

CIS6930 Spring 2009, Project 4

This is an open project to be finished by groups of **three** students. You can propose your own topic or choose one from the topics I recommend (see the bottom of this page for details). These topics listed also give you some examples of the magnitude and flavor of the projects I wish to see. Basically, you are expected to implement a non-trivial data structure or algorithm (preferably within a database management system). For example, you can implement the quad tree in PostgreSQL. Application development on top of a DBMS (e.g., building an online bookstore with a database backend) is not encouraged unless it carries substantial academic prestige.

Project requirements

1. Your code should be clearly documented and commented. Code without detailed comments are generally regarded as of little value to people who takes over the project or simply wants to understand your design. You are required to submit all the relevant code and accompanying documents.
2. I expect you to write a report in the format of a publishable paper. In the report, you should clearly state the objective of your project, your design of the data structure or algorithm, and relevant testcases by which the grader can evaluate your code.
3. Your project will be graded according the functionality you implemented and the details and format of your report. The report and code should be submitted via Blackboard by 11:55pm, April 26, 2009 (Sunday).

This is a group project. Once you form a group, please choose a topic to work on and inform the instructor of the topic. I also recommend you choose one person as the team leader to represent your group in communicating with the instructor and coordinate your activities inside the group.

The following are two sample topics that you can choose from. The following descriptions are very concise; please schedule an appointment with the instructor for more details.

1. Implementation of a **density map** based on quad-tree in PostgreSQL.
A density map is a data structure that divides the data space into regions and records some aggregates (e.g., COUNT) of all data points in these regions. Its implementation can be accomplished by augmenting a quad tree with such aggregates. In this project, you are provided with a copy of quad tree implemented within PostgreSQL and your task is to install this implementation of quad tree in the latest version of PostgreSQL (modify the current quad tree code if you have to), apply the relevant changes to implement the density map, implement at least one algorithm to use and maintain (e.g., searching, insertion, and deletion) the density map. Estimation of the workload: moderate amount of code to read/understand, 200+ lines of code to write.
2. Memory allocator in the buffer manager. The current buffer management in PostgreSQL (and in almost all DBMSs) does not set a memory quota to each query process. However, in some systems where there are multiple long-running concurrent

queries, it is worth building a mechanism that allocates memory to each individual query for the purpose of ensuring fairness in resource distribution. For this, we need to modify the current data structure (i.e., an array of bufferdescriptors) in the buffer manager that manages all the buffer pages as a whole. Instead, we have to keep multiple lists of pages, each list serving one query. If the memory quota of one query is used up, this query will have to find a victim page within its own set of pages. If no pages from its own pool is available, this query stops and waits till one page is available. Estimation of workload: small amount of code to read (thanks to project 3, you are already familiar with the buffer manager code), simple data structure/algorithm design, 300-400 lines of code to write.

3. Implementation of an algorithm to compute continuous SDH. The Spatial Distance Histogram (SDH) problem is stated as: **given N points in a 2D space, draw a histogram of all point-to-point distances.** A histogram can be described by three parameters: minimum distance l , maximum distance u , and resolution r . Therefore, such a histogram consists of the counts of pairwise distances that fall into ranges $[l, l+r)$, $[l+r, l+2r)$, $[l+2r, l+3r)$, Each such range is also called a *bucket* of the histogram. We have proposed an efficient algorithm to compute SDH in a single data frame. However, we are often required to compute the SDH of consecutive data frames. I have an idea to reduce the processing time of such SDH in consecutive frames given the SDH of the first frame of this series and your job is to implement this idea in a standalone (C or Java) program. No hacking of PostgreSQL code is needed. This requires 500-600 lines of code. Anand Kumar will lead this project, he is still looking for partners.
4. Stripped PostgreSQL DBMS. This ought to be a fun project. Instead of writing some chunks of code to implement some data structure or algorithm, you are required to **disable** some functionality of the DBMS. Such things are sometimes required for testing the performance overhead of a specific DBMS module. For example, for concurrency control purposes, locking is done when a query requests to read/write data. However, locking is meaningless when we only read data from a database and it becomes a burden to have locking mechanism on under such situation. Then one thing we can do is to disable all locking-related operation at runtime (and see how much time we save). You only have to write a small amount of code for testing. But be careful, it does not mean it is an easy project; the PostgreSQL may not come with a global flag that you can simply reset to disable locking. Instead, you have to touch many lines of code to make sure no locking is being performed. I can point you to a paper to get started. Code hacking is the only thing you need to worry about in this project.
5. A random query search algorithm in PostgreSQL. PostgreSQL adopts a tree-based algorithm to search through the large number of possible plans for query optimization purposes. Your job in this project is to implement a strategy to find query plans in a random way. You can first define a total number of plans (N) to visit, then the algorithm randomly choose N plans from the search space and evaluates the cost, and selects the one with lowest cost to execute. The key is to understand the data structure

used in the current PostgreSQL code to store the information of plans. For this project, you can get help from one of my PhD students who is very familiar with the query optimizer in PostgreSQL. Rumors are that some type of random query optimizer has been implemented and you can probably benefit from those implementations (using others' code is not regarded cheating here as long as you cite the source in your code). There will be 300-400 lines of code to write.

6. A composite virtual DB communication framework. The databases can now be run under virtual machines (different OSs running on the same piece of hardware). However, for research purposes, we sometimes need to make the DB interact with the virtualization machine controller (a software module that controls the multiple OS in the same physical machine). Your job is to enable such interactions. The main (specific) goal is to pass some internal status information of the DBMS to another process (i.e., the virtual manager). You have to design this communication channel and decide what tools you use to realize it. (e.g., mailboxes, shared file, or sockets). This project involves minimal hacking of the PostgreSQL code but requires familiarity of inter-process communication mechanisms. There will be 200-300 lines of code to write.

Again, feel free to propose your own topics related to database systems.