# Contributing

## Code of Conduct

This project and everyone participating in it is governed by a [Code of Conduct](CODE\_OF\_CONDUCT.md). By participating, you are expected to uphold this code. Please report unacceptable behavior to [quack@duckdb.org](mailto:quack@duckdb.org).

## \*\*Did you find a bug?\*\*

\* \*\*Ensure the bug was not already reported\*\* by searching on GitHub under [Issues](https://github.com/cwida/duckdb/issues).

\* If you're unable to find an open issue addressing the problem, [open a new one](https://github.com/cwida/duckdb/issues/new). Be sure to include a \*\*title and clear description\*\*, as much relevant information as possible, and a \*\*code sample\*\* or an \*\*executable test case\*\* demonstrating the expected behavior that is not occurring.

## \*\*Did you write a patch that fixes a bug?\*\*

\* Great!

\* If possible, add a unit test case to make sure the issue does not occur again.

\* Make sure you run the code formatter (`make format`).

\* Open a new GitHub pull request with the patch.

\* Ensure the PR description clearly describes the problem and solution. Include the relevant issue number if applicable.

## Branches

\* Do not commit/push directly to the master branch. Instead, create a feature branch/fork and file a merge request.

\* When maintaining a branch, merge frequently with the master.

\* When maintaining a branch, submit merge requests to the master frequently.

\* If you are working on a bigger issue try to split it up into several smaller issues.

## Testing

\* `make unit` runs the \*\*fast\*\* unit tests (~one minute), `make allunit` runs \*\*all\*\* unit tests (~one hour).

\* Make sure \*\*all\*\* unit tests pass before merging into the master branch.

\* Write many tests

\* Test with different types, especially numerics and strings

\* Try to test unexpected/incorrect usage as well, instead of only the happy path

\* Slower tests should be added to the \*\*all\*\* unit tests. You can do this by adding `[.]` after the test group. For an example see `test\_tpch.cpp`.

\* Look at the code coverage report of your branch and attempt to cover all code paths in the fast unit tests. Attempt to trigger exceptions as well. It is acceptable to have some exceptions not triggered (e.g. out of memory exceptions or type switch exceptions), but large branches of code should always be either covered or removed.

## Formatting

\* Tabs for indentation, spaces for alignment

\* 120 columns

\* `clang\_format` enforces these rules automatically, use `make format` to run the formatter.

## C++ Guidelines

\* Do not use `malloc`, prefer the use of smart pointers

\* Strongly prefer the use of `unique\_ptr` over `shared\_ptr`, only use `shared\_ptr` if you \*\*absolutely\*\* have to

\* Do \*\*not\*\* import namespaces in headers (e.g. `using std`), only in source files

\* When overriding a virtual method, avoid repeating virtual and always use override or final

\* Use `[u]int(8|16|32|64)\_t` instead of int, long, uint etc. In particular, use `index\_t` instead of `size\_t` for offsets/indices/counts of any kind.

\* Prefer using references over pointers

\* Use C++11 for loops when possible: for (const auto& item : items) {...}

\* Use braces for indenting `if` statements and loops. Avoid single-line if statements and loops, especially nested ones.

\* \*\*Class Layout:\*\* Start out with a `public` block containing the constructor and public variables, followed by a `public` block containing public methods of the class. After that follow any private functions and private variables.

## Error Handling

\* Use exceptions \*\*only\*\* when an error is encountered that terminates a query (e.g. parser error, table not found). Exceptions should only be used for \*\*exceptional\*\* situations. For regular errors that does not break the execution flow (e.g. errors you \*\*expect\*\* might occur) use a return value instead.

\* Try to add test cases that trigger exceptions. If an exception cannot be easily triggered using a test case then it should probably be an assertion. This is not always true (e.g. out of memory errors are exceptions).

\* Use \*\*assert\*\* only when failing the assert means a programmer error. Assert should never be triggered by a user input. Avoid code like `assert(a > b + 3);` without comments or context.

\* Assert liberally, but make it clear with comments next to the assert what went wrong when the assert is triggered.

## Naming Conventions

\* Files: lowercase separated by underscores, e.g., abstract\_operator.cpp

\* Types (classes, structs, enums, typedefs, using): CamelCase starting with uppercase letter, e.g., BaseColumn

\* Variables: lowercase separated by underscores, e.g., chunk\_size

\* Functions: CamelCase starting with uppercase letter, e.g., GetChunk

\* Choose descriptive names.

\* Avoid i, j, etc. in \*\*nested\*\* loops. Prefer to use e.g. \*\*column\_idx\*\*, \*\*check\_idx\*\*. In a \*\*non-nested\*\* loop it is permissible to use \*\*i\*\* as iterator index.