

Data transfer - MySQL to Infobright -

General: This document describes one way of transferring data from a MySQL database into an Infobright database using standard Linux tools and a CSV file as the exchange format.

Infobright is a column based database server and a possible replacement for a traditional datawarehouse. Actually it is "just" a different MySQL engine, but offering enhanced speed, because it uses a knowledge grid to store the data. It does not need indexes, nor does it need a specific star-schema. Have a look at their website at <http://infobright.org>.

As I work a lot with MySQL databases, it is just natural to look for an easy way of taking the data from a productive MySQL database and load it into an Infobright instance. Now there are differences between the two, so there is some effort necessary for transferring the data, but fortunately Linux offers strong text processing tools to cope with this task.

Working a lot with databases and data in general, I find that nothing tops the versatility and speed of *awk* (besides other tools). So over time I developed a couple of *awk* scripts to handle textual data that has previously been dumped from a databases. But we will look at this in detail a little bit later.

The first step in the process is often to simply dump the data of a given scope into a CSV file. Comma-separated files are actually very nice to work with: you can take them with you on your laptop if you need to work offline, you can use a simple editor to look at the data if you e.g. need an overview, you can extract some lines and much more, you can use it as an input to an ETL process or for debugging. You can also compress (zip) the data conveniently into smaller file.

There are of course several ways how to extract data from a source system. In this case we do it from the command line and using the *MySQL* client. We pipe the result of the SQL query through *sed* to get rid of tab characters and replace them with a semicolon (;).

Lets pretend we have a database called "flightsdb" in MySQL. It contains a table with data about all countries of the world. You are the "root" user and we want to output the result to a file called "countries.csv"

The data from the MySQL table would like like this (excerpt):

Name	Area	Code	Phone_Prefix
Nigeria	Africa	NG	+234
Algeria	Africa	DZ	+213
Spain	Europe	ES	+34
Netherlands	Europe	NL	+31
Belgium	Europe	BE	+32
Netherlands Antilles and Aruba	Central America & Caribbean	AN	+599
Greece	Europe	GR	+30
United States	North America	US	+1
Argentina	South America	AR	+54
France	Europe	FR	+33

Using the *mysql* client on the command line we extract the data from the database table, we replace the tabs and output the result to a file:

```
mysql -u root -p flightsdb -e "select * from country" | sed 's/\t;/g' > countries.csv
```

The result looks something like this:

```
Name;Area;Code;Phone_Prefix
Nigeria;Africa;NG;+234
Algeria;Africa;DZ;+213
Spain;Europe;ES;+34
Netherlands;Europe;NL;+31
Belgium;Europe;BE;+32
Netherlands Antilles and Aruba;Central America & Caribbean;AN;+599
Greece;Europe;GR;+30
United States;North America;US;+1
Argentina;South America;AR;+54
France;Europe;FR;+33
```

Note that the first row contains the names of the columns in the database (or as they have been output by the SQL query).

In some cases – where you don't have access to the source database yourself - this is all you get: a flat file with data. All the metadata about column types, length of columns etc, might not be available to you and getting this information might take some time. If you don't want to wait or are curious and want to get an overview of what the file is made of, you need to analyze the file yourself.

This example is an easy one of course – just by looking at the file in a text editor you would quickly understand its meaning. But if you have a huge CSV file with say 1 million rows and 25 columns, that is nothing you can easily work with.

Before we continue you will need to download a file from my website <http://datamelt.com>. It contains a couple of awk scripts we will use to to the analyses of the data. Download the file and unzip the *awk* scripts to a folder of your choice.

Now we do a couple of analyzing steps to better understand the data in the file. The header row of the file, of course, already gives us some indication of the content of the fields/columns. On the other hand sometimes a header row might not be present and you have to find out the meaning of the data in the columns all on your own.

First we do a check if all lines have the same number of columns. Otherwise the *awk* scripts might not display the correct results or we make invalid assumptions. So we run the following *awk* script by inserting the name of the script on the command line and passing the name of the output file we created previously as a parameter:

```
$ number_of_fields_min_max.awk countries.csv
number of records:      61
number of fields - high value: 4
number of fields - low value: 4
```

You can see from the output, that all rows in the file have 4 columns (and that we have a total of 61 lines). This is good – all rows have the same number of columns.

One point of interest is how many permutations (distinct values) are there within a specific column. We can use *awk* to find this out. On the command line run:

```
field_permutations.awk -v fieldnumber=2 countries.csv
```

This will give out a list of all permutations (distinct values) of the field – in this case of the second - which contains the name of the area in which the country is located.

Let's look at the output:

```
$ field_permutations.awk -v fieldnumber=2 countries.csv
evaluated field: Area (column=2)

[others]: 1
Africa: 14
Asia: 9
South America: 4
North America: 2
Central America & Caribbean: 4
Europe: 21
Europ: 1
Middle East: 4

number of permutations: 8
total lines (w/o header): 60
```

So we see that e.g. the area "Europe" occurs 21 times in the file. We can continue to apply the same script to the other columns by simply increasing the field number variable. Usually in this list you already find values that are e.g. "invalid" (at least we assume that using human sense) or duplicated (like "Europ") but written differently. For the computer the two values "Europe" and "Europ" are different, but we know - because we are smart - that they really are the same. This type of processing already gives a good indication of the quality of the data. The worst the quality is, the more you might want or have to dig in and see, if you can use the file at all for further processing.

