sql-lint Documentation

Release 0.0.11

Joe Reynolds

Jun 25, 2022

Contents:

CHAPTER 1

Introduction

1.1 What is sql-lint?

`sql-lint` is a linter for SQL dialects. It currently supports MySQL and Postgres. It brings errors to your attention, suggests what's wrong with them, why it may be wrong, and what you can do as a developer to fix it. Generally these errors are more verbose and specific than those coming from an SQL server.

Here's a small excerpt of its use:

```
: sql-lint test/test-files//test.sql
test/test-files//test.sql:16 [sql-lint: unmatched-parentheses] Unmatched parentheses.
test/test-files//test.sql:20 [sql-lint: missing-where] DELETE statement missing WHERE␣
↪clause.
test/test-files//test.sql:22 [sql-lint: invalid-drop-option] Option 'thing' is not a␣
↪valid option, must be one of '["database","event","function","index","logfile",
↪"procedure","schema","server","table","view","tablespace","trigger"]'.
test/test-files//test.sql:26 [sql-lint: invalid-truncate-option] Option 'something'␣
↪is not a valid option, must be one of '["table"]'.
test/test-files//test.sql:30 [sql-lint: odd-code-point] Unexpected code point.
test/test-files//test.sql:32 [sql-lint: invalid-limit-quantifier] Argument 'test' is␣
↪not a valid quantifier for LIMIT clause.
test/test-files//test.sql:24 [ER_PARSE_ERROR] You have an error in your SQL syntax;␣
↪check the manual that corresponds to your MySQL server version for the right syntax␣
↪to use near 'CREATE test person' at line 1
test/test-files//test.sql:39 [ER_NO_SUCH_TABLE] Table 'symfony.dont_exist' doesn't␣
↪exist
```

1.2 Usage

`sql-lint` is used from the command line in several ways.

1.2.1 Via stdin

```
echo "DELETE FROM person;" | sql-lint
```

1.2.2 With a file

```
sql-lint test-file.sql
```

1.3 Command line options

1.3.1 -V –version

The version of `sql-lint`. Useful for bug reports and confirming what features are available to you.

```
sql-lint --version
> 0.0.11
```

1.3.2 -d –driver

`mysql|postgres`

Default is `mysql`.

The driver to use.

1.3.3 -v –verbose

How verbose to be with output. `-v` will print out the output from the lexer. Usually you do not want any verbosity. Useful for bug reports and debugging.

```
sql-lint --verbose
> ...
```

1.3.4 –config

The path for the configuration file.

Default is `$HOME/.config/sql-lint/config.json`

1.3.5 –format

`simple|json`

Default is `simple`.

The output format of `sql-lint`.

`simple` is the most user friendly and human readable. You won't usually change the format unless you have a reason to.

```
echo 'DELETE FROM person;' | sql-lint
> stdin:1 [sql-lint: missing-where] DELETE statement missing WHERE clause.
```

`json` can be used if you wish. Usually this is done for editor integration or for consumption via some other service.

```
echo 'DELETE FROM person;' | sql-lint --format json
> {
    "source":"stdin",
    "error":"[sql-lint: missing-where] DELETE statement missing WHERE clause.",
    "line":1
}
```

1.3.6 –host

The host for the connection.

1.3.7 –user

The user for the connection.

1.3.8 –password

The password for the connection.

1.3.9 –port

Default is `3306`.

The port for the connection.

1.3.10 -h –help

```
: sql-lint -h
Usage: sql-lint [options]

Options:
  -V, --version          output the version number
  --fix [string]         The .sql string to fix
  -d, --driver <string>  The driver to use, must be one of ['mysql', 'postgres']
  -v, --verbose          Brings back information on the what it's linting and the␣
↪tokens generated
  --format <string>      The format of the output, can be one of ['simple', 'json']␣
↪(default: "simple")
  --host <string>        The host for the connection
  --user <string>        The user for the connection
  --password <string>    The password for the connection
  --port <string>        The port for the connection
  --config <string>      The path to the configuration file
  -h, --help             display help for command
```

1.4 Programmatic Access

```
import sqlLint from 'sql-lint'

// using async/await

const errors = await sqlLint({
  sql: 'SELECT my_column FROM my_table',
})

// or using promise

sqlLint({ sql: 'SELECT my_column FROM my_table' }).then(errors => {
  for (const error of errors) {
    // do something
  }
})
```

1.4.1 Parameters

sql-lint accepts an object using the following interface as its only argument

```
{
  sql: string
  host?: string
  user?: string
  port?: number
  driver?: string
  prefix?: string
  password?: string
  verbosity?: number
}
```

1.4.2 Notes on some of the parameters

`sql`: can have multiple queries separated by `;`

`host`: if host is not provided `sql-lint` will only perform checks that do not require a connection

`driver`: defaults to mysql

`port`: if port is not provided it will use the default port for the driver you are using

1.4.3 Output

sql-lint returns an `array` of objects with the following shape

```
{
  line: number
  error: string
  source: string
  additionalInformation: string
}
```

CHAPTER 2

Installation

Installation is simple. Download a pre-built binary from Github. Once you've installed `sql-lint`, you will want to configure it for the best experience.

