# Database AS3 Phase 2 Report

## Parsing logic

Surprisingly, our implementations were nearly identical to the TA's implementation, aside from differences in variable names. The similarity goes all the way down to using a getter method to retrieve the flag variable from a QueryData determining whether the query is an explain query or not.

## Planner logic

We've also correctly added the optional explain plan as the last step in the planner's plan tree construction, again identical to the TA's implementation aside from method name differences.

## Plans

This is where the main differences between our implementation and the TA's implementation lie.

For creating the explain string, the TA's implementation relies on overriding `toString()` to print each plan subtree. This means that when performing `toString()`, a plan always assumes that it at the root level, and the proper indentations at each level have to be added indirectly by the ancestor plans *after* the string has already been built. A lot of unnecessary copying and rebuilding of basically the same string has to occur as a result, and similar logic is written over and over again across *every* plan.

Our implementation has better code reuse, by adding a `void explain(StringBuilder sb, int numIndents)` method to the `Plan` interface, which permits each plan to define how its explain string's hierarchy is arranged (may vary for plans with one or multiple sub-plans), and a `StringBuilder addOptionalInfo(StringBuilder sb)` method that lets each plan define optional info to be included in its explain string, so that the rest of the required string format can be organized into a single static method that provides the correct formatting for each single plan's explain string - `void ExplainPlan.explainNode(Plan p, StringBuilder sb, int numIndents)`.

By separating the three main components of an explain string into different methods (how a plan's explain string is placed relative to its children, how a plan's explain string is formatted, and what a plan's optional info contains), we achieved better code reuse, and through passing the number of indents needed down the recursive invocations, we made it so the same `StringBuilder` only needs to build the entire string once, with no wasted intermediate results. This should be slightly more performant than the TA's implementation, too.

## Explain Scan

The main difference here is that we delegated the actual construction of the explain plan string to the first invocation of `ExplainScan.next()`, whereas the TA's implementation constructs the explain plan string when the `ExplainPlan` is opened, passed to the `ExplainScan`.

Another (somewhat meaningless) difference is that the TA's implementation uses the `ExplainPlan`'s schema for `ExplainScan`, whereas we wrote the same schema (implicitly) twice.

And another interesting difference is that the TA's implementation returns a generic `VarcharConstant` constructed from the resulting explain string directly, whereas we explicitly specified it to be of type

`Type.VARCHAR(500)`. It appears that calling the `VarcharConstant` constructor with just the input string and nothing else will result in the actual length of the `VARCHAR` type to be dynamic - same as the length of the input string. This might be a deviation from the spec on the TA's part. 😉