Loading Data

The Jethro's loader utility **JethroLoader** allows fast and efficient data loading from text files into a Jethro table. It parses its input into rows, and loads them into an existing Jethro table.
Note: In production environments where large data loads are running during user activity, we recommend to run the *JethroLoader* from a separate host, to minimize the impact on running queries.

Running Jethro Loader

The JethroLoader command line tool expects three parameters:
JethroLoader instance-name description-file
STDIN | input-location [input-location]...

- *Instance Name* – The name of the local instance to connect.
- *Description File* – A file describing the format of the input file and the way it will be processed.
- *Input location* – Either STDIN (to load from a pipe) or a list of one or one Each location is a path (local or on HDFS) to a file or directory to be loaded.

Files can be a mix of uncompressed files and compressed files (.gz or .gzip extension) – compressed files are automatically uncompressed in-memory when they are read.
Examples:

- **JethroLoader dev1 t1.desc t1_input_dir**
- **JethroLoader dev1 t1.desc t1.part1.csv part2.csv**
- **JethroLoader dev1 t1.desc HDFS://home/jethro/files/t1.gz**
- **sed ... | JethroLoader dev1 t1.desc stdin**

When loading from a pipe, the program that sends data to *JethroLoader* may fail in the middle. In that case, the loader is not be notified of error - it gets a standard end-of-file (EOF) notification for its input, and therefore will commit the data and exit without error. It is your responsibility to monitor for such cases and respond accordingly.

The loader report log is placed at: ***/var/log/jethro/{instance-name}/loader.*** The loader report file name includes the target table name of each load.

Loader Description File Format

The loader description file describes the structure of the input file and the way it will be processed. It has three sections:

1. **Table-level section** – Describes the format of the input file and several processing options.
2. **Column-level section** – A mapping between the fields in the file and columns in the table.
3. *(Optional)* **Record description section** – special clause for handling variable format files – files with different format per line (discussed in section "Input Files with Variable Format" on page).

To get started, you can copy and modify the sample description file that is located at: **$JETHRO_HOME/doc/sample_loader_desc_file.desc.**
The sample description file also include a summary of the syntax, commented out.

Table-Level Description Section

If using all the default options, the minimal table-level section is just the table name to be loaded into – like *TABLE my_table.* You can optionally specify non-default options like the file format and processing options.
The general syntax of the table level section is: *(the full version is at end of the chapter)*

```
TABLE [schema_name.]table_name
        [APPEND | OVERWRITE] [PARTITION partition_list]
        [ROW FORMAT DELIMITED
                 [FIELDS [TERMINATED BY char]
                                 [QUOTED BY char | NONE]
                                 [ESCAPED BY char]
                 ]
                 [NULL DEFINED AS string]
        ]
        [OPTIONS [SKIP n] [ROW REJECT LIMIT n]]
APPEND / OVERWRITE and the optional PARTITION limit
```

The default behavior of **JethroLoader** is **APPEND**- inserting the input file data into the existing table. You can also optionally limit the load to just a specific set of partitions – the rest of the records will be skipped. Alternatively, you can specify **OVERWRITE** to replace either an entire table or just a specific set partitions. This allows updating or deleting rows from a table, in cases where dropping a partition is not fine-grained enough.

Overwriting (like appending) is an atomic operation – until the operation is done, queries will see the older version, and once it is done, the queries will see the new version. So, you can safely refresh the contents of a dimension table while users are running queries, for example, without the need to drop the dimension table, create it again and reload it. This avoids the risk of some queries failing (if the dimension table does not exist momentarily) or returning zero results (if joining to the dimension table while it is still empty, during **JethroLoader** run).

Overwriting specific partitions is useful for a data correction process – for example, replacing a daily partition with a corrected data. Limiting the loader to a specific set of partitions allows smooth replacement of specific partitions with new data. Input records outside the list of partitions will be automatically skipped.