`sql-lint` supports Mac, Linux, and Windows.

If you prefer, you can do `npm i -g sql-lint` or `yarn global add sql-lint` if you're using yarn.

2.1 Programmatic Access

For programmatic access you'll want to instead install `sql-lint` into your node project with `npm i sql-lint` or `yarn add sql-lint`.

CHAPTER 3

Configuration

Configuring `sql-lint` to connect to your database of choice allows even more errors to come through. Errors that `sql-lint` wouldn't find itself. To do this is easy, simply supply the connection details to your database in one of two ways:

3.1 Via CLI

```
sql-lint --driver="mysql" --host="localhost" --user="root" --password="hunter2"
```

3.2 Via File

`sql-lint` will search the current working directory and its parent directories for a configuration file `.sql-lint.json`. This allows you to have directory-local configurations for different projects. If no `.sql-lint.json` is found, it will fall back to the global configuration file.

A global configuration file for `sql-lint` can reside in `~/.config/sql-lint/config.json`. It follows the XDG Base Directory Specification. Specifically, it uses `$HOME/.config`.

You can also manually specify a path for the config with the `--config` flag.

You should put the following in there for more intelligent errors to come through

```
{
    "driver": "mysql",
    "host": "localhost",
    "user": "root",
    "password": "hunter2",
    "port": 3306
}
```

3.3 Configuration options

An exhaustive list of the configuration options for your `config.json` file are below.

3.3.1 driver

The driver to be used to check for errors. Accepted ones are `mysql` and `postgres`.

Optional, default is `mysql`.

3.3.2 host

The host of the database server.

3.3.3 user

The user for the database server.

3.3.4 password

The password for the database server.

3.3.5 port

The port to connect to.

Optional, default is `3306`.

3.3.6 ignore-errors

Don't want to be warned about a particular error? In that case add it to the `ignore-errors` array in `~/.config/sql-lint/config.json`.

```
{
    "host": "localhost",
    "user": "root",
    "password": "password",
    "ignore-errors": [
        "odd-code-point",
        "missing-where"
    ]
}
```

The example above will skip checks for odd code points and `DELETE` statements with missing `WHERE` clauses.

For a full list of all available checks, see the check documentation

You cannot skip checks that are returned from the DB server itself, only the checks built into `sql-lint`.

Note that this option is also available as a flag on the cli. i.e.

```
sql-lint --ignore-errors=trailing-whitespace some-sql-file.sql
```

Multiple errors can be comma separated:

```
sql-lint --ignore-errors=trailing-whitespace,missing-where,hungarian-notation some-
↪sql-file.sql
```

3.3.7 Example configuration

The below configuration contains every option available.

```
{
    "host": "localhost",
    "user": "root",
    "password": "password",
    "ignore-errors": [
        "odd-code-point",
        "missing-where",
        "invalid-drop-option",
        "invalid-create-option",
        "invalid-truncate-option",
        "invalid-alter-option",
        "hungarian-notation",
        "trailing-whitespace"
    ]
}
```

3.3.8 A word of warning

Do not version control your configuration file unless you know what you're doing. Stick it in your global `.gitignore` to be safe.

3.4 Editor Integration

`sql-lint` can integrate with any editor that supports external plugins.

3.4.1 Vim / Neovim

Ale

`sql-lint` can be integrated into (Neo)Vim with Ale.

Vanilla

If you want to go without a plugin, the simplest option is to run the following:

```
:!sql-lint %
```

CHAPTER 4

Checks

`sql-lint` comes with its own suite of checks. Aside from its own checks, it also returns any errors from the SQL server you have connected to. Generally you'll find that the errors from `sql-lint` are more informative than those from the server. That said, you will still want errors from the server as it covers more cases and will catch things that `sql-lint` does not.

4.1 Reference

4.1.1 unmatched-parentheses

Shown when a query has an unbalanced amount of parentheses.

Example output

```
test/test-files//test.sql:16 [sql-lint: unmatched-parentheses] Unmatched parentheses.
```

4.1.2 missing-where

Shown when a `DELETE` statement is missing a `WHERE` clause.

Example output

```
test/test-files/test.sql:20 [sql-lint: missing-where] DELETE statement missing WHERE␣
↪clause.
```

4.1.3 invalid-drop-option

Shown when an invalid option is given to the `DROP` statement.

Example output

```
test/test-files/test.sql:22 [sql-lint: invalid-drop-option] Option 'thing' is not a␣
↪valid option, must be one of '["database","event","function","index","logfile",
↪"procedure","schema","server","table","view","tablespace","trigger"]'.
```

4.1.4 invalid-create-option

Shown when an invalid option is given to the `CREATE` statement.

Example output

```
:24 [sql-lint: invalid-create-option] Option 'test' is not a valid option, must be␣
↪one of '["algorithm","database","definer","event","function","index","or","procedure
↪","server","table","tablespace","temporary","trigger","user","unique","view"]'.
```

4.1.5 invalid-truncate-option

Shown when an invalid option is given to the `TRUNCATE` statement.

