The increment processing module may further correct the target primary key index based on the to-be-deleted primary key recorded for the first primary key index in step 101. The correction herein may include at least deleting, from the target primary key index, the primary key index entry corresponding to the to-be-deleted primary key.

Referring to FIG. 2, the corrected target primary key index may be used as one of delivery results after the second stage ends.

Still referring to FIG. 2, in step 103, the increment processing module may provide the to-be-deleted primary key recorded for the first primary key index in step 101 to the stock processing module, so as to trigger the stock processing module to correct the column store file based on the to-be-deleted primary key. In this embodiment, considering that data in the column store file is stored in the column store manner, directly deleting the data in the column store file has high costs. Therefore, preferably, the correction herein may be marking deletion for the column store file. An exemplary marking deletion solution may be as follows. A deletion bitmap (bitmap) is constructed for the column store file, and the corresponding data position of the to-be-deleted data row in the

column store file may be marked as deleted through the deletion bitmap. In this way, correctly maintaining the deletion bitmap for the column store file may represent a data row deletion state in the column store file, and actual data in the column store file does not need to be deleted. Accordingly, in step 103, the stock processing module may mark deletion for the column store file based on the to-be-deleted primary key.

Referring to FIG. 2, the corrected column store file may be used as one of delivery results after the second stage ends.

In addition, it should be noted that, in the second stage, the increment processing module still keeps synchronizing the monitored data operation performed on the target data table. That is, in the second stage, both the incremental file and the second primary key index part in the target primary key index dynamically change as the data operation is synchronized. This can ensure that the increment data operation on the target data table can be continuously replicated to the column store side. In this embodiment, a deletion bitmap (bitmap) may also be maintained for the incremental file. In this way, in a process of synchronizing the data operation, when the incremental data row needs to be deleted, this can also be achieved by marking deletion for the incremental file, and the actual data in the incremental file does not need to be deleted.

Referring to FIG. 2, the incremental file obtained at the end of the second stage may also be used as one of delivery results.

Accordingly, referring to FIG. 2, the data states between the column store and the row store are still consistent at the end of the second stage. In this case, the data state of the column store is jointly represented by the following delivery results: the corrected target primary key index (including two parts: the first primary key index and the second primary key index), the corrected column store file (including the deletion bitmap maintained for the corrected column store file), and the incremental file (including the deletion bitmap maintained for the incremental file) obtained at the end of the second stage. However, it should be understood that, at the end of the second stage, the data state jointly represented by these delivery results is consistent with the data state of the row store.

After the second stage ends, the foregoing delivery results of the column store in this embodiment may be delivered for use. The column store file and the incremental file may be used as a data basis for query and analysis operations in the foregoing OLAP scenario. Certainly, in actual application, to support the foregoing OLAP scenario, a column store index (Column Store Index, CSI) may be generated based on the column store file and the incremental file. The target primary key index provided in this embodiment may be used to support synchronization of the data operations on the row store side to the column store file and the incremental file, thereby maintaining data consistency between the column store and the row store after being delivered for

use.

Specifically, after the second stage ends, the increment processing module may continue to monitor the data operation performed on the target data table. If the data operation requiring insertion of the data row is monitored, the data row that needs to be inserted may be stored into the incremental file, and the corresponding primary key index entry may be added to the second primary key index included in the target primary key index. If the data operation requiring deletion of the data row is monitored, the corresponding primary key index entry is deleted from the target primary key index, and deletion is marked for the file (which may be the incremental file, or may be the column store file) in which the deleted data row is located.

Based on the above, in this embodiment, on the premise that normal provision of the online service based on the row store is maintained, the data rows can be pulled from the target data table in sequence to perform the column store in response to the triggering event, to generate the column store file and populate the first primary key index for the column store file based on the actually pulled data rows. In addition, the data operation performed on the target data table and monitored during the pulling can be further synchronized, to generate the incremental file for existing the incremental data row and the second primary key index corresponding to the incremental file. This can ensure consistency between column store and row store in terms of the increment. Furthermore, based on the data operation monitored during the pulling, the to-be-deleted primary key can be recorded for the first primary key index. In response to the end of the pulling, the first primary key index and the second primary key index are combined into the target primary key index, and the target primary key index and the column store file are corrected based on the to-be-deleted primary key. In this way, the data operation performed during the pulling on the stock aspect can be replicated to the target primary key index and the column store file through the recorded to-be-deleted primary key, to ensure consistency between the column store and the row store in terms of the stock.