To limit the loader to a specific set of partitions, provide a comma-separated list of partition values, each quoted by single quotes, for example:

```
TABLE my_partitioned_table
           OVERWRITE PARTITION '2013-06-01', '2013-06-02', '2013-06-03'
…
```

Element	Explanation
ROW FORMAT DELIMITED	a clause describing a non-default file format.
FIELDS	specifies non-default field definition; how to break an input line into fields.
TERMINATED BY char	specifies an alternative field. The default field separator is tab (\t). For example: • TERMINATED BY '|' • TERMINATED BY '\001' uses ASCII 001 (^A), used by Hive TEXTFILE by
QUOTED BY char | NONE	specifies a quote character. Default is none - surrounding quotes are considered part of a string.
ESCAPED BY char	specifies an escape character (usually the backslash \ character) for quoted strings (must also specify a quote character). Needs to be explicitly specified. Once specifies, it allows: • Embedding the quote character and escape characters inside a string, for example \"\' • Embedding special characters - \t for tab, \n for newline, \r for carriage return.
NULL DEFINED AS string	specifies an alternative NULL string, in addition to the default empty string. For example, Hive and Impala CLIs write NULLs to their output as the string "NULL"
OPTIONS	A clause describing non-default processing options. • **SKIP** *n* instruct the JethroLoader to skip *n* header lines (default is 0 no skipping). • **ROW REJECT LIMIT** *n* allows up to *n* invalid records in the input file before aborting the load. The The default row reject limit is 100. Rejected rows are stored in a file called */loader_rejects_date_time_pid.out* in the local directory, each with the reject reason.

⚠ End of line character is recongnized automatically by JethroLoader. The Supported characters are CR, LF or CR/LF.

Column-Level Description Section

The basic column description section maps the file fields to table columns. For each field in the file, it mentions the target column name, by input file position.
For example: ***TABLE my_table (c1,c9,c5)***

In the above example, the first field of each input record will be mapped to c1 column, the second field will be mapped to c9 column, and the third field will be mapped to c5 column.

- If a specific record has additional fields, they will be ignored. *In the above example – if a specific record has five fields, the last two will be ignored.*
- If a specific record has fewer fields than mentioned in the description file, fields with missing value will have NULL value. *In the above example , if a specific record has only one field, then columns c9,c5 will have NULL*
- Any table columns that are not mentioned in the description will have NULL value. *In the above example, if table MY_TABLE also has a column named c4, it will have NULLs.*

Column NULL Definition
A specific per-column NULL identifier can be declared, adding NULL defined as "string" following the specific column"s name.
This comes as addition to the global NULL definition that can be declared in the table-level description. for example:

```
c1 null defined as 'c1-null'
c2
c3 null defined as 'c3-null'
```

In the above example, where **c1-null** *appears in c1 and* **c3-null** *appears in c3, NULLs will be written.*

Skipping Input File Fields

An input file may include fields that are you may wish not to load. You can ask the loader to skip them by specifying skip_columns in the description file. The skipped fields are not mapped to any column. For example:

```
TABLE my_table
    OPTIONS SKIP 2
          (c1,
          skip_columns(3),
          c9)
```

In the above example, the first field of each input record will be mapped to **c1** column and the fifth field will be mapped to c9 column – skipping three fields from each input record (fields 2,3,4).
In addition, the first two records (lines) in the input file will also be skipped **(OPTIONS SKIP 2)**.

Assigning Constants, Internal Variables and Functions

Instead of assigning a column the corresponding field from the input file, you can assign it a constant, an internal loader variable, or the output of a few function calls.
In that case, the corresponding field from the input file will be skipped.
Note: If you do not want to skip any the input file fields, put all the special column assignments at the end of the description file column list.
The **internal loader variables** can help you track the lineage of each row where did it come from:

- **$LOAD_START** - When did this JethroLoader run start (returns TIMESTAMP).
- **$FILE_NAME** - The name of the current file that is being loaded (returns STRING).
- **$FILE_PATH** - The full path of the current file that is being loaded (returns STRING).
- **$LINE_NUMBER** - The line number in the current input file (returns BIGINT).

In addition, there are several **built-in functions** you can use:

- **getenv('env_variable_name')** - Gets an environment variable value.
- **substr(another_column, start_pos[,length])** - Returns a substring of another column from character position **pos**, either until its end or only **length** characters.
- **extract(another_column,'timestamp_element')** - Extracts a element from a timestamp column, such as day of month. Valid elements are **'year', 'quarter', 'month', 'week', 'day', 'hour', 'minute', 'second'** and **'microsecond'**.
- **url_extract_domain(another_column)** - Extracts domain name from URL.
- **url_extract_base_url(another_column)**- Extracts base URL(removes parameters and anchor from URL).

