Project Description

Project Outline

Your CSC 468 project is to build a miniature Native XML Database Management System. Following database course projects tradition, we call this system MiniNXBase.

This document contains an informal description of MiniNXBase. Formal specifications will be given for each stage of the project.

MiniNXBase is a collection of native XML index structures with a query processing engine built on top of it. In the context "native" means that the XML data is stored in specially designed index structures rather than as a collection of relations in a DBMS. The back-end of MiniNXBase will be the NEUStore Java package.

MiniNXBase will utilize client-server architecture, but will, otherwise, be a single-user system with no user account management, concurrency control or other multi-user system features. Some of this functionality will be left for extra credit, time permitting.

MiniNXBase functionality will include the following:

- creation of XML repositories;
- storage of XML files in XML repositories;
- querying of created XML repositories using a subset of XPath language, called here XPLite.

When completed, MiniNXBase will come as two executables: the DBMS server `MiniNXBase`, and the thin client, `XPLitePlus`. The client will have two modes: interactive and batch. In intercative mode, `XPLitePlus` will prompt user for a command, execute it, display the result, and prompt for a new command. In batch mode, `XPLitePlus` will execute a command passed to it, display/output the result and stop execution.

The role of the client is mostly to provide the front-end interaction with the user, assure syntactic correctness of the commands, and to display the results of the commands to the user. All syntactically correct commands are passed to the `MiniNXBase` server for execution. MiniNXBase will (a) parse the command, (b) construct the plan for executing the command, (c) execute the command by accessing necessary data stored in its index structures, (d) obtain results, (e) produce output if any and (f) pass the output to `XPLitePlus`.

Project Architecture Overview

MiniNXBase server consists of the following layers (from bottom to top):

1. **Disk storage and Buffer Manager.** The bottom layer of MiniNXBase is completely serviced by the NEUStore package. The package provides support for paginated disk data access and has an implementation of an LRU (Least-recently used) buffer management policy, which is sufficient for the purposes of this project. You will not need to build any code for this layer, however, acquaintance with NEUStore and its operations may be needed and will be facilitated through a lab exercise.

2. **Storage Layer**. This layer consists of a number of native XML indexes built on top of the NEUStore package. The detailed description of the index structures is released as a separate document. The API for this layer is given in Stage 1 documentation.

3. **XML Operations Layer**. This intermediate layer of operations provides implementations of some of the standard XML tree traversal operations used to execute XPLite queries. Methods of this layer will use Storage Layer API to access data, and will, in turn, be used by the Query Processing Layer. The API for this layer will be given in Stage 2 documentation.

4. **Query Processor**. This layer of MiniNXBase is responsible for executing XPLite queries. XPLite query compilation, and any optimization will happen in this layer.

5. **Query Parser**. This is the top layer of MiniXBase. It is responsible for taking the contents of the `<Command>` as input, parsing it, and invoking the query processor.

6. **XML Generator**. This is not a separate layer of the MiniNXBase architecture. However, it may be convenient to think of it as a separate component of MiniNXBase (depending on how you choose to implement the Query Processor layer). XML Generator uses the results obtained by the Query Processor (which need not be in the form of XML yet) and produces XML output that is returned to the user.

   Stage 3 of the project will involve building the Query Processor, Query Parser and the XML Generator.

Project Logistics

This is a group project. I have broken the class into four groups. Three groups have four members, one group has three members.

There is enough of programming in the core components of MiniNXBase (storage layer, operations layer, query processor) to ensure that every member of each team gets an adequate hands-on experience with building index structures and implementing query processing algorithms.

Project implementation will proceed in three stages, outlined above (Storage Layer, Operations Layer and Query Engine).

Stage 1 will involve building a large API to ensure convenient data access in the upper layers of MiniNXBase.

On **Stage 2**, the client-server architecture will emerge, and data management commands (management of repositories, data insertion, data deletion) will be completed. Portions of the query parser responsible for handling "DDL" and "DML" commands will need to be written as well.

Stage 3 will be completely devoted to XPLite query processing.

Time permitting, an extra-credit **Stage 4** may also take place. At this stage, I will release a list of MiniNXBase extensions, and each team would be able to choose the one(s) it wants to pursue. Possible MiniNXBase extensions are: enhanced query optimization, concurrency control, implementation of XML updates[1], logging/crash recovery, extension of XPLite to cover more XPath/XQuery functionality.

Each team gets one grade per project stage. I will consider complaints about (non)participation of individual group members in the project only if the circumstances are extraordinary.

Grading

Grading will be based solely on the correctness of operation of your software on the set of tests.

I will release a test suite for each project stage at least one week prior to the due date for that stage's submissions. I will keep a smaller private set of tests hidden. Your submission will be graded based on its performance on both the released and the private test sets. The grade will be the percentage of the tests[2] your program passes successfully.

Generally, the final project grade is the grade for the final stage of the project (since correctly executing tests for that stage would imply that all layers of your implementation of MiniNXBase perform correctly). However, we may also con-

[1] One of the chief simplifications of MiniNXBase is the fact that addition of data to XML repositories occurs on document basis. Core MiniNXBase functionality does not allow for addition/updates of individual XML elements inside an existing XML document.

[2] Possibly weighted, if some tests are deemed to be more important than others.

sider a weighted sum of your grades for each individual stage, it that number is better[3]

[3]If your early submissions had bugs in them, which were fixed by the final submission, you benefit from the first grading approach. If your early submissions were good, but you did not get the final submission working properly, you benefit from the second approach.