In the foregoing embodiments or the following embodiments, in the first stage, the following processing solution may be used to ensure the correctness of the incremental file:

storing, if a data operation requiring insertion of a data row performed on the target data table is monitored during the pulling, the inserted data row as the incremental data row into the incremental file;

marking deletion for the incremental file based on a data operation requiring deletion of an incremental data row and monitored during the pulling.

The data operation requiring insertion of the data row may include the insertion operation (insert) and the update operation (update) mentioned above, etc. In addition, in actual application, the incremental file may be in a format such as CSV. When a new incremental data row appears,

an additional write operation may be performed at an end of the incremental file, to store the incremental data row into the incremental file.

The data operation requiring deletion of the incremental data row may include the deletion operation (delete) and the update operation (update) mentioned above, etc. As mentioned above, in this embodiment, the actual data in the incremental file does not need to be deleted. Instead, these data operations requiring deletion of the incremental data rows are replicated on the incremental file in a manner of marking deletion.

For the deletion operation (delete) and the update operation (update), an implementation solution of marking deletion for the incremental file during the pulling may be:

marking, if a deletion operation performed on the target data table for an incremental data row is monitored during the pulling, a data position corresponding to the to-be-deleted data row in the incremental file as deleted; marking, if an update operation performed on the target data table for an incremental data row is monitored during the pulling, a data position corresponding to a data row deleted due to the update operation in the incremental file as deleted. In addition, as mentioned above, the update operation (update) may be understood as first deleting a data row and then inserting a data row. When the update operation for the incremental data row is monitored, two aspects of operations may be performed. In one aspect, a data position corresponding to the data row deleted due to the update operation in the incremental file may be marked as deleted. In the other aspect, the data row inserted due to the update operation may be stored into the incremental file.

In this way, in the deletion bitmap (bitmap) maintained for the incremental file, the data position corresponding to the to-be-deleted incremental data row is marked as deleted.

As mentioned above, in the first stage, the second primary key index maintained for the incremental file also needs to be dynamically updated with synchronization of the data operation. For various data operations, the following processing solution may be used to ensure correctness of the second primary key index:

adding, if an insertion operation performed on the target data table is monitored during the pulling, a primary key index entry corresponding to an inserted incremental data row into the second primary key index;

deleting, if a deletion operation performed on the target data table for an incremental data row is monitored during the pulling, a primary key index entry corresponding to the to-be-deleted data row in the second primary key index;

updating, if an update operation performed on the target data table for an incremental data row is monitored during the pulling, a data position to which a primary key index entry corresponding to the updated data row in the second primary key index points to a data position

corresponding to a data row inserted due to the update operation in the incremental file.

It may be understood that in the foregoing processing solution for the incremental file, after the incremental data row is stored into the incremental file, the data position of the incremental data row in the incremental file and the primary key corresponding to the incremental data row can be determined for the incremental data row, so that the primary key index entry can be constructed for the incremental data row and populated into the second primary key index. In addition, as mentioned above, the update operation (update) may be understood as deleting a data row first and then inserting a data row. Consequently, the data position of the updated data row in the incremental file changes, but the primary key of the data row does not change. In this case, it suffices to update the data position to which the primary key of the updated data row is mapped in the second primary key index.

Based on the above, in this embodiment, for the increment data operations monitored in the first stage, a processing solution for the incremental file and the second primary key index is provided, thereby effectively ensuring correctness of the incremental file and the second primary key index. It should be understood that the processing solution provided in this embodiment is also applicable to the increment data operations monitored in the second stage, to ensure correctness of the incremental file and the second primary key index in the second stage. Details are not described herein again.