For example:

```
TABLE my_table
( product_code,
  product_code_type =   substr(product_code,1,4),
  load_start_ts     =   $LOAD_START,
  load_full_path    =   $FILE_PATH,
  load_line_number  =   $LINE_NUMBER,
  source_region     =   'us-east-1',
  loader_home_dir   = getenv('JETHRO_HOME')
)
```

Note: In the above example, only **product_code** column is read from the input file. The rest are either constants **(load_start_ts, load_full_path, source_region, load_home_dir)** or computed per row **(product_code_type, load_line_number)**.

INTEGER/BIGINT Formats

Numeric values loaded into INTEGER or BIGINT column can be formatted as a decimal number or as hexadecimal number using the'0x' prefix. The format is automatically identified by the Loader and converted to INTEGER or BIGINT. here are so,e examples for numeric input for loader and their converted INTEGER/BIGINT values:

Input Value	Value in INTEGER column	Value in BIGINT column
1234	1234	1234
-1234	-1234	-1234
0x4D2	1234	1234
0xFFFFFB2E	-1234	4294966062
0xFFFFFFFFFFFFFB2E	Out of range	-1234

TIMESTAMP Formats and Timezone

The default timestamp format is **'yyyy-MM-dd HH:mm:ss'** (without sub-second component). However, you can specify a FORMAT clause to specify an alternative format per column.
For example:

```
TABLE my_table
( event_id,
  event_ts         format='dd-MM-yyyy HH:mm:ss.SSS'
  processing_date  format='yyyy-MM-dd'
```

If the input string of the field is longer than the format string, the rest of field content is ignored. For example: for **format='yyyy/MM/dd'**, the input field **'2014-02-14 15:16:17'** will be stored as the timestamp **'2014-02-14 00:00:00'**. This allows truncating the input record to a lower precision – in the above example, from a second-level to a day-level.
Timestamp format strings are **case sensitive**. The valid format elements are:

Format Element	Meaning
yyyy	4-digit year (1970-2038)
M	1 or 2 digit month (1-12)
MM	2-digit month (01-12)
MMM	3-character month (Jan-Dec)
d	1 or 2 digit day (1-31)
dd	2-digit day (01-31)
H	1 or 2 digit hour (0-23)
HH	2-digit hour (00-23)
m	1 or 2 digit minute (0-59)
mm	2-digit minute (00-59)
s	1 or 2 digit second (0-59)
ss	2-digit second (00-59)
SSS...	1-6 -digit sub-second element For example: S 1 digit, SSS 3 digits, SSSSSS 6 digits
unix_timestamp	The number of seconds since 1/1/1970 (microseconds/milliseconds optional after the decimal point). The format cannot be mixed with other format elements.

In addition, the **JethroLoader** by default does not perform any **timezone** manipulation – the input is loaded as is. In other words, we store all TIMESTAMP data in UTC timezone and we assume all input is already in UTC. However, in some cases you may want to adjust the timezone of an input field. For example, you may want to load log files from different data centers – each file has timestamps in the local timezone, so each needs its own adjustment.

You can ask for a timezone adjustment of a specific field by using the TIMEZONE keyword for timestamp field. In that case, the **JethroLoader** will compute the current offset of that timezone vs. UTC once (at the beginning of the run) and apply it to all values of that field. The offset is specified at the field-level – different fields may have different offsets.

For example:
...event_ts timezone='America/New_York',
processed_ts format='unix_timestamp' timezone='America/Los_Angeles',
...

Input Files with Variable Format

In some cases, different records (lines) in the same file have different format. For example, a file can include several event types or log message formats, each with a different number and order of fields.
JethroLoader description file supports these type of input files in the following manner:

1. You can define several, different record description blocks different mapping of the source record to table columns, you must provide each of them a different alias (record_desc_alias).
2. If you defined more than one record description block, you must also add a record description chooser block to the description file a list of rules for choosing the correct record format per

The record description chooser syntax allows defining a list of rules on the source data. Each rule has a condition plus a record description alias.
For example: ***@if (int($1) == 2) type2***

The rule above says find the first field in each record, cast it to integer, check if that integer equals to two, and if so, process this record according to record description called type2 (see an example below).
The rules are processed one-by-one, by their order in the description file. Once a rule is matched, the record is processed according to the record description alias and the JethroLoader moves to the next record. If none of the rules match a specific record, it is skipped.

