Sqoop2 CSV Intermediate representation

This wiki does not yet cover the complex type such as Array/Map/NestedArray representation that will be used inside one of the CSV implementations

See Intermediate Data Format API for 1.99.5 version

Intermediate representation

In sqoop 2 connectors will supply their own map phase that will import data into HDFS. Because this piece of code will be fully under connector maintenance, we need to agree on common intermediate (map output) format for all connectors and all cases. This page goal is to do comparison of different intermediate representation, so that we can pick up the appropriate one for sqoop 2.

Current solutions

List of ideas that we've explored. Rather than reinventing the wheel, this page contains summary of already present ways of data representation.

MySQL's mysqldump format

Comma separated list of values present in one single Text instance. Various data types are encoded as following:

Data type	Serialized as
BIT	String (array of bites rounded up to 1 byte, 20 bits are rounded to 24 bits/3 bytes)
INT(small, big, ...)	Direct value (666)
BOOL	Direct number (1 or 0)
DECIMAL(fixed, ...)	Direct value (66.6)
FLOAT (double, ...)	Direct value, might be in scientific notation (666.6, 5.5e-39). MySQL is not supporting NaN and +/- Inf.
DATE	String with format YYYY-MM-DD (2012-01-01)
DATETIME	String with format YYYY-MM-DD HH:MM:DD (2012-01-01 09:09:09)
TIMESTAMP	String with format YYYY-MM-DD HH:MM:DD (2012-01-01 09:09:09)
TIME	String with format HH:MM:SS (09:09:09)
CHAR(varchar, text, blob)	String
ENUM	String with enumerated value
SET	String with comma separated enumerated values

DATE and DATETIME types are returning same content as was stored in the table (no timezone conversions), whereas TIMESTAMP is always stored as UTC and is converted to connection timezone automatically. Explicit timezone specification do not seem to be part of the export.

Missing value is represented as constant NULL (it's not a string constant, therefore it's not quoted). Strings have very simple encoding -- most of the bytes (characters) are printed as they are with exception of following bytes:

Byte	Written as
0x00	\0
0x0A	\n

0x0D	\r
0x1A	\Z
0x22	\"
0x27	\'
0x5C	\ \ (no space)

For example:

```
0,'Hello world','Jarcec\'s notes',NULL,66.6,'2012-06-06 06:06:06'
```

PostgreSQL's pg_dump format

Similarly as in case of MySQL dump format, data would be represented as one Text instance where multiple colums would be separated by commas. Strings are quoted in single quotes (for example 'string'). All characters are printed as they are with exception of single quote that is doubled – e.g. two single quotes '' represents one single quote inside the string and not end of the string (for example 'Jarcec''s notes'). One quoted single quote is represented by four single qootes -- '''' represents just one ' (first one is opening, than there are two single quotes in a row that encodes one single quote inside the string and lastly the last single quote is closing the encoding). Null byte (0x00) is not allowed inside string constants. Binary constants are also quoted in single quotes, however entire field is converted to hexa with \x prefix – for example '\x4d7953514c' stands for string 'MySQL' (saved in binary column like bytea).

Data type	Serialized as
INT (and all variants)	Direct value (666)
NUMERIC	Direct value (66.60)
REAL(and all variants)	Direct value (66.5999985, 55e55) or string constant for special cases ('Infinity', '-Infinity', 'NaN')
VARCHAR(text, ...)	String
CHAR	String, unused positions at the end are filled with spaces
TIMESTAMP(date, time, ...)	String in format YYYY-MM-DD HH:MM:SS.ZZZZZZ (Date and hour part)
TIMESTAMP with time zone (and others)	String in format YYY-MM-DD HH:MM:SS.ZZZZZZ[+-]XX ('2012-07-03 14:07:11.876239+02')
BOOLEAN	Constants true and false (not quoted as a String)
ENUM	String
ARRAY	String that contains special structure - '{ITEM1, ITEM2, ITEM3}', ITEMX itself might be in separate quotes if needed.

Encoded examples:

```
666, 66.60, 'jarcec', 'Jarcec''s notes', '2012-07-03 14:07:11.876239', true, '{1,2,3}', NULL, '\x4d7953514c'
```

Microsoft SQL Server's bcp utility

SQL Server bcp utility is by default producing binary output that would be hard to interpret. Fortunately it can be forced to produce character output using command line switch "-c". In this case bcp utility will produce CSV file -- rowdelimiter can be specified by another command line switch "-t" (tab by default), similarly using "-r" switch user can specify row delimiter (new line by default).

Encoding overview of various data types follows:

Data type	Serialized as
INT (and it's variants)	Direct value (666)
NUMERIC (decimal, ...)	Direct value (666.66)
FLOAT(real, ...)	Direct value(33.299999999999997)
DATE	Constant in format YYYY-MM-DD (2012-01-01)
DATETIME	Constant in format YYYY-MM-DD HH:MM:DD.ZZZ (2012-06-06 01:01:01.000)

TIME	Constant in format HH:MM:DD.ZZZZZZ (01:01:01.0000000)
VARCHAR(nvarchar, char,...)	Directly as it without any encoding, empty string is represented as zero byte
NULL	Empty or missing value (for example when using comma as separator - ,,)

NOTE: String columns are not allowed to contain row or column delimiters, which is actually quite tricky.