In the foregoing embodiments or the following embodiments, in the second stage, the following processing solution may be used to ensure proper correction on the target primary key index:

deleting, from a first primary key index part included in the target primary key index, a primary key index entry corresponding to the to-be-deleted primary key. This can ensure that in the second stage, the primary key existing in the target primary key index can be consistent with a primary key actually existing in the target data table.

In the research process, the inventors find that primary keys corresponding to some incremental data rows may be duplicated in the first primary key index part and the second primary key index part in the target primary key index. This is because, in the first stage, some incremental data rows on the target data table are pulled and stored into the column store file in the column store manner, primary keys corresponding to such incremental data rows appear in the first primary key index part, and as mentioned in the first stage, such incremental data rows also appear in the second primary key index part. However, in the first stage, if such incremental data rows are not deleted, it will result in duplicated primary keys appearing in the first primary key index part and the second primary key index part in the target primary key index. However, data positions to which such a duplicated primary key is mapped in the first primary key index part and the second

primary key index part are different because the incremental data row corresponding to such a duplicated primary key is simultaneously stored in the incremental file and the column store file.

It should be noted that such duplicated primary keys do not affect correctness of the column store, but occupy additional memory space. Therefore, it is further proposed in this embodiment that, in the second stage, a primary key index entry corresponding to a primary key duplicated between the first and second primary key index parts may further be deleted from the first primary key index part included in the target primary key index.

In this way, two aspects are involved in the correction of the target primary key index in the second stage.

1. The primary key index entry corresponding to the to-be-deleted primary key is deleted from the first primary key index part.

2. The primary key index entry corresponding to the primary key duplicated between the first and second primary key index parts is deleted from the first primary key index part included in the target primary key index.

By means of correction in these two aspects, it can be ensured that when the second stage ends, the target primary key index can be consistent with the primary keys in the target data table.

In addition, in the second stage, at least the following two exemplary processing solutions may be used to ensure proper correction of the column store file.

In the first exemplary processing solution, if it is obtained by query that a primary key corresponding to any data row in the column store file is the to-be-deleted primary key, a data position corresponding to the data row in the column store file is marked as deleted, to mark deletion for the column store file.

In actual application, data positions in the column store file may be traversed, to determine which data positions need to be marked as deleted. It should be understood that each data position in the column store file is used to store one data row, and the primary key corresponding to the data row is recorded. Therefore, in the traversing process, it may be determined whether a primary key recorded at a traversed data position can be obtained by query in the to-be-deleted primary keys. If the primary key recorded at the traversed data position can be obtained by query, the traversed data position is marked as deleted.

In addition, it should be noted that, after repairing on the target primary key index is completed in the second stage, the increment processing module can obtain a global view consistent with the actual data state of the target data table. Then, the increment processing module may synchronize, based on the target primary key index, the monitored data operation performed on the target data table into the incremental file or the column store file without an obstacle. Deletion marking performed on the column store file caused by synchronization of data operations

and deletion marking performed on the column store file caused based on the to-be-deleted primary keys may be performed in parallel. The two aspects do not interfere and conflict with each other.

In the second exemplary processing solution, if it is obtained by query that a primary key corresponding to any data row in the column store file is not the to-be-deleted primary key, the data row is rewritten into a new file; if a data operation performed on the target data table is monitored during the rewriting of the column store file, deletion for a generated new file is marked based on the monitored data operation in response to completion of the rewriting of the column store file.

FIG. 5 is a schematic diagram of an exemplary processing solution for correcting a column store file according to the present disclosure. Referring to FIG. 5, FIG. 5 shows the second exemplary processing solution.

In the research process, the inventors find that some column store files have many to-be-deleted data rows. These to-be-deleted data rows are actually useless data for the column store, but occupy a large amount of storage space. Therefore, in the second exemplary processing solution, it is proposed that the data rows that need to be reserved in the column store file are rewritten, to generate a new file. In this way, a plurality of column store files can be combined into the new file, and a large amount of storage space can be released. It may still be determined by traversal which data positions in the column store file have data rows to be deleted and which data positions have data rows to be reserved.

