



Infobright Data Loading Guide

Revision 2.1 – November 11, 2010

Infobright includes a dedicated high-performance loader that differs from the standard MySQL Loader. The Infobright Loader is designed for speed, but supports less LOAD syntax than the MySQL Loader, and only supports variable length text formatted load files. IEE additionally supports the MySQL Loader, and the INSERT statement.

Default Loader

ICE only supports the Infobright Loader.

IEE defaults to the MySQL Loader, which has more robust error handling, but is not as fast as the Infobright Loader. For fastest LOAD results, use the Infobright Loader by setting the @bh_dataformat environment variable as described below:

To use the Infobright Loader with variable-length text in CSV format, enter:

```
mysql> set @bh_dataformat = 'txt_variable';
```

To use the Infobright Loader with binary data, enter the following command:

```
mysql> set @bh_dataformat = 'binary';
```

To return to the default MySQL Loader, set the data format to the standard MySQL format:

```
mysql> set @bh_dataformat = 'mysql';
```

Infobright Loader Syntax

Import your data into an Infobright table by using the following load syntax (all other MySQL Loader syntax is not supported):

```
LOAD DATA INFILE '/full_path/file_name'  
  
    INTO TABLE tbl_name  
    [FIELDS  
  
        [TERMINATED BY 'char']  
        [ENCLOSED BY 'char']  
        [ESCAPED BY 'char']  
  
    ];
```

The data is committed when the load completes if AUTOCOMMIT is set to on. This is default setting, but you can make it explicit by setting:

```
Set AUTOCOMMIT=1;
```

If you want to check the data via select before committing, then set AUTOCOMMIT to off:

```
Set AUTOCOMMIT=0;
```

New data can be seen by the loading session even though its not committed. If AUTOCOMMIT is off, you must complete the load using an explicit COMMIT (or it will roll back when the connection exits):

```
COMMIT;
```

FIELDS Clause

FIELDS subclauses are optional. If not specified the default values are used:

CLAUSE	DEFAULT VALUE
FIELDS TERMINATED BY	';' (semicolon)
FIELDS ENCLOSED BY	'"' (double quote)
FIELDS ESCAPED BY	'' (none)

Field delimiters must be single characters and not two (ie, not '//'). The Loader does not have a default value for the escape character, so if you are using one it needs to be stated explicitly.

MySQL full syntax is available on <http://dev.mysql.com/doc/refman/5.1/en/load-data.html>

FIELDS TERMINATED BY

The input file may be delimited using, for example, a semicolon, comma, pipe (|) or tab (\t) – it must be a single character (ie, not '//'). It is important that the character used as a delimiter does not appear in the actual data unless it is specifically escaped or enclosed (see details in FIELDS ESCAPED BY further on).

FIELDS ENCLOSED BY

The input file may have fields enclosed by a character as long as it is stated explicitly, otherwise the default enclosure of " is assumed. If there is no enclosure, then either ENCLOSED BY 'NULL' needs to be stated explicitly as the enclosure type, or alternatively, the ENCLOSED BY clause can be omitted. It is important that an enclosure character does not appear in the actual data unless it is specifically escaped (see details in FIELDS ESCAPED BY further on).

FIELDS ESCAPED BY - Case 1: Delimiters

If a character that is used as a delimiter appears in the actual data it must either be escaped or the entire field must be enclosed.

For example if we want to import a text field of [one, two or three] where the data fields are also terminated by ',' as in:

```
1,one,two or three,1234
```

then we can either use ESCAPED BY '\\' which requires adding the \ escape character to the data, or we can use ENCLOSED BY "" which requires the text field to be enclosed by ".

The input file and corresponding load statement:

```
1,one\, two or three,1234
```

```
LOAD DATA INFILE '/usr/tmp/file1.txt' INTO TABLE test_table1 FIELDS  
TERMINATED BY ',' ENCLOSED BY 'NULL' ESCAPED BY '\\';
```

is equivalent to an input file and corresponding load statement of:

```
1,"one, two or three",1234
```

```
LOAD DATA INFILE '/usr/tmp/file2.txt' INTO TABLE test_table1 FIELDS  
TERMINATED BY ',' ENCLOSED BY '"';
```

so these two files and load statements will result in:

```
mysql> select * from test_table1;  
+-----+-----+-----+  
| id  | textfield          | numerical |  
+-----+-----+-----+  
| 1   | one, two or three | 1234     |  
| 1   | one, two or three | 1234     |  
+-----+-----+-----+  
2 rows in set (0.00 sec)
```

FIELDS ESCAPED BY - Case 2: Enclosures

If a character used as an enclosure appears in the actual data, it must be escaped otherwise it will not load.

