mysqljsonexport 1.2

JSON file export for MySQL

mysqljsonexport 1.2
2013-01-04
Anders Karlsson
anders@papablues.com

Table of contents

Introduction

MySQL has come a long way and when this document is written, in May 2012, MySQL 5.6 is around the corner. At the same time, there is a lot of competition, the different NoSQL alternatives are breaking new ground where MySQL used to be king. In reality, what has opened up is not only a market with several alternatives that are much less common than what used to be the case (i.e. “Which SQL database should I choose”) but an environment where these alternatives coexist.

In addition to this, there is now also many more data formats, partly because of these new technologies (which many, me included, doesn’t really consider “new” in terms of not having existed before), and partly because we deal with new kinds of data, like GPS positioning data, document based data, lists of items, objects etc.

One of the many new formats of data that is appearing is JSON (JavaScript Object Notation). JSON has proved to be very popular not only with the JavaScript focused technologies, but as a generic format used in conjunction with REST and as a data transformation format.

In the case of MySQL, there isn’t really that much JSON support. Instead, MySQL seems to rely on CSV (Coma Separated Values) for data exchange. CSV is in some ways a neat format, but it is not standardized and has its roots in table/column/field only world. We now have objects, we have to understand that, and we have unstructured data. This fits easily in JSON and MySQL can sure handle it,

but there are few or any tools to support this. So along comes **mysqljsonimport** and **mysqljsonexport** (mysqljsonimport is not yet released as of this writing, instead there is **mysqljsonload**, which is like a beta version of mysqljsonimport). These are tools that allows you to load and export files JSON format to be loaded into and exported from MySQL. These are not advanced tools, but they should be advanced enough for most uses. The document is for **mysqljsonexport** which is the export tool for JSON data from MySQL. The two programs are pretty similar on many cases and are meant to be used together. The command line options are similarly named and the build process is similar. As in the case of **mysqljsonload**, the program supports using either a configuration file or the command line or a combination of the two. Unlike **mysqljsonload**, **mysqljsonexport** is not multithreaded, except that multiple tables are exported in parallel, but it does have another feature which is batching. When reading data from MySQL, all requested data is first read from the database and is materialized in memory on the client machine issuing the query (you may also materialize the result set on the server, and this is also supported by this program, but the result set still must be materialized before one can read a single row). This is an issue when exporting large result sets that might not fit in memory. To support this when exporting with **mysqljsonexport**, batching can be applied and this works with a combination of ordering and fetching using a specified key column (but as nearly all tables has a primary key, this is not an issues) where each batch is gotten starting where the previous batch left off, and each batch being limited in size by using the LIMIT clause.

Building mysqljsonexport

mysqljsonexport use the *autobuild* tools for building, so building it follows these steps:

- Download **mysqljsonexport** package from sourceforge: https://sourceforge.net/projects/mysqljson
- Unpack it
- Configure it
  - You need to point to the MySQL installation directory if this is not the default, you do this using the --with-mysql=<mysql directory>, for example *--with-mysql=/opt/local/mysql*
- Make it

Like this:

```
$ wget http://sourceforge.net/projects/mysqljson/files/mysqljsonexport/mysqljsonexport_
1.0/mysqljsonexport-1.0.tar.gz
$ tar xvfz mysqljsonexport-1.0.tar.gz
$ cd mysqljsonexport-1.0
$ ./configure –with-mysql=/usr/local/mysql
$ make
```

Optionally, you may run the test suite:

```
$ make check
```

Once the program is built, we are ready to configure it. You can use a normal MySQL configuration file, pass the options on a command line, or use a combination of the two. For normal MySQL connection options, like socket, port and username, these are also read from the *client* section

MySQL usual configuration files. See mysqljsonexport configuration options reference for a reference to all options.