In the research process, the inventors further find that the rewrite operation takes a certain amount of time. Therefore, the data operation may be performed on the target data table during the rewriting. In the second exemplary processing solution, before the rewriting is completed, the original column store file does not become invalid, and the increment processing module synchronizes the non-increment data operation monitored during the rewriting to the column store file. As a result, during the rewriting, the column store file may change (in addition to the recorded to-be-deleted primary keys, more primary keys that need to be deleted may appear during the rewriting), but the new file does not sense the change.

Therefore, referring to FIG. 5, in the second exemplary processing solution, it is proposed to divide the second stage into two phases: a first phase and a second phase. The first phase is from completion of the pulling to the end of the rewriting of the column store file. The second phase is from the end of the rewriting of the column store file to delivery of the new file for use.

Referring to FIG. 5, in the first phase, the column store file remains valid, and the increment processing module may synchronize the monitored stock data operation to the column store file, and update the first primary key index part corresponding to the column store file in the target primary key index. In the first phase, in response to start of the rewrite operation, the stock

processing module may respectively record original data positions of the data rows in the column store file based on the target primary key index. It should be understood that in the first phase, the increment processing module may update, as a result of the monitored data operations, primary key index entries in the target primary key index that correspond to some data rows in the column store file. For example, in the first phase, if a data row at data position 1 in the column store file is deleted, the primary key index entry corresponding to the data row in the target primary key index is deleted. However, the original data position recorded by the stock processing module for the data row is the data position 1. For another example, in the first phase, if the data row at the foregoing data position 1 is updated, the data position recorded by the primary key index entry corresponding to the data row in the target primary key index already becomes data position 5 in the incremental file, but the original data position recorded by the stock processing module for the data row is the data position 1. It may be understood that in the two cases, the stock processing module will rewrite the data row to the new file, but the data row is actually deleted, and the data position corresponding to the data row should be marked as deleted in the new file.

Therefore, in the second phase, if it is obtained by query that a corresponding data position, in the target primary key index, of any rewritten data row in the column store file is different from an original data position corresponding to the data row or a corresponding primary key index entry of any rewritten data row in the column store file no longer exists in the target primary key index, the stock processing module may mark the data row in the new file as deleted in response to completion of the rewriting. In this way, correctness of the deletion bitmap (bitmap) corresponding to the new file delivered at the end of the second phase can be ensured.

In addition, considering that the data positions of the data rows will change due to the rewriting, referring to FIG. 4, in the second exemplary processing solution, the stock processing module may further separately record, in the first phase, the data positions in the new file for the data rows that are rewritten into the new file. Based on this, in the second phase, in response to the completion of the rewriting, the stock processing module may update the data positions, in the new file, of the remaining data rows that are not marked as deleted in the new file to the target primary key index. Thereafter, the new file can be delivered for use, and the column store file that has been rewritten is invalidated. In this way, it can be ensured that the first primary key index part in the target primary key index delivered when the second phase ends can correctly record the data positions corresponding to the related primary keys.

In the research process, the inventors find that in the second exemplary processing solution, the stock processing module also needs to perform a write operation on the target primary key index. To prevent a write conflict between the stock processing module and the increment processing module on the target primary key index, an exclusive write lock may be configured for

the target primary key index. Referring to FIG. 5, before the second phase is started, the increment processing module may hold the exclusive write lock. Then, in response to the completion of the rewriting, the stock processing module may obtain the exclusive write lock, to update the data positions, in the new file, of the remaining data rows that are not marked as deleted in the new file to the target primary key index. After the update operation is completed, the exclusive write lock is returned to the increment processing module.

It may be learned that in the second exemplary processing solution, the second stage is divided into two phases, so that correctness of the deletion bitmap (bitmap) corresponding to the new file delivered when the second stage ends can be effectively ensured, and correctness of the data positions corresponding to the related primary keys recorded in the first primary key index part in the target primary key index can be effectively ensured. In addition, a write conflict problem that may occur on the target primary key index can be effectively prevented through the exclusive write lock, thereby further ensuring the correctness of the target primary key index.