Another interesting script is used to find out what is the minimum and the maximum length of the data of a specific field. This comes handy when you don't exactly know how the columns are defined in the source database. Let's have a look:

```
$ field_minmax_length.awk -v fieldnumber=2 countries.csv

evaluated field: Area
minimum length: 4 : Asia
maximum length: 27 : Central America & Caribbean
total lines (w/o header): 60
```

"Central America & Caribbean" is the longest value of column number 2 with 27 characters, so we can note this down and enter this as the maximum width of the column in the Infobright database a little bit later. The script above simply counts the number of characters of a specific column, regardless if it is a number or a string.

Another script - *field_highlow.awk* - allows you to determine the highest and the lowest value of a specific field. If the field contains text, this script returns the orthographically highest and lowest value, if the field contains numbers, it contains the highest and the lowest numeric value. This helps to get a feeling of the size or range of the data for a specific field.

Now we can repeat the evaluations for all fields in our file. If your file is very large - let's say above 100Mb - you usually have problems working with it in a text editor because of its size. The high speed of *awk* does a good job here to quickly give out results also for very large files - give it a try!

We are now at a point at which we know our data, have found out what types of data (numeric, text, etc) are used, we have found errors or unclear values which we can communicate to the responsible persons and have them corrected if possible. So we are in a good position to load the data into an Infobright database and use its fast query capabilities. Don't worry about indexes, because Infobright has no indexes! It gains its speed from enhanced logic and packaging of the data during load time.

Of course you need to install the Infobright database engine first. I would recommend to install it standalone (in parallel to an existing MySQL instance, but on a different port) or in a virtual machine for the first steps. Go to <http://infobright.org> and download the appropriate installation package and documentation.

Once you have Infobright running, login using the mysql client program and create a new database. We

simply name the database and the table, the same as in the original source database:

```
create database flightsdb;  
use flightsdb;
```

We can now go ahead and create a new table in Infobright, based on the evaluations we have done above. My results from the evaluation of the countries file are as follows:

- Field 1 (name of the country) is a field containing textual data of variable length. It's maximum length in the file is 30 characters.
- Field 2 (name of the area) is a field containing textual data of variable length. It's maximum length in the file is 27 characters.
- Field 3 (code of the country) is a field containing textual data of variable length. It's usual length is 2 characters, but there is a value "[undefined]" in some rows, so the maximum length is really 11 characters
- Field 4 (phone prefix) is a field containing actually numbers prefixed by a plus sign (+) but they are not really numbers that could e.g. be added up – they represent the prefix that has to be dialed when calling a specific country. So we treat it as textual data of variable length. It's usual length is 2 to 4 characters, but there is a value "[undefined]" in some rows, so the maximum length is really 11 characters.

With this in mind we design the table in Infobright. You should still be connected and see the prompt from the mysql client. Enter the following command:

```
create table country (name varchar(30), area varchar(27), code varchar(11), phone_prefix  
varchar(11));
```

You will receive a prompt indicating that the table was successfully created. The syntax of Infobright's SQL statements is usually the same as that of MySQL but it differs here and there.

Please note that the community edition does not allow you to delete records or update records; only inserts are possible. That means that for deleting or updating, you will have to drop the table, recreate it and reload all the data. The enterprise edition does not have this limitation.

The final step now is to load the data from the file we exported from the MySQL source database or that was given to us by somebody (meaning we have not produced it ourselves) into the newly created table. Unfortunately we still have a header row in the file, but we do not want it to be imported as well, so we remove it from the CSV file, using following command:

```
tail -n +2 countries.csv > countries_noheader.csv
```

So effectively we created a new file "countries_noheader.csv", but without the first row. Now we load the data of the file into the Infobright database table we previously created:

```
load data infile '/tmp/countries_noheader.csv' into table country fields terminated by ',';
```

The data will be loaded into the country table. As we have done some thorough checking of the data, we are optimistic that the load of the data will succeed. If not check the messages and your data to determine what is wrong; watch out also for warning messages that might occur during load.

At this point the data is available and usable for your queries. Infobright is very fast and offers a lot for analyzing your data. In the easiest case you check and analyze the data as shown above and then create the structure in Infobright and finally load the data which results in a datawarehouse-like fast-performing database.

This of course was a very simple example to get you going and to help understand what is required to transfer the data between the two systems. The larger the data is you load the more you will benefit from the performance of Infobright to analyze your data.

The zip file with the *awk* scripts you downloaded, contains a few other scripts that might come in handy while working with text files. They are all well documented – just look inside the scripts. Also, the scripts presented above usually have additional parameters you can pass to them, e.g. in the case when no header row is present in the text file.

Additionally, on my website at <http://datamelt.com> there are two other free tools available to the public: one is a **data generator**. It allows to generate mass data based on regular expressions, word lists or simply random. This way you can generate CSV files with millions of lines in a few minutes. The other tool is a **business rule engine** allowing you to check (mass) data in a file or database against rules you define. So you can quickly check if data adheres to your companies standards or logic. Both tools are written in Java and thus are portable.

Final words: Tools such as *awk*, *sed*, *head* or *cut* just to name some are flexible and fast-processing tools. They can be of a big help for processing or analysing and checking text efficiently. Text files are commonly used to exchange data but often they contain errors or are otherwise inconsistent. In this case *awk* scripts help you to quickly get an overview and prepare the data to load it into a column based database – Infobright – for fast analyses in a BI environment.

I hope this small tutorial is helpful for somebody and I would appreciate any feedback – positive and negativ – to uwe.geercken@web.de ! If you have additional questions, please don't hesitate to ask.
