Database Integrity Feasibility Report for Original Purposes

Goals: Understand the current project and how we can transform it for better scalability as the client intended it to be.

Assumptions about scalability: We want to able to have multiple shims that are as lightweight as possible. These are to be used by users to perform normal JDBC queries and updates. The shims point/connect to a transformer. The transformer then interacts with the database and returns queries when needed back to the shim. The transformer is used to do most of the bulk work involving character insertion and hashing and shim is just used to translate the commands to this standalone application.

Current Project Status:

As per the prototype given to us we have been working through Java Native Interface(JNI) Bridge which works by compiling cpp code into a native folder and it being called as functions in the cpp. One thing that was quickly realized is that the cpp does not have any JDBC support and that the all of the functions that we need to modify are in the shim itself. The cpp functions that are in the JNI are just single functions that replace a given character in a string or long for example. It does not actually process statements. Those are processed by the shim when we call executeUpdate. So in essence it is transforming the data, but it is just overriding very simple processes done in the JDBC. So in conclusion, the pipeline works as following: We run JDBC functions as normal in a java file(JDBCTest in github) that accesses JDBC functions as it normally would except we override some functions and pass them to the JNI for data to be transformed. These only run when we run something like executeUpdate(One giant optimized function that we would not like to touch). We do not want to mess with executeUpdate because it has a lot of legacy code that is very difficult to understand and change.

Issues:

1. The JNI bridge must be in the same jar file. Through research I have not found anything that would help with this. That means theoretically the code we wrote in the shim and the cpp will always be together. Basic research into using sockets to abstract the cpp with bytestreams has not given any good results. Even if we were to separate the JNI, we would have to write our own TCP protocols with sockets to communicate between Java and cpp.
2. The shim has most of our functionality. The hashing can only be done in the shim as it is. All of the information for things like create table statement stays in the shim. Unless we completely rewrite most of the code in cpp. We cannot port our code. There is a cpp interface for postgres, but we would still have to deal with Java and Cpp talking with each other efficiently and safely. Also, we would be throwing away 2 months of work.
3. The scope of our project would be massive. Many of these issues as described above would be huge technical problems as they do not seem to be established problems, we can just look up on stack overflow. We have spent a lot of time just understanding the prototype as it is now.

Alternatives:

* Maybe keep it as it is and host it on its own server and use API calls to interact with it. At this point we could make a new shim in Java that could maybe emulate JDBC functionality. Doability: Does not sound terrible. We could work on effiency and emulating the JDBC interface second semester with the knowledge we already accumulated with working on the project as is. Possibly port more code into Cpp making it more efficient. The shim that we are building now could also be used as a standalone driver for personal use.
* Make a standalone Cpp application that can handle requests with sockets and worry about emulating JDBC end of semester 2. We can use a parser to handle requests efficiently and mask and hash data efficiently as well. We can have someone else worry about making it look like JDBC later and have that as a stretch goal. The purpose of this would to focus on a robust transformer that would have all the functionality it needs and JDBC can be worried about later. Doability: The team has no experience with writing Cpp applications but since we are writing from scratch, we could probably achieve this with some legwork. We can use what we learned from the earlier shim and we would just have to use an interface to put in the correct statements. However, this could have its own problems not yet discovered.
* Figure out some way of separating the JNI bridge and move most of the work to the transformer. This was the original goal and we could try doing this in semester 2. Doability: Seems to be the hardest of the solutions. We would have issues with Cpp and Java talking in bytestreams to each other that would be very hard to debug and write our own protocols for. In addition we would have to cater having multiple shims. Most of code as it currently is would not make the shim light weight at all and we still run executeUpdate in the shim. If we wanted to use a cpp interface in the transformer and separate it, we would not use any code we have written so far with hashing, tracking chars and writing to JSON.