It should be noted that, some processes described in the foregoing embodiments and the accompanying drawings include a plurality of operations that appear in a specific sequence. However, it should be clearly understood that, these operations may not be performed in the sequence presented in this specification or be performed in parallel. Sequence numbers of the operations such as 101 and 102 are merely used to distinguish different operations, and the sequence numbers do not indicate any execution sequence. In addition, the processes may include more or fewer operations, and the operations may be performed in sequence or in parallel. It should be noted that, the descriptions such as “first” and “second” in this specification are used to distinguish different stages, primary key indexes, phases, or the like, and neither indicate a sequence nor indicate that “first” and “second” are different types.

FIG. 6 is a schematic diagram of a structure of an electronic device according to another exemplary embodiment of the present disclosure. As shown in FIG. 6, the electronic device includes a memory 60 and a processor 61.

The processor 61 is coupled to the memory 60, and is configured to execute a computer program in the memory 60 to:

pull, in response to a column store triggering event, data rows from a target data table in sequence to perform column store, to generate a column store file and a first primary key index corresponding to the column store file;

synchronize a data operation performed on the target data table and monitored during the pulling, to generate an incremental file for storing an incremental data row and a second primary key index corresponding to the incremental file;

record, based on the data operation monitored during the pulling, a to-be-deleted primary key

for the first primary key index;

combine, in response to an end of the pulling, the first primary key index and the second primary key index into a target primary key index, and correct the target primary key index and the column store file based on the to-be-deleted primary key.

In an optional embodiment, when synchronizing a data operation performed on the target data table and monitored during the pulling, to generate an incremental file, the processor 61 may be specifically configured to:

store, if a data operation requiring insertion of a data row and performed on the target data table is monitored during the pulling, the inserted data row as the incremental data row into the incremental file;

mark deletion for the incremental file based on a data operation requiring deletion of an incremental data row and monitored during the pulling.

In an optional embodiment, when marking deletion for the incremental file based on a data operation requiring deletion of an incremental data row and monitored during the pulling, the processor 61 may be specifically configured to:

mark, if a deletion operation performed on the target data table for an incremental data row is monitored during the pulling, a data position corresponding to the to-be-deleted data row in the incremental file as deleted;

mark, if an update operation performed on the target data table for an incremental data row is monitored during the pulling, a data position corresponding to a data row deleted due to the update operation in the incremental file as deleted.

In an optional embodiment, when synchronizing a data operation performed on the target data table and monitored during the pulling, to generate a second primary key index, the processor 61 may be specifically configured to:

add, if an insertion operation performed on the target data table is monitored during the pulling, a primary key index entry corresponding to an inserted incremental data row into the second primary key index;

delete, if a deletion operation performed on the target data table for an incremental data row is monitored during the pulling, a primary key index entry corresponding to the to-be-deleted data row in the second primary key index;

update, if an update operation performed on the target data table for an incremental data row is monitored during the pulling, a data position to which a primary key index entry corresponding to the updated data row in the second primary key index points to a data position corresponding to a data row inserted due to the update operation in the incremental file.

In an optional embodiment, when recording, based on the data operation monitored during

the pulling, a to-be-deleted primary key for the first primary key index, the processor 61 may be specifically configured to:

record a primary key to which a deletion operation or an update operation performed on the target data table and monitored during the pulling points as the to-be-deleted primary key.

In an optional embodiment, when correcting the target primary key index based on the to-be-deleted primary key, the processor 61 may be specifically configured to:

delete, from a first primary key index part included in the target primary key index, a primary key index entry corresponding to the to-be-deleted primary key.

In an optional embodiment, the processor 61 may be further configured to:

delete, from a first primary key index part included in the target primary key index, a primary key index entry corresponding to a duplicate primary key in a second primary key index part.

In an optional embodiment, when correcting the column store file based on the to-be-deleted primary key, the processor 61 may be specifically configured to:

mark, if it is obtained by query that a primary key corresponding to any data row in the column store file is the to-be-deleted primary key, a data position corresponding to the data row in the column store file as deleted, to mark deletion for the column store file.

In an optional embodiment, when correcting the column store file based on the to-be-deleted primary key, the processor 61 may be specifically configured to:

rewrite, if it is obtained by query that a primary key corresponding to any data row in the column store file is not the to-be-deleted primary key, the data row into a new file;

mark, if a data operation performed on the target data table is monitored during the rewriting of the column store file, deletion for a generated new file based on the monitored data operation in response to completion of the rewriting of the column store file.

