## How to contribute to Knex.js

- Make changes in the `/lib` directory.

- Before sending a pull request for a feature or bug fix, be sure to have

[tests](https://github.com/knex/knex/tree/master/test). Every pull request that changes the queries should have

also \*\*integration tests which are ran against real database\*\* (in addition to unit tests which checks which kind of queries

are being created).

- Use the same coding style as the rest of the

[codebase](https://github.com/knex/knex/blob/master/knex.js).

- All pull requests should be made to the `master` branch.

- Pull request description should have link to corresponding PR of documentation branch.

- All pull requests that modify the public API should be updated in [types/index.d.ts](https://github.com/knex/knex/blob/master/types/index.d.ts)

## Documentation

Documentation is no longer maintained in knex master repository. All the documentation pull requests should be sent to https://github.com/knex/documentation

Documentation pull requests should not be merged before knex version which has the new documented feature is released.

## I would like to add support for new dialect to knex, is it possible?

Currently there are already way too many dialects supported in `knex` and instead of adding new dialect to central codebase, all the dialects should be moved to separate npm packages out from `knex` core library with their respective maintainers and test suites.

So if you like to write your own dialect, you can just inherit own dialect from knex base classes and use it by passing dialect to knex in knex configuration (https://runkit.com/embed/90b3cpyr4jh2):

```js

// simple dialect overriding sqlite3 dialect to use sqlite3-offline driver

require('sqlite3-offline');

const Knex = require('knex');

const Dialect = require(`knex/lib/dialects/sqlite3/index.js`);

Dialect.prototype.\_driver = () => require('sqlite3-offline');

const knex = Knex({

client: Dialect,

connection: ':memory:',

});

console.log(knex.select(knex.raw(1)).toSQL());

await knex.schema.createTable('fooobar', (t) => {

t.bigincrements('id');

t.string('data');

});

await knex('fooobar').insert({ data: 'nomnom' });

console.log('Gimme all the data:', await knex('fooobar'));

```

## What is minimal code to reproduce bug and why I have to provide that when I can just tell whats the problem is

Writing minimal reproduction code for the problem is time-consuming and sometimes it is also really hard, for

example when the original code where the bug happens is written using express or mocha. So why is it necessary

for me to commit so much time to it when the problem is in `knex`? Contributors should be grateful that I reported

the bug I found.

The point of runnable code to reproduce the problem is to easily verify that there really is a problem and that the one

who did the report did nothing wrong (surprisingly often problem is in the user code). So instead of just description

what to do the complete code encourages devs to actually test out that problem exists and start solving it and it

saves lots of time.

tl;dr list:

1. Actually in most of the cases developer already figures out what was the problem when writing the minimal test case

or if there was problem how stuff was initialized or how async code was written it is easy to point out the problem.

2. It motivates developer to actually try out if the bug really exist by not having to figure out from incomplete example

environment in which and how bug actually manifests.

3. There are currently very few people fixing knex issues and if one has to put easily 15-30 minutes time to issue just

to see that I cannot reproduce this issue it just wastes development hours that were available for improving knex.

Test case should initialize needed tables, insert needed data and fail...

```

const knex = require('knex')({

client: 'pg',

connection: 'postgres:///knex\_test'

});

async function main() {

await knex.schema.createTable(...);

await knex('table').insert({foo: 'bar}');

await knex.destroy();

}

main();

```

Usually issues without reproduction code available are just closed and if the same issue is reported multiple

times maybe someone looks into it.

One easy way to setup database for your reproduction is to use database from knex's docker-compose setup (npm run db:start) and by checking the connection settings from tests' `test/knexfile.js`.

## Integration Tests

### The Easy Way

By default, Knex runs tests against sqlite3, postgresql, mysql, mysql2, mssql and oracledb drivers. All databases can be initialized and ran with docker.

Docker databases can be started and initialized and started with:

```bash

npm run db:start

```

and stopped with:

```bash

npm run db:stop

```

In case you don't need all of the databases, you can use simplified dev Docker configuration that only runs PostgreSQL, by running `npm run db:start:postgres` and `npm run db:stop:postgres` accordingly.

### Installing support for oracledb

Oracle has started providing precompiled driver libs for all the platforms, which makes it viable to run oracle tests also locally against oracledb running in docker.

Check message when running

```bash

npm install oracledb

```

and download driver library binary packages and unzip it to ~/lib directory.

### Specifying Databases

You can optionally specify which dialects to test using the `DB` environment variable. Values should be space separated and can include:

- mysql

- mysql2

- postgres

- sqlite3

- oracledb

- mssql

```bash

$ DB='postgres mysql' npm test

```

### Custom Configuration

If you'd like to override the database configuration (to use a different host, for example), you can override the path to the [default test configuration](https://github.com/knex/knex/blob/master/test/knexfile.js) using the `KNEX\_TEST` environment variable.

```bash

$ KNEX\_TEST='./path/to/my/config.js' npm test

```

### Creating Postgres User

If you are running tests against own local database one might need to setup test user and database for knex to connect.

To create a new user, login to Postgres and use the following queries to add the user. This assumes you've already created the `knex\_test` database.

```

CREATE ROLE postgres WITH LOGIN PASSWORD '';

GRANT ALL PRIVILEGES ON DATABASE "knex\_test" TO postgres;

```

Once this is done, check it works by attempting to login:

```

psql -h localhost -U postgres -d knex\_test

```

## Want to be Collaborator?

There is always room for more collaborators. Be active on resolving github issues / sending pull requests / reviewing code and we will ask you to join.

### Etiquette (/??t?k?t/ or /??t?k?t/, French: [e.ti.k?t])

Make pull requests for your changes, do not commit directly to master (release stuff like fixing changelog are ok though).

All the pull requests must be peer reviewed by other collaborator, so don't merge your request before that. If there is no response ping others.

If you are going to add new feature to knex (not just a bugfix) it should be discussed first with others to agree on details.

Join Gitter chat if you feel to chat outside of github issues.