For example if we want to import the text field [one "and" two], then the input file required is:

```
1,"one \"and\" two",1234
```

and the corresponding load statement is:

```
LOAD DATA INFILE '/usr/tmp/file3.txt' INTO TABLE test_table1 FIELDS  
TERMINATED BY ',' ENCLOSED BY '"' ESCAPED BY '\\';
```

so this third file was read in as:

```
mysql> select * from test_table1;  
+-----+-----+-----+  
| id  | textfield          | numerical |  
+-----+-----+-----+  
| 1   | one, two or three | 1234     |  
| 1   | one, two or three | 1234     |  
| 1   | one "and" two     | 1234     |  
+-----+-----+-----+  
3 rows in set (0.00 sec)
```

Note: There is currently a bug being tracked in ICE where un-escaped embedded enclosures are handled in IB differently than in MySQL. If an even number of un-escaped embedded enclosures is included in the text, the data loads without requiring an escape character and is handled as expected as per MySQL. However, if there are an odd number of un-escaped embedded enclosures within the text then that row of data does not load and the load completes at the previous row; no error message is returned. With MySQL this case would load the data. This behaviour is being updated to reflect how MySQL handles this case.

ESCAPE CHARACTERS

If a text field includes escape characters, these must be escaped explicitly, otherwise the string is read in as straight text.

For the following input file:

```
2,other \t\t\ttext,4567
```

Using two different load commands (one with and one without an ESCAPED BY):

```
LOAD DATA INFILE '/usr/tmp/file4.txt' INTO TABLE test_table1 FIELDS  
TERMINATED BY ',' ENCLOSED BY 'NULL' ESCAPED BY '\\';
```

```
LOAD DATA INFILE '/usr/tmp/file4.txt' INTO TABLE test_table1 FIELDS  
TERMINATED BY ',' ENCLOSED BY 'NULL';
```

