# Contributing to Cube.js

Thanks for taking the time for contribution to Cube.js!

We're very welcoming community and while it's very much appreciated if you follow these guidelines it's not a requirement.

## Code of Conduct

This project and everyone participating in it is governed by the [Cube.js Code of Conduct](./CODE\_OF\_CONDUCT.md).

By participating, you are expected to uphold this code. Please report unacceptable behavior to info@statsbot.co.

# Contributing Code Changes

Please review the preceding section before proposing a code change. This section documents how to do so.

\*\*When you contribute code, you affirm that the contribution is your original work and that you license the work to the project under the project’s open source license. Whether or not you state this explicitly, by submitting any copyrighted material via pull request, email, or other means you agree to license the material under the project’s open source license and warrant that you have the legal authority to do so.\*\*

## Contribution Prerequisites

Cube.js works with Node.js 8+ and uses yarn as a package manager.

## Development Workflow

### Cube.js Client

1. After cloning Cube.js repository run `$ yarn` in `packages/cubejs-client-core` and `packages/cubejs-client-react` to install dependencies.

2. Use `$ yarn link` to add these packages to link registry.

3. Perform required code changes.

4. Use `$ yarn build` in the repository root to build CommonJS and UMD modules.

5. Use `$ yarn link @cubejs-client/core` and/or `$ yarn link @cubejs-client/react` in your project to test changes applied.

6. Use `$ yarn test` where available to test your changes.

7. Ensure commit CommonJS and UMD modules as part of your commit.

To get set up quickly, you can perform 1) and 2) with one line from the `cube.js` clone root folder:

```

cd packages/cubejs-client-core && yarn && yarn link && cd ../.. && cd packages/cubejs-client-react && yarn && yarn link && cd ../..

```

### Implementing Driver

1. Copy existing driver package structure and name it in `@cubejs-backend/<db-name>-driver` format.

`@cubejs-backend/mysql-driver` is very good candidate for copying this structure.

2. Please do not copy \*CHANGELOG.md\*.

3. Name driver class and adjust package.json, README.md accordingly.

4. As a rule of thumb please use only Pure JS libraries as a dependencies where possible.

It increases driver adoption rate a lot.

5. Typically you need to implement only `query()` and `testConnection()` methods of driver.

The rest will be done by `BaseDriver` class.

6. If db requires connection pooling prefer use `generic-pool` implementation with settings similar to other db packages.

7. Make sure your driver has `release()` method in case DB expects graceful shutdowns for connections.

8. Please use yarn to add any dependencies and run `$ yarn` within the package before committing to ensure right `yarn.lock` is in place.

9. Add this driver dependency to [cubejs-server-core/core/index.js](https://github.com/statsbotco/cube.js/blob/master/packages/cubejs-server-core/core/index.js#L8).

### Implementing JDBC Driver

If there's existing JDBC Driver in place for Database of interest you can just create `DbTypes` configuration inside

[cubejs-jdbc-driver/driver/JDBCDriver.js](https://github.com/statsbotco/cube.js/blob/master/packages/cubejs-jdbc-driver/driver/JDBCDriver.js#L31).

Most of times no additional adjustments required for base `JDBCDriver` implementation as JDBC is pretty standard.

In case you need to tweak it a little bit please follow [Implementing Driver](#implementing-driver) steps but use `JDBCDriver` as your base driver class.

### Implementing SQL Dialect

1. Find the most similar `BaseQuery` implementation in `@cubejs-backend/schema-compiler/adapter`.

2. Copy it, adjust SQL generation accordingly and put it in driver package. Driver package will obtain `@cubejs-backend/schema-compiler` dependency from that point.

3. Add `static dialectClass()` method to your driver class which returns `BaseQuery` implementation for the database. For example:

```javascript

const BaseDriver = require('@cubejs-backend/query-orchestrator/driver/BaseDriver');

const FooQuery = require('./FooQuery');

class FooDriver extends BaseDriver {

// ...

static dialectClass() {

return FooQuery;

}

}

```

If driver class contains `static dialectClass()` method it'll be used to lookup corresponding SQL dialect. Otherwise default dialect for the database type will be used.

### Publishing Driver npm Package

Cube.js looks up `{dbType}-cubejs-driver` package among installed modules to fullfil driver dependency if there's no corresponding default driver for the specified database type.

For example one can publish `foo-cubejs-driver` npm package to fullfil driver dependency for the `foo` database type.

### Testing Schema Compiler

In order to run tests in `cubejs-schema-compiler` package you need to have running [Docker](https://docs.docker.com/install/) on your machine.

When it's up and running just use `$ npm test` in `packages/cubejs-schema-compiler` to execute tests.

### Linking Server Core for Development

It's convenient to link `@cubejs-backend/server-core` into your project for manual tests of changes of backend code.

Cube.js uses `yarn` as package manager instead of `npm`.

In order to link `@cubejs-backend/server-core`:

1. Create new project using `cubejs create` or use existing one.

2. Install yarn: `npm install -g yarn`.

3. Link server-core package: `yarn link` inside `packages/cubejs-server-core`.

4. Link all drivers and dependent packages where you make changes in `packages/cubejs-server-core`.

5. Run `yarn build` in `packages/cubejs-playground`.

6. Install dependencies in all linked packages using `yarn`.

7. Run `yarn link @cubejs-backend/server-core` in your project directory.

### Client Packages

If you want to make changes to the Cube.js client packages and test them locally in your project you can do it the following way:

1. Make the desired changes and run `yarn build` in the root directory (you can also use `yarn watch`)

2. Go to the `~/some-path/cube.js/packages/cubejs-client-core` directory and run `yarn link`. (You'll see the messages \_Registered \*\*"@cubejs-client/core"\*\*\_)

3. Now you can link it in your project (e.g. \_/my-project/dashboard-app\_). You can do so running `yarn link "@cubejs-client/core"`

If you want to make changes to the `@cubejs-client/react` package you'll need a few extra steps

1. Go to your project's \*\*node\_modules\*\* directory and find the react package (e.g. \_/my-project/dashboard-app/node\_modules/react\_ and run `yarn link`

2. Go to the `~/some-path/cube.js/packages/cubejs-client-react` directory and run `yarn link react`

Now your project will be using the local packages.

\*\*NOTE:\*\* You might need to restart your project after linking the packages.

## Style guides

We're passionate about what code can do rather how it's formatted.

But in order to make code and docs maintainable following style guides will be enforced.

Following these guidelines is not a requirement but you can save some time for maintainers if you apply those to your contribution beforehand.

### Code

1. Run `npm run lint` in package before committing your changes.

If package doesn't have lint script, please add it and run.

There's one root `.eslintrc.js` file for all packages except client ones.

Client packages has it's own `.eslintrc.js` files.

2. Run `npm test` before committing if package has tests.

3. Please use [conventional commits name](https://www.conventionalcommits.org/) for your PR.

It'll be used to build change logs.

All PRs are merged using squash so only PR name matters.

4. Do not reformat code you aren't really changing unless it's absolutely necessary (e.g. fixing linter). Such changes make it really hard to use git blame feature when we need to find a commit where line change of interest was introduced.