Zebra Users Guide

Table of contents

1 Column Security

NOTE: THIS FEATURE IS EXPERIMENTAL AND SUBJECT TO CHANGE IN THE FUTURE

Since Zebra supports reading and writing data in a column-oriented fashion, you can store secure and non-secure data in separate columns. Then, using the HDFS file system, you can enable access control by setting the appropriate permissions on the columns containing secure data.

About the data:

- All the files and directories containing secure data will have the same permissions and groups within a table.
- If no security information is provided, then the HDFS file system default behavior applies.

About the users:

- The user who creates the data will set the file permissions.
- If a permissions-related error happens, it will be communicated to the user as a normal IO exception.
- A user running a client application needs to have chgrp permissions to execute the "secure by group" operations on a table.
- If a user running a client application does not have read permissions for a secure column group, an IO exception is issued.

One simple Pig example:

```
a = load '/path_to_input_table' as (a:int, b:float,c:long,d:double);
b = store a into '/path_to_output_table' using org.apache.hadoop.zebra.pig.TableStorer('[a,
 b] secure by group:secure perm:640');
```

One simple MapReduce example:

```
zStorageHint = ZebraStorageHint.createZebraStorageHint(“[a, b] secure by group:secure
 perm:640”);
zSchema = …;
zSortInfo = …;
setStorageInfo(jobConf, zSchema, zStorageHint, zSortInfo);
```

2 Drop Column Groups

NOTE: THIS FEATURE IS EXPERIMENTAL AND SUBJECT TO CHANGE IN THE FUTURE

Zebra allows you to delete a column group (CG) using the column group name. For examples, see Drop Column Groups.

Please note the following:

- Any failures during a drop will leave the table in consistent state (either with or with out the column group). While success of a column group removal guarantees a column removal, a failure does not imply the column group is not removed. In rare cases, you might receive an error but the column could still be deleted.
- MapReduce jobs and other clients that are currently accessing the table might fail with exceptions. It is recommended that column groups be dropped when there are no accesses to a table. It might not be feasible to ensure that there are no readers for a table; in these cases the readers should handle the exception.
- Once a column group is dropped, the column group data is deleted from the underlying file system. In the case of the HDFS filesystem, it may not imply that physical data is actually removed because of earlier snapshot of the file system; handling this is out side the scope of Zebra.
- Concurrent column group deletions are supported and their access is serialized.
- Deleting a non-existant column group or a column group that is already deleted is not allowed.
- If you delete all the remaining columns in a table, it logically leaves an empty null table. The difference between a non-existant table and a table with zero columns is that opening a non-existant table causes an error.

3 Order-Preserving Sorted Table Union

With Zebra you can group all records from all "delta tables" on some sort key to form an output set of records while preserving the sorted ordering of the records in the original tables. For instance, if the client application wants to fetch records from a union of tables of T1, T2 on a column "C1", then all records from T1 with a particular value of column "c1" and all records from T2 with that value of column "C1" will be output. The ordering of the rows of the output set of the same value of column "C1" is undefined. As a prerequisite, both T1 and T2 must be sorted on column "C1". More specifically the input and results could be as follows:

Table T1:

```
C1    C2
-------------
A     11
A     12
B     21
B     22
D     41
```

Table T2:

```
C1    C2
-------------
A     101
A     102
C     301
D     401
```

T1 Sort-Unioned with T2:

```
source_table    C1     C2
------------------------------
0                A      11
1                A      101
0                A      12
1                A      102
0                B      21
0                B      22
1                C      301
1                D      401
0                D      41
```

Note that the sortness is guaranteed per mapper and among all mappers arranged with certain ordering, but not among mappers arranged in any ordering. For instance, the outputs generated by four mappers, m1, m2, m3 and m4, could be in total ordering between m1, m3, m2 and m4, but not in any other arrangements.

3.1 Indexing Sort-Unioned Results

The order-preserving sort-unioned results above can be further indexed by the component tables if the projection contains column(s) named "source_table". If so specified, the component table index is output at the position(s) as specified in the projection list. If the underlying table is not a union of sorted tables, the use of the special column name in a projection will cause an exception. If an attempt is made to create a table of a column named "source_table", an exception will be thrown as the name is reserved by zebra for the virtual name.

3.2 MapReduce Jobs

TableInputFormat has static method, requireSortedTable, that allows the caller to specify the behavior of a single sorted table or an order-preserving sorted table union as described above. The method ensures all tables in a union are sorted. For more information, see TableInputFormat.

One simple example: A order-preserving sorted union B. A and B are sorted tables.

```
```

```
...
TableInputFormat.setInputpaths("path_to_A, path_to_B");
TableInputFormat.requireSortedTable();
TableInputFormat.setProjection(conf, "f1, f2, source_table");
...
```

3.3 Pig Scripts

Pig takes an extra string argument of "sorted" indicating the desire to load from a sorted table or an order-preserving sorted table union. For more information, see Zebra Pig Examples.

One simple example:

```
...
T = load ('path_to_A, path_to_B') using TableLoader('f1, f2, source_table', 'sorted');
...
```

4 Merge Join and Sorted Tables

In data pipelines, there is often a need to join datasets. Zebra supports merge join on Zebra tables. For more information, see Merge Join.

One simple example:

```
class myMapper extends Mapper<…> {
  …
  Object keyGenerator;
  …
  public void map(…) {
     bytesKey = BasicTableOutputFormat.getSortKey(keyGenerator, userKey);
     …
     output.collect(bytesKey, valueTuple);
     …
  }
  public void configure(JobConf job) {
     keyGenerator = BasicTableOutputFormat.getSortKeyGenerator(job);
     …
  }
}
```