The data will be read in differently and produce different results respectively:

```
mysql> select * from test_table1;
+-----+-----+-----+
| id   | textfield          | numerical |
+-----+-----+-----+
|    2 | other              |          4567 |
|    2 | other \t\t\ttext  |          4567 |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

LINES TERMINATED Clause

It is important to note that IB loader ignores the LINES TERMINATED BY clause. Instead it detects how records are terminated based on the data in the input file.

There are two supported EOL formats:

1. Windows specific - '\r\n'
2. Unix specific - '\n'

About IEE Data Import Methods

The following methods are supported for loading data into IEE, listed from slowest to fastest:

- **Database INSERT** – Ensure that "autocommit=0" then do an explicit COMMIT at the end of your process. INSERT is not designed to load very large amounts of data, so for hundreds of thousands of rows or greater; it is recommended you use one of the Loader options.
- **MySQL Loader** – Loads from a flat file that includes text and field delimiters. Supports more features for escaping embedded characters, error handling, and transforming data using functions at the time of load then the Infobright Loader. This loader is set using an environment variable (@bh_dataformat='mysql').
 - For more information see <http://dev.mysql.com/doc/refman/5.1/en/load-data.html>
- **ETL Tools** – This uses an ETL tool to directly connect to your data source and load data directly to Infobright over a named pipe connection (data files not needed). When using IEE you can also load data in binary format, which is typically twice as fast as using text format. This also requires additional connectors available on the infobright.org contributed code page:
 - <http://www.infobright.org/Downloads/Contributed-Software/>
- **Infobright Loader, text data** – Loads from a flat file that includes text and field delimiters. The Infobright loader has reduced syntax support to optimize load speeds, and requires a clean load file. This loader is set using an environment variable (@bh_dataformat='txt_variable').

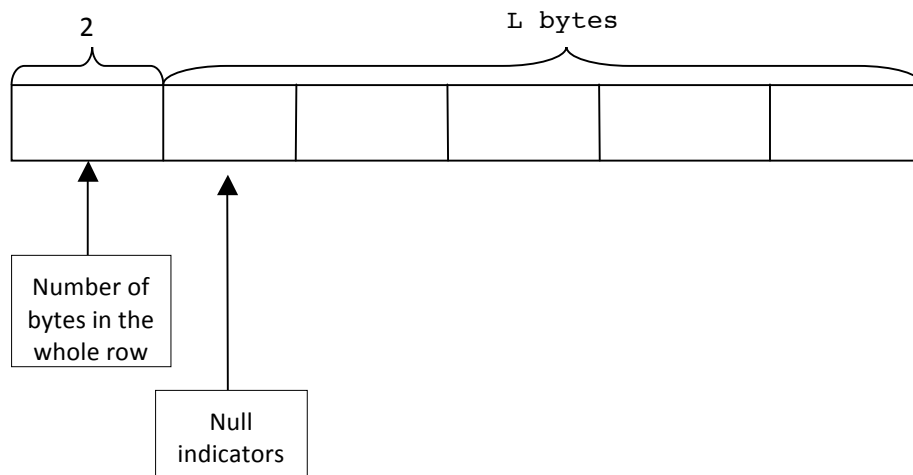
Note: When using text format, you must respect format conventions particularly for DATE, DATETIME and TIMESTAMP types. The only accepted format for DATE is YYYY-MM-DD. The only accepted format for DATETIME and TIMESTAMP is YYYY-MM-DD HH:mm:ss where HH represents hours on a 24-hour clock. In particular, the AM/PM modifier is not supported.

- **Infobright Loader, binary data** – Loads from a flat file in binary data format (see below for details), which is typically twice as fast as using text format. This loader and data format is set using an environment variable (@bh_dataformat='binary').

IEE Binary Format

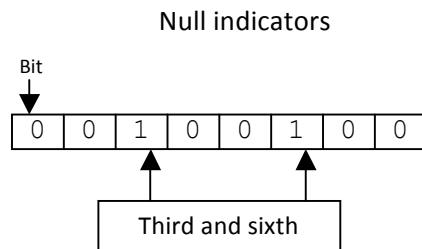
With Infobright's binary format load, individual rows are not separated by any special characters. There are also no values' delimiter or qualifier.

The following schema shows format of one row in the `BINARY` format.



Every row starts with `L` (2-byte integer) that specifies number of following bytes of data.

Null indicators are an array of bits – one bit per each column. 1 on *m-th* bit means that *m-th* value in the row is `NULL`.



The number of columns in a record determines the numbers of bytes in `NULL` indicators. For example, for a record that contains from one to eight columns indicator bits are stored on one byte. If a record contains from nine to 16 columns, two bytes are used and so on.

`NULL` indicators array is followed by `N` values where `N` is a number of columns in a row.

Formats and lengths in bytes for particular data types are showed in the following table.

Data type	Format	Length in bytes	
TINYINT		1	
SMALLINT		2	
MEDIUMINT		3	
INTEGER		4	
BIGINT		8	
FLOAT	IEEE 4-byte Float	4	
DOUBLE	IEEE 8-byte Double	8	
DECIMAL(N, M)	(Actual value) * 10 ^M	N	Length in bytes
		[1,2]	1
		[3,4]	2
		[5,9]	4
		[10,18]	8
TIME	[sign][h]hh:mm:ss	8-10	
YEAR	2-bytes integer	2	
DATE	4-bytes integer yyyyymmdd where yyyy = year - 1900	4	
TIMESTAMP/DATETIME	yyyy-mm-dd hh:mm:ss	19	
CHAR(N)	N characters	N	
VARCHAR(N)	2-byte integer of value L followed by L characters	2+L	
BINARY(N)	N bytes	N	
VARBINARY(N)	2-byte integer of value L followed by L bytes	2+L	

Note that CHAR is constant sized, whereas VARCHAR occupy only the size needed for actual value. Integer and floating-point data are stored as a “natural” binary representation of these values (little endian).