For example, assume an input file that holds multiple event types. The first field of each line holds the event type. In the example, we chose not to load the event type into a column, just use it for record format decision (though you could definitely also map it to a column in the record description blocks):

```
TABLE events
OPTIONS
ROW REJECT LIMIT 10000
(skip_columns(1),event_id,event_time,url) type1
(skip_columns(1),event_id,event_time,user_id,url) type2
(skip_columns(1),event_id,event_time,host,path,filesize) type3
@if (int($1) == 1) type1
@if (int($1) == 2) type2
@if (int($1) >= 3) AND int($1) <= 7) type3
```

CSV Output Mode

Sometimes it is useful to export a query result set as a CSV file. For example, you may want to store the query result or load it into another database (or even back to Jethro).
The CSV output mode (-c option) allows simple export to CSV. It writes the query result set into a compact form (without wrapping each column with extra whitespace). The CSV output mode also suppresses all informational message (quiet mode), thereby avoiding any extra lines to at the output's beginning or end, lets you control the column separator character (default is comma), and allows you to optionally remove the header line.
JethroClient Command Line Options

- **-i FILE**: Executes an SQL script and exits. This command runs all SQL queries from a local FILE (each query should be terminated with a semicolon).
- **-o FILE** : Writes the output to FILE.
- **-q TEXT** : Executes the TEXT query and exits.
- **--no_header** : Does not print the header lines for queries (the column names' lines).
- **--quiet** : Does not print informational messages.
- -c : Switches to CSV output mode
- **-d 'char'** : For CSV mode, changes the column delimiter from comma to **char**. You can also use ASCII annotation such as **\001** or use **\t** for tab delimiter.

Diagram of the Description File Syntax

The following sections display the contents of the description file.
Description File Format
table_level_desc
record_desc record_desc]...
[record_desc_chooser]...
table_level_desc
TABLE [schema_name.]table_name
[APPEND | OVERWRITE] [PARTITION partition_list]
[ROW FORMAT DELIMITED
[FIELDS [TERMINATED BY char]
[QUOTED BY char | NONE] [ESCAPED BY char]
]
[NULL DEFINED AS string]
]
[OPTIONS [SKIP n] [ROW REJECT LIMIT n]
]

record_desc
(column_desc [,column_desc]...) [record_desc_alias]

column_desc
column_name |
column_name [NULL DEFINED AS 'string'] |
column_name [FORMAT='format_string'] [TIMEZONE='region/location'] |
column_name = const | builtin_var | builtin_function |
skip_columns

builtin_var
$LOAD_START | $FILE_NAME | $FILE_PATH | $LINE_NUMBER

builtin_function
extract(another_column,'timestamp_element') |
getenv('env_variable_name') |
substr(another_column, start_pos [,length])

record_desc_chooser
@if(condition) record_desc_alias;
condition
simple_condition |
condition) |
condition AND | OR condition |
NOT condition
simple_condition
data_type($col_position) = | != | <> | > | >= | < | <= constant

Scheduled Loads

Jethro allows scheduling of automatic loads, based on pre-defined time cycles. Input files are pulled periodically from a source folder called *drop folder*, and loaded into table.
Note: If several loads were scheduled, the loads are processed consecutively (one after another) and not in parallel.
Scheduled loads are managed by loads scheduler service. The service initiates periodic check for new files in the drop folder, and will start a loader task if one or more new files are found in the drop folder. Following a successful or failed load, input files will be renamed, deleted or moved based on the definitions in the **create scheduled load** command.

Starting Scheduled Loads Service

To enable scheduled loads, start the Jethro loads scheduler service:

```
service jethro start {Instance-name} loadscheduler
```

To stop the service, run:

```
service jethro stop {Instance-name} loadscheduler
```

Alternatively, this service can be set to start automatically upon the execution of the service start command. Edit the **$JETHRO_HOME/instances/services.ini** file and set the third parameter for this instance to **yes**:
vi $JETHRO_HOME/instances/services.ini
--> Instance-name:port:yes:yes:yes

Scheduling a Load

To schedule a load, run the following command from JethroClient:

```
create scheduled load

# Example: Creating a scheduled load, which starts every 15 minutes and loads a file from HDFS drop folder /data
/sales/ into table 'sales':
CREATE SCHEDULED LOAD sales FROM hdfs://mycluster:8020/data/sales/ SCHEDULE PERIODIC 15 MINUTES DESCFILE
hdfs://mycluster:8020/desc/sales.desc;
```

For further information, visit CREATE SCHEDULED LOAD page.