In an optional embodiment, the processor 61 may be further configured to:

respectively record, in response to a start of a rewrite operation, original data positions of data rows in the column store file based on the target primary key index;

mark, if it is obtained by query that a corresponding data position, in the target primary key index, of any rewritten data row in the column store file is different from an original data position corresponding to the data row or a corresponding primary key index entry of any rewritten data row in the column store file no longer exists in the target primary key index, the data row as deleted in the new file in response to the completion of the rewriting;

update a data position, in the new file, of a remaining data row that is not marked as deleted in the new file to the target primary key index.

In an optional embodiment, an exclusive write lock may be configured for the target primary key index, and before the rewriting is completed, a process responsible for performing the rewrite

operation obtains the exclusive write lock after the rewriting is completed, to update the data position, in the new file, of the remaining data row that is not marked as deleted in the new file to the target primary key index.

In an optional embodiment, after correction performed on the target primary key index and the column store file is completed, the processor 61 may be further configured to:

continue to monitor a data operation performed on the target data table;

store, if a data operation requiring insertion of a data row is monitored, the to-be-inserted data row into the incremental file, and add a corresponding primary key index entry into the second primary key index included in the target primary key index;

delete, if a data operation requiring deletion of a data row is monitored, a corresponding primary key index entry from the target primary key index, and mark deletion for a file in which the deleted data row is located.

Further, as shown in FIG. 6, the electronic device further includes other components such as a communication component 62 and a power supply component 63. Although only some of the components are exemplified in FIG. 6, it does not mean that the electronic device includes only the components shown in FIG. 6.

It should be noted that, for technical details in the embodiments of the electronic device, reference can be made to the related descriptions in the foregoing method embodiments. To save space, details are not described herein again, but this should not cause any loss to a protection scope of the present disclosure.

Accordingly, an embodiment of the present disclosure further provides a computer-readable storage medium storing a computer program. The computer program, when executed, can implement the steps in the foregoing method embodiments.

Accordingly, an embodiment of the present disclosure further provides a computer program product, and a computer program included in the computer program product, when executed, can implement the steps in the foregoing method embodiments.

The foregoing memory in FIG. 6 is configured to store the computer program, and may be configured to store various other data to support an operation on a computing platform. Examples of the data include instructions of any application or method configured to be operated on the computing platform, contact data, phonebook data, messages, pictures, videos, and the like. The memory may be implemented by any type of volatile or non-volatile storage devices or a combination thereof, such as a static random access memory (SRAM), an electrically erasable programmable read-only memory (EEPROM), an erasable programmable read-only memory (EPROM), a programmable read-only memory (PROM), a read-only memory (ROM), a magnetic memory, a flash memory, a disk, or an optical disc.

The foregoing communication component in FIG. 6 is configured to facilitate wired or wireless communication between a device in which the communication component is located and another device. The device in which the communication component is located may access a wireless network based on a communication standard, such as Wi-Fi, a mobile communication network such as 2G, 3G, 4G/LTE, 5G, or a combination thereof. In an exemplary embodiment, the communication component receives a broadcast signal or broadcast-related information from an external broadcast management system through a broadcast channel. In an exemplary embodiment, the communication component further includes a near field communication (NFC) module, to facilitate short-range communication. For example, the NFC module may be implemented based on a radio frequency identification (RFID) technology, an infrared data association (IrDA) technology, an ultra-wideband (UWB) technology, a Bluetooth (BT) technology, and other technologies.

The foregoing power supply component in FIG. 6 provides power to various components of a device in which the power supply component is located. The power supply component may include a power supply management system, one or more power supplies, and other components associated with generation, management, and power distribution for the device in which the power supply component is located.

A person skilled in the art should understand that the embodiments of the present disclosure may be provided as a method, a system, or a computer program product. Therefore, the present disclosure may use a form of hardware-only embodiments, software-only embodiments, or embodiments combining software and hardware. In addition, the present disclosure may use a form of a computer program product implemented on one or more computer-usable storage media (including but not limited to a magnetic disk memory, a CD-ROM, an optical memory, and the like) that include computer-usable program code.