Example output

```
test/test-files/test.sql:26 [sql-lint: invalid-truncate-option] Option 'something' is␣
↪not a valid option, must be one of '["table"]'.
```

4.1.6 invalid-alter-option

Shown when an invalid option is given to the `ALTER` statement.

Example output

```
test/test-files/test.sql:28 [sql-lint: invalid-alter-option] Option 'mlady' is not a␣
↪valid option, must be one of '["column","online","offline","ignore","database",
↪"event","function","procedure","server","table","tablespace","view"]'.
```

4.1.7 odd-code-point

Shown when there are unsupported/unusual* code points in your code.

*This check came about whilst working Microsoft Excel. Microsoft likes to add a lot of zany characters which can subtly break your data without you realising.

Example output

```
test/test-files//test.sql:30 [sql-lint: odd-code-point] Unexpected code point.
```

4.1.8 invalid-limit-quantifier

Shown when you specify something other than a number to the `LIMIT` statement.

Example output

```
test/test-files//test.sql:32 [sql-lint: invalid-limit-quantifier] Argument 'test' is␣
↪not a valid quantifier for LIMIT clause.
```

4.1.9 hungarian-notation

Shown when the string `sp_` or `tbl_` is present in the query.

Example output

```
test/test-files/test.sql:34 [sql-lint: hungarian-notation] Hungarian notation present␣
↪in query
```

4.1.10 trailing-whitespace

Shown when a query has trailing whitespace.

Example output

```
test/test-files/test.sql:34 [sql-lint: trailing-whitespace] Trailing whitespace
```

CHAPTER 5

Troubleshooting

5.1 I'm not seeing any warnings

Run `sql-lint your-file` and it will display the exception. Add the `-v` flag for more information.

5.2 It's telling me there's a syntax error when there's clearly not.

Chances are you're using an old(er) version of MySQL. `EXPLAIN`ing on `INSERT|UPDATE|DELETE` was added in Mysql 5.6.

CHAPTER 6

Development

If you're interested in helping further the development of `sql-lint` then read on. Casual users can ignore this section.

6.1 How it works

A raw query (either from stdin, a file, or a string) hits `main.ts`. This query then gets categorised into the type of statement it is (`SELECT`, `INSERT`, `UPDATE`, `DELETE` etc...), as the SQL grammar is pretty damn huge, there is a lexer per statement. This adds redundancy but increases flexibility.

Once a query has been categorised, it is then lexxed by the relevant lexer. See the `src/lexer` directory for the inner workings.

i.e. if we have the statement

```
SELECT name FROM user
```

This will hit the lexer which will categorise this as a `SELECT` statement which the `SELECT` lexer will then tokenise. The tokenised string is then passed through to every checker to look for any linting errors.

6.2 Adding a check

If you want to add your own check, read on. It's quite simple but also verbose.

This can probably be automated to make it WAY easier.

Anyway, here are the steps.

- Create a check under `src/checker/checks`
  - The name of the class is also the name of the checker so name it well
- Add your check to `src/barrel/checks.ts`

  - All checks live here so we can import them all conveniently
- Import your check in `src/checker/checkFactory.ts`
- Add your check to the `checkMap` in `src/checker/checkFactory.ts`
- Add it to the README.md so people know it's a thing
- Add it to `configuration.md`. This is an exhaustive list of the checks
- Add tests. The name of the test should match the name of the check
- Add it to `checks.md`, the main documentation for checks
- `npm run build` to compile the changes

6.3 Troubleshooting

6.3.1 `TypeError: checkMap[check] is not a constructor`

Your check is not being picked up by the `checkerRunner`. log out what the value of `checks` is in `checkerRunner` **after** the `splice`ing.

6.4 Testing the code

Testing requires sql-lint to be installed.

```
npm install -g sql-lint
./build/build.sh //This will run more than just the tests (recommended)
```

6.5 Using the Docker container

First, make sure port `3306` is available locally. (You can do this by inspecting the output of `sudo lsof -i :3306` and `docker ps` and `kill`ing anything using that port) Now do:

```
docker-compose up --build -d --force-recreate
```

At this point the container(s) will be up and ready to use. You can login with the following credentials: `mysql -u root -ppassword`.

Here's an example of a query:

```
docker exec sqllint_mysql_1 mysql -u root -ppassword -e "SHOW DATABASES"
```

6.5.1 Connecting sql-lint to the Docker container

Change your config file in `~/.config/sql-lint/config.json` to have the following values:

```
{
    "driver": "mysql",
    "host": "localhost",
    "user": "root",
    "password": "password",
    "port": 3306
}
```

6.6 This documentation

This documentation is built on `sphinx` and `readthedocs`. To run it locally, you will need the following:

- The `sql-lint` repository (documentation lies in `docs/`)
- `sphinx` to be installed (`pip install sphinx`)
- `sphinx-rtd-theme` to be installed (`pip install sphinx-rtd-theme`)
- `recommonmark` to be installed (`pip install recommonmark`)

Once those prerequisites are met, you can edit the files and see them exactly how they would appear on readthedocs. installed.