**How Random Sampling in Hive Works, And How to Use It**

A diagram of a sample

Description automatically generated

*Random sampling* is a technique in which each sample has an equal probability of being chosen. A sample chosen randomly is meant to be an unbiased representation of the total population.

In the big data world, we have an *enormous* total population: a population that can prove tricky to *truly* sample randomly. Thankfully, Hive has a few tools for realizing the dream of random sampling in the data lake.

And, with a little thought and effort, it can be done in a parallel, efficient way that leverages the strengths of parallel-processing, not punish its shortcomings. Let’s get started with **how not to accomplish true random sampling, then move into the best ways to do it effectively.**

**Brief Overview of Your Options**

Hive provides three key ways to randomly sample data:

* randomized selection, distribution, and sorting
* bucketized table sampling
* block sampling

This post won’t cover the latter two: bucketized table sampling or block sampling. We will instead cover what I consider to be the best *everyday*method for sampling, which leverages random selection, distribution and sorting.

Hopefully, we can cover block sampling and bucketized table sampling later!

**A Quick Look at How rand() Works in Hive**

rand()  
rand(INT seed)

rand()returns a random number (that changes from row to row) that is distributed uniformly from 0 to 1. Specifying the seed will make sure the generated random number sequence is deterministic.

The rand() function returns a number (double), from 0 to 1, that is randomly generated from row to row. You can even seed it if you wish. Now that we know how the random function in Hive works, let’s get going.

**Less Effective Ways to do Random Sampling**

There are a few ways to limit and randomize your data in Hive that are not recommended — either because they’re inefficient or unsuitable for the goal: true *random* sampling.

Here’s a way not to accomplish true random sampling:

select \* from my\_table  
limit 10000;

This query simply selects all data from your table and limits how much data returns. It’s never random (unless the files themselves are being randomized (unlikely)), and it’s only sampling so far as the first n (10,000) rows is considered a sample.

select \* from my\_table  
order by rand()  
limit 10000;

**ORDER BY** should be used with *caution*. Using it *guarantees* *total order* in the final output. This query will take the entire dataset, order it randomly by shuffling it to a single reducer (remember, total order), and will return you the first 10k lines. Not spectacular. Even though it gives you “random” data, it shuffles all the data to one place to do it. Nowhere near necessary for random sampling of data. Inversely, shuffling to one reducer *is* useful for normal sorting where final total order is important.

select \* from my\_table  
sort by rand()  
limit 10000;

**SORT BY** is a parallel-friendly way to do randomization of records, and is actually a fine way to do random sampling. *But it’s not the best option you have*. This query will take the entire dataset, and pass the data to multiple reducers (unless you have super tiny data, in which case random sampling is either simple or unnecessary). This means each reducer will sort its own data, but the final data will only be partially sorted at best (because no global sort is occurring — i.e. ORDER BY). Your output will then be limited to 10k lines. It’s also unclear how truly random your final output will be, because the reducers all will randomly sort their own rows, which could end up being similar in the final output.

**The Recommended Ways to do Random Sampling**

Let’s look at two methods that make for effective random sampling: **random distribution of records**, and **randomly selected proportions of records**.

select \* from my\_table  
distribute by rand()  
sort by rand()  
limit 10000;

We’re pretty close to the golden goose with this query. This will take the entire dataset, **distribute it** **randomly to different reducers** (meaning sending the data to reducers in a random fashion), and **sort it randomly on each reducer**. Normally, random distribution is a ***nightmare*** for Hive, because people want similarly distributed data (for joins and group bys)! But in our case, we don’t care about all that — we want some random data! So by telling Hive to distribute the data randomly to reducers, and sort it randomly on the reducers, we have a very high probability of truly randomized data when our limit kicks into play. It’s also pretty quick.

select \* from my\_table where rand() <= 0.0001 distribute by rand() sort by rand() limit 10000;

**Welcome to the golden goose of Hive random sampling**. Check out the **where clause**:

“ **where rand() <= 0.0001** “ takes the random number that is generated between 0 and 1 everytime a new record is scanned, and if it’s less than or equal to “0.0001”, it will be chosen for the reducer stages (the random distribution and random sorting).

That, friends, is map-side filtering (always good). If the total size of the table is known, you can randomly select some proportion of the data on a record-by-record basis. In this case, it’s .0001 of the total data. In short, it means that as the mappers are retrieving the data, they will generate a new random number, and if it’s less than or equal to the provided proportion, it will be chosen! If it’s greater, it’s ignored.

This is ***awesome***random sampling.

Again, this will do the same as the query above it, but in addition, some more randomness is added to the initial collection of the data. Before, we were selecting all the data, stirring it up a lot, and then taking 10k from it. Now, we’re more *randomly* selecting the rows in the first place, then stirring up a lot, then taking 10k from it.

Export hive table to excel

IN: [HIVE](https://bigdataprogrammers.com/category/hive/)

**Requirement**

You have one hive table named as infostore which is present in bdp schema. It is needed to get the data into Excel file.

**Solution.**

Let’s say the location where output file should present is

/root/local\_bdp/posts/export-hive-data-into-file

Step 1: Create Output directory

mkdir /root/local\_bdp/posts/export-hive-data-into-file/output

Step 2: Go to hive CLI and execute the following code.

set hive.cli.print.header=true;

INSERT OVERWRITE LOCAL DIRECTORY '/root/local\_bdp/posts/export-hive-data-into-file/output'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t'

*SELECT \* FROM bdp.infostore;*

set hive.cli.print.header=true;INSERT OVERWRITE LOCAL DIRECTORY '/user/hive/warehouse/output' ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' select \* from hive\_features where rand() <= 0.01 distribute by rand() sort by rand() limit 40;

Explanation of the query:

1. Insert overwrite is written so that if the file is already loaded then it will update the file in the second time run.
2. You have to define a local directory which was created in the first step. The output file will be available under the created directory.
3. Look at the line where fields terminated is written, there we have mentioned Tab  (\t) as we want columns to be separated by a tab delimiter.
4. We are taking all the data of infostore table so ‘\*’ is mentioned in the select query. You can also add filter condition if you wish to transfer certain records.

Step 3: Output

Verify the file which is generated in ‘/root/local\_bdp/posts/export-hive-data-into-file/output’ directory.

Go to the directory via WinSCP and see the content, it should be delimited by a tab \t .

If there are multiple part files you can combine all files using below command

cd /root/local\_bdp/posts/export-hive-data-into-file/output

cat \* > op.txt

and then copy content of op.txt and paste it into Excel. and before pasting into Excel make sure you are doing below step:

Select complete sheet and go to format cell and change the cell category to Text.

A screenshot of a computer

Description automatically generated

 https://bigdataprogrammers.com/export-hive-table-to-excel/