The present disclosure is described with reference to the flowcharts and/or block diagrams of the method, the device (system), and the computer program product in the embodiments of the present disclosure. It should be understood that computer program instructions may be used to implement each process and/or each block in the flowcharts and/or the block diagrams and a combination of processes and/or blocks in the flowcharts and/or the block diagrams. These computer program instructions may be provided for a processor of a general-purpose computer, a special-purpose computer, an embedded processor, or other programmable data processing devices to generate a machine, so that the instructions executed by the processor of the computer or the other programmable data processing devices generate an apparatus for implementing functions specified in one or more processes in the flowcharts and/or in one or more blocks in the block diagrams.

These computer program instructions may alternatively be stored in a computer-readable memory that can direct the computer or the other programmable data processing devices to operate in a specific manner, so that the instructions stored in the computer-readable memory generate a product that includes an instruction apparatus. The instruction apparatus implements functions specified in one or more processes in the flowcharts and/or in one or more blocks in the block diagrams.

These computer program instructions may alternatively be loaded onto the computer or the other programmable data processing devices, so that a series of operations and steps are performed on the computer or the other programmable devices, thereby generating computer-implemented processing. Therefore, the instructions executed on the computer or the other programmable devices provide operations for implementing functions specified in one or more processes in the flowcharts and/or in one or more blocks in the block diagrams.

It should be further noted that, the terms “include”, “comprise”, or any other variant thereof are intended to encompass non-exclusive inclusion, such that a process, a method, a commodity, or a device including a series of elements not only includes those elements, but also includes other elements not listed explicitly, or further includes elements inherent to the process, the method, the commodity, or the device. Without more limitations, an element defined by a statement “include a/an/one ...” does not exclude additional same elements existing in the process, the method, the commodity, or the device that includes the element.

It should be noted that, user information (including, but not limited to, user device information, user personal information, and the like) and data (including, but not limited to, data for analysis, stored data, displayed data, and the like) involved in the present disclosure are all information and data that are authorized by users or fully authorized by all parties. The collection, use, and processing of relevant data need to conform to relevant laws, regulations, and standards of relevant countries and regions, and provide corresponding operation entries for the users to choose to authorize or refuse.

The foregoing descriptions are merely embodiments of the present disclosure and are not intended to limit the present disclosure. For a person skilled in the art, various modifications and variations may be made to the present disclosure. Any modification, equivalent replacement, or improvement made within the spirit and principle of the present disclosure shall fall within the protection scope of the present disclosure.

CLAIMS

1. A column store method, comprising:

pulling, in response to a column store triggering event, data rows from a target data table in sequence to perform column store, to generate a column store file and a first primary key index corresponding to the column store file;

synchronizing a data operation performed on the target data table and monitored during the pulling, to generate an incremental file for storing an incremental data row and a second primary key index corresponding to the incremental file;

recording, based on the data operation monitored during the pulling, a to-be-deleted primary key for the first primary key index;

combining, in response to an end of the pulling, the first primary key index and the second primary key index into a target primary key index, and correcting the target primary key index and the column store file based on the to-be-deleted primary key.

2. The method according to claim 1, wherein the synchronizing a data operation performed on the target data table and monitored during the pulling, to generate an incremental file comprises:

storing, if a data operation requiring insertion of a data row and performed on the target data table is monitored during the pulling, the inserted data row as the incremental data row into the incremental file;

marking deletion for the incremental file based on a data operation requiring deletion of an incremental data row and monitored during the pulling.

3. The method according to claim 2, wherein the marking deletion for the incremental file based on a data operation requiring deletion of an incremental data row and monitored during the pulling comprises:

marking, if a deletion operation performed on the target data table for an incremental data row is monitored during the pulling, a data position corresponding to the to-be-deleted data row in the incremental file as deleted;

marking, if an update operation performed on the target data table for an incremental data row is monitored during the pulling, a data position corresponding to a data row deleted due to the update operation in the incremental file as deleted.