Using mysqljsonexport

Sample mysqljsonexport session

The default mysqljsonexport configuration file is *mysqljson.cnf* and the options are in the *jsonexport* section in this file. The only required option is a database name. If you do not specify at least one table name or a SQL statement to execute, all tables in the given database are exported. Once you have the configuration file set up, you can start the program:

```
$ ./mysqljsonexport --database=test --table=mydata mydata.json
```

You can define if you want any additional columns exported beyond those in the table, and is this is a numeric column the value of that column may be incremented for each row.

The way the exporting works is that by default all columns in a table are exported, and the type of the column determines how it is exported. All numeric types are exported as JSON numbers with no quoting and all other types are quoted. Boolean columns may optionally be exported as JSON Boolean types, and NULL columns are exported as JSON null by default, all this is controlled by options on the command line or a configuration file.

Look at this example table:

```
CREATE TABLE users(id INTEGER NOT NULL PRIMARY KEY,
  firstname VARCHAR(256) NOT NULL,
  middlename VARCHAR(256),
  lastname VARCHAR(256) NOT NULL,
  active BOOL NOT NULL);
```

By default, this may be exported like this:

```
./mysqlexport --database=test --table=users
```

And the result will be something like this:

```
{"_id": 1, "firstname": "John", "middlename": "Frank", "lastname": "Doe", "active": 0}
{"_id": 2, "firstname": "Anne", "middlename": null, "lastname": "Doe", "active": 1}
```

To use a configuration file to get a more JSON style export, we create a config file, *users.cnf*, that looks like this:

```
[jsonexport]
database=test
table=users
tiny1-as-bool
skip-null
```

And the result will be this:

```
{"_id":1,"firstname":"John","middlename":"Frank","lastname":"Doe","active":false}
{"_id":2,"firstname":"Anne","lastname":"Doe","active":true}
```

Note that the Boolean column is now using true/false and that the middlename column is skipped when it is NULL.

An alternative file format is to have the export file being a single JSON array of objects, where each object represents a MySQL row. To use this format, use the --array-file option like this:

```
[jsonexport]
database=test
table=users
tiny1-as-bool
skip-null
array-file
```

And the output will the look like this:

```
[
{"id":1,"firstname":"John","middlename":"Frank","lastname":"Doe","active":false},
{"id":2,"firstname":"Anne","lastname":"Doe","active":true}
]
```

Batched export

MySQL will always gather a resultset of data and by default transport that resultset to the client (if you use the *--use-result* option, the resultset will be gathered on the server) before the first row is returned. This gathered resultset will be put in RAM, and this has the bad side effect that you need as much RAM as your largest resultset, so exporting a large table will be an issue. The solution is to export the table data in several batches and this can be done automatic with mysqljsonexport.

The way this works is that a small set is gathered by adding a LIMIT clause to the SQL statement and a WHERE clause is used to start the set, and also an ORDER BY statement is required. For this to work, there must be an appropriate column to use for batching, a column that is unique for every single row, is a single column and is not NULL. Usually there is a PRIMARY KEY column that can do this, and mysqljsonexport automatically find the appropriate primary key. Optionally you may specify a key column to use.

The way the automatic setup of batched export works, is that a single primary key column is found in each table to be exported, and if none is found for a table, then that particular table is exported in non-batch mode.

Batched export is initiated by setting the number of rows per batch to some non-zero value using the *--batch-size* option.

Types of export

There are three different ways to determine what to export:

- Database level export - If just a database and no specific table of an SQL statement to export is specified, then all tables in the given database are exported.
- Table level export - If a database and one or more tables are specified, either by just specifying them on the command line or using the --table option, then the specified tables are exported.