Encoded example:

```
35,15.20,33.299999999999997,2012-06-06,2012-06-06 01:01:01.000,01:01:01.0000000,jarcec's comment with, comma
```

Utility bcp is also using so called format files where user can specify different column separator for each column. This functionality might be used to some extent to "emulate" escaping. For example knowing that all columns are string, user could specify separator for first column as a quote and for second (and others) as a quote, comma and an additional quote. Resulting in example row 'jarcec','arvind'. Unfortunately this technique is not powerful enough to solve a case when advanced multicharacter separator is still included in the data.

AVRO

We can also utilize AVRO for intermediate format. Avro do have support only for limited number of data types, so that we would have to encode some types (for example we might encode Date as a string with same format as above). List of all supported types can be found on following wiki page. Example of mapper code in the Connector itself would be:

```
GenericRecordBuilder builder = new GenericRecordBuilder(getSchemaSomehow());
builder.set("column1", value1);
builder.set("columnX", valueX);
context.write(new AvroKey(builder.build()), NullWritable.get());
```

Netezza

I did not find any special data exporting utility. Just recommendation to use nzsql and save output to file, that is generating tables in similar way as normal mysql client.

Example output:

```
  ID | TXT
 ----+------
   2 | cau
   1 | ahoj
(2 rows)
```

Teradata

Teradata have FastExport mechanism that is implemented by fexp utility. Default output formats are binary and a table like structure (show below). User might put together any arbitrary output format by adjusting select query. For example I've seen on many places recommendation to simply concatenate all columns together with proper column separator.

```
         id  description
-----------  ------------------------------------------------------------
          5  Cus
          5  Cus
          3  Hi
          1  Ahoj
          6  Bus
          4  Hello
          2  Nazdar
```

CSV Intermediate format representation proposal

I would like to make an proposal for suitable intermediate representation for Sqoop 2 based on my research of current solutions. I come to a conclusion that neither of the format is fully suitable. The most promising formats are mysqldump and pg_dump, however both are having issues - mysqldump is not supporting timezones and special constants for floating numbers (NaN, Infinity) whereas pg_dump is not escaping new line characters that might break any following hadoop processing. Therefore I would like to propose combination of both formats. Each row will be represented as a single line (no new line characters are allowed) where all columns will be present in CSV structure with comma as a column separator. Data types will be encoded as follows:

Note: the formats for Date related types has changed. See the comment below for more upto date information on the changes in the spec or refer to Intermediate Data Format API for the latest details as of 1.99.5

Data type	Serialized as
BIT	String (array of bites rounded up to 1 byte, 20 bits are rounded to 24 bits/3 bytes)
INT(small, big, ...)	Direct value (666)
BOOL	Direct number (1 or 0)
DECIMAL(fixed, ...)	Direct value (66.6)
FLOAT (double, ...)	Direct value, might be in scientific notation (666.6, 5.5e-39) and special sting constants 'Infinity', '-Infinity', 'NaN'
DATE	String with format YYYY-MM-DD[+/-XX] (2012-01-01)
DATETIME	String with format YYYY-MM-DD HH:MM:DD[.ZZZZZZ][+/-XX] (2012-01-01 09:09:09)
TIMESTAMP	String with format YYYY-MM-DD HH:MM:DD[.ZZZZZZ][+/-XX] (2012-01-01 09:09:09)
TIME	String with format HH:MM:DD[.ZZZZZZ][+/-XX] (09:09:09)
CHAR(varchar, text, ...)	String
BINARY(blob, ...)	String
ENUM	String with enumerated value
SET	String with comma separated enumerated values

Note: Date/time values have optional timezone specification in format +XX or -XX (number of hours that is different from UTC). This format was chosen based on pg_dump example. I'm not sure whether it's the best for zones with daylight saving shifts and for timezones that are not off by entire hour (like +11:30).

Note: Time values might have optional second fraction parts.

String will be encoded like in mysqldump case. Entire string will be enclosed in single quotes and all bytes will be printed as they are will exception of following bytes:

Byte	Encoded as
0x00	\0
0x0A	\n
0x0D	\r
0x1A	\Z
0x22	\"
0x27	\'
0x5C	\ \ (no space)

Fast export utility compatibility

We've also investigated compatibility between various fast exporting tools that are available out there.

mysqldump --compatible

Utility mysqldump do contain parameter --compatible that accept parameter "postgresql". However this parameter do not seem valid as using it results on nearly the same output format which is definitely not compatible with postgresql:

- Binary constants are very likely to fail
- Date time columns with timezone intell will get server default timezone