C++ Style Guide

Impala C++ Code follows a modified version of the Google C++ Style Guide at:

https://google.github.io/styleguide/cppguide.html

There are some key differences which are documented here.

Header Files

1. Separate inline headers: We allow putting inline functions in separate files, using the suffix `.inline.h`. This can speed up compile times by reducing the volume of code to be compiled and reducing dependencies between headers.
2. Header include guards: for new code, we prefer #pragma once because it is cleaner than the classic #include guards

Variable Naming

We use UPPER_CASE for constants instead of kConstantName.

Scoping

1. namespaces the `using namespace` directive is allowed in .cc files in limited cases where it greatly reduces code volume.
    a. Pros: reduces code volume, less churn in "using namespace::class" directives.
    b. Cons: pollutes the namespace, causes conflicts, makes it more difficult to determine the type of an object

Commenting

- TODO Comments we typically do not include the name in the TODO comment. Where possible it's best to fix the TODO immediately (if it is a small task) or create a JIRA to track a more substantial improvement.

Formatting

The .clang-format file checked into the Impala repository should be used to format whitespace (see Contributing to Impala for more info). The .clang-format file is the source of truth for whitespace formatting, except when its output significantly diverges from practices in the existing codebase or from common sense. In those cases .clang-format should be updated. We aim to gradually adapt the codebase to the output of clang-format Therefore we only recommend using it on lines that have non-whitespace changes. This can be accomplished with the git-clang-format tool.

Some key differences from the Google C++ style are:

1. Line Length We use 90 character line lengths
2. Function Declaration we line wrap differently than Google, typically packing more parameters per line, e.g.:

```
// Google Recommends:
ReturnType LongClassName::ReallyReallyReallyLongFunctionName(
    Type par_name1,  // 4 space indent
    Type par_name2,
    Type par_name3) {
  DoSomething();  // 2 space indent
  ...
}
// we use:
ReturnType LongClassName::ReallyReallyReallyLongFunctionName(
    Type par_name1, Type par_name2, Type par_name3) { // 4 space indent
  DoSomething();  // 2 space indent
  ...
}
```

3. Conditionals we format conditionals as follows

```
// Google Recommends:
if (x == kFoo) return new Foo();
if (condition)
  DoSomething();  // 2 space indent.
if (condition) {
  DoSomething();  // 2 space indent.
}
// we only use:
if (x == kFoo) return new Foo();  // If the whole line fits into the 90 character limit
if (condition) {
  DoSomething();  // Otherwise, 2 space indent.
}
```

Re-formatting

You can use clang-format to automatically reformat only the parts of the code you touched. See https://clang.llvm.org/docs/ClangFormat.html#script-for-patch-reformatting for a general clang-format reference. Impala's clang-format file is located at ${IMPALA_HOME}/.clang-format. The clang-* command uses the format file.

To reformat the code around the lines that your last commit touched, you can run:

```
git diff -U0 --no-color HEAD^ | "${IMPALA_TOOLCHAIN}/llvm-${IMPALA_LLVM_VERSION}/share/clang/clang-format-diff.
py" -binary "${IMPALA_TOOLCHAIN}/llvm-${IMPALA_LLVM_VERSION}/bin/clang-format" -p1 -i
```

The -i option means in-place replacement. Run it without -i option if you'd like to see difference before applying the effect of the command.

If the above approach does not work or you may want to use a different version, you can use a system version of clang-tidy. On Ubuntu,

```
sudo apt-get install clang-format-3.9
git diff -U0 --no-color HEAD^ | clang-format-diff-3.9 -p1 -i
```

Tip

clang-format is not perfect and sometimes reformats things in strange ways, or will reformat large areas of code. If clang-format doesn't match the existing style of the surrounding code, please consider manually formatting that part of the code - mixing large white

Third-Party Libraries

- Boost - we use a different set of Boost libraries. Reducing # of dependencies is encouraged and adding dependencies to new libraries should be carefully evaluated.

Tidy Code

You can check that your code is tidy with clang-tidy:

```
git diff asf-gerrit/master | "${IMPALA_TOOLCHAIN}/llvm-${IMPALA_LLVM_VERSION}/share/clang/clang-tidy-diff.py" -
clang-tidy-binary "${IMPALA_TOOLCHAIN}/llvm-${IMPALA_LLVM_VERSION}/bin/clang-tidy" -p 1
```