- SQL statement export - If a single SQL statement is given using the --sql option, then the result from this SQL statement is exported. Batching is supported, but for that to work, you have to tell mysqljsonexport where to insert the generated WHERE and ORDER BY clauses. You do this by inserting a %W (replaced with a "WHERE" for the first run and a "WHERE <generated condition> AND" for the following round or a %w (replaced with an empty string for the first round and "WHERE <generated condition>" for the following rounds) and a %O which is replaced by "ORDER BY <batching column>". The statement will also have an appropriate LIMIT clause added to it.

mysqljsonexport configuration options reference

The mysqljsonexport program takes many options, and is very flexible. The options may either be in a configuration file or passed on the command line or both, the options then prefixed with double dashes (--). The configuration file is passed using the *--defaults-file* argument, and if a configuration file called *mysqljson.cnf* is found, this one is also read. Some options have single character shortcuts, then they are prefixed with a single dash. When used in the configuration file, only the long option may be used.

Any unspecified options on the command line are treated as names of tables to export.

The *client* section in the normal MySQL configuration files is also read, to pick to defaults for the MySQL host, socket and such things. Options with a * may be specified more than once.

- array-file - Export as multiple object in a single JSON array in the output file.
- skip-auto-batch – Usually, the program figures out which columns to batch on by itself, and silently ignores tables where batching cannot be used. Using this option, this is not the case and a specific batching column must be specified
- batch-col - Name of the column to use for batching
- batch-size - Size of each batch in rows. If not specified, batching is not used
- skip-col - Do not export the specified column *
- col-incr - Increment per row of a fixed column, in the format column-name=increment, for example --col-incr=mycol=2 *
- col-json-name - name of the JSON export file for a specified column in the format column-name=json-name, for example --col-json-name=colname1=jsoncolname1 *
- col-value - Name and value of a column to export to the JSON file that is not part of the table to export in the format column-name=value, for example --col-value=mycol=57 *
- col-quoted - Name of column that will always be quoted *
- col-unquoted - Name of column that will never be quoted *
- database - Database to export from
- defaults-file - Configuration file to use
- directory - Directory to export into. The default is to use the database name
- dryrun - Do not export, just print what would be exported, table by table and column by column
- skip-empty - Do not export empty string values

- extension - File extension for export files, the default is .json
- file - File to export into
- help - Show help
- host - MySQL host to connect to
- include - Extra configuration file to read *
- limit - Max number of rows to export per table
- logfile - File to write log messages to
- loglevel - Level of logging, one of status, error, info, verbose and debug. The default is info
- skip-null - Ignore null values in the export file (the default is to use JSON null)
- skip-parallel - By default, tables are exported in parallel, but this can be turned off using this option
- password - The MySQL password to use
- port - The MySQL port to connect to
- socket - The MySQL socket to connect to
- skip-sql-no-cache - By default a SQL_NO_CACHE option is added to the generated SELECT statement, but this option will remove this
- sql-init - SQL statements to run before starting processing. These statements are run before a database is selected *
- sql-thread-init - SQL statements to run in each thread before processing starts *
- sql - SQL statement to export, instead of using a specific table
- sql-where-suffix - Extra condition to add to the WHERE statement that is used to export a table
- sql-finish - SQL statements to run after all other processing is done and before exiting *
- stats - Statistics to show after processing, use one of none, normal and full
- skip-stop-on-error - Do not stop processing if an error occurs.
- skip-stop-on-init-error - Do not stop if there is an error in a sql-init statement
- table - Name of table to export *
- tiny1-as-bool - Threat columns of the type tiny(1) as JSON Boolean.
- user - The MySQL username to connect with
- skip-utf8 - Do not enable UTF8 for MySQL connection
- use-result - Use the mysql_use_result for processing the data instead of mysql_store_result
- version - Print the program version

To do

There are several things to add in later versions. Among them are

- Parallel batching - While one batch is fetched, another is written to the export file.
- Base64 encoding of binary and BLOB columns
- Table level options.

- Printing of status on interrupt.
- Extending the documentation with more samples and better description of the different options.

Change log

Version 1.2

Added the array-file option to export as a single JSON array.

Version 1.1

Made options more like they do in the new *mysqljsonimport* tool

Fixed a few minor bugs.

Version 1.0

The first public release.