4. The method according to any one of claims 1 to 3, wherein the synchronizing a data operation performed on the target data table and monitored during the pulling, to generate a second primary key index comprises:

adding, if an insertion operation performed on the target data table is monitored during the pulling, a primary key index entry corresponding to an inserted incremental data row into the second primary key index;

deleting, if a deletion operation performed on the target data table for an incremental data row is monitored during the pulling, a primary key index entry corresponding to the to-be-deleted data row in the second primary key index;

updating, if an update operation performed on the target data table for an incremental data row is monitored during the pulling, a data position to which a primary key index entry corresponding to the updated data row in the second primary key index points to a data position corresponding to a data row inserted due to the update operation in the incremental file.

5. The method according to any one of claims 1 to 4, wherein the recording, based on the data operation monitored during the pulling, a to-be-deleted primary key for the first primary key index comprises:

recording a primary key to which a deletion operation or an update operation performed on the target data table and monitored during the pulling points as the to-be-deleted primary key.

6. The method according to any one of claims 1 to 5, wherein the correcting the target primary key index based on the to-be-deleted primary key comprises:

deleting, from a first primary key index part comprised in the target primary key index, a primary key index entry corresponding to the to-be-deleted primary key.

7. The method according to any one of claims 1 to 6, further comprising:

deleting, from a first primary key index part comprised in the target primary key index, a primary key index entry corresponding to a duplicate primary key in a second primary key index part.

8. The method according to any one of claims 1 to 7, wherein the correcting the column store file based on the to-be-deleted primary key comprises:

marking, if it is obtained by query that a primary key corresponding to any data row in the column store file is the to-be-deleted primary key, a data position corresponding to the data row in the column store file as deleted, to mark deletion for the column store file.

9. The method according to any one of claims 1 to 8, wherein the correcting the column store file based on the to-be-deleted primary key comprises:

rewriting, if it is obtained by query that a primary key corresponding to any data row in the column store file is not the to-be-deleted primary key, the data row into a new file;

marking, if a data operation performed on the target data table is monitored during the rewriting of the column store file, deletion for a generated new file based on the monitored data operation in response to completion of the rewriting of the column store file.

10. The method according to claim 9, further comprising:

respectively recording, in response to a start of a rewrite operation, original data positions of data rows in the column store file based on the target primary key index;

marking, if it is obtained by query that a corresponding data position, in the target primary key index, of any rewritten data row in the column store file is different from an original data position corresponding to the data row or a corresponding primary key index entry of any rewritten data row in the column store file no longer exists in the target primary key index, the data row as deleted in the new file in response to the completion of the rewriting;

updating a data position, in the new file, of a remaining data row that is not marked as deleted in the new file to the target primary key index.

11. The method according to claim 10, wherein an exclusive write lock is configured for the target primary key index, and before the rewriting is completed, a process responsible for performing the rewrite operation obtains the exclusive write lock after the rewriting is completed, to update the data position, in the new file, of the remaining data row that is not marked as deleted in the new file to the target primary key index.

12. The method according to any one of claims 1 to 11, wherein after correction performed on the target primary key index and the column store file is completed, the method further comprises:

continuing to monitor a data operation performed on the target data table;

storing, if a data operation requiring insertion of a data row is monitored, the to-be-inserted data row into the incremental file, and adding a corresponding primary key index entry into the second primary key index comprised in the target primary key index;

deleting, if a data operation requiring deletion of a data row is monitored, a corresponding primary key index entry from the target primary key index, and marking deletion for a file in which the deleted data row is located.

13. An electronic device, comprising a memory, a processor, and a communication component, wherein

the memory is configured to store one or more computer instructions;

the processor is coupled to the memory and the communication component, and is configured to execute the one or more computer instructions, to perform the column store method according to any one of claims 1 to 12.

14. A computer-readable storage medium storing a computer program, wherein the computer program, when executed by one or more processors, causes the one or more processors to perform the column store method according to any one of claims 1 to 12.

15. A computer program product, comprising a computer program, wherein the computer program, when executed by one or more processors, causes the one or more processors to perform the column store method according to any one of claims 1 to 12.