CS 457/657 Database Management Systems

Programming Assignment 4: Transactions

Overview

In this assignment you will write a program that allows a database user to enclose SQL statements into a transaction block. That is, you will implement the "all-or-nothing" property commonly seen in real-world database systems. This assignment assumes the basic metadata and data management have been implemented in the first three programming assignments.

System Design

- You will decide how to guarantee the atomicity of transactions
    - In demo lecture: locking
    - As always, you are free (in fact, encouraged) to come up with your own design
- In the design document, you should clearly explain how your program implements the atomicity property of transactions. The easiest way to do this is to declare lock variables shared between processes. For example, you can create an empty file called "<table_name>_lock" to indicate an existing process is accessing <table_name>. In this case, we also assume the procedure of creating the empty file is atomic: At any time, only one process is able to create/delete the lock file (as an analogy, think of the P/V operations in an operating system). As a result, after a transaction starts, the system will first check the <table_name>_lock file before granting accesses to that specific table <table_name>. If there does exist such a lock file, then the system will reject/suspend the access request. Otherwise, everything works as before (e.g., updating tuples), except that *the change will not be persisted to the disk until a "commit" is encountered*.

Implementation

- The program should not use external database libraries, frameworks, or applications.
- Any programming language is acceptable, e.g., Python, Java, C/C++, Go
    - Just pick one(s) that you are most comfortable/proficient with
    - But keep in mind we will test your code in Linux with OS-level utilities (e.g., files)
        - So, probably not: C#, Object-C, JavaScript, Prolog…
- Functionalities:
    - SQL: Begin Transaction, Commit

Interface

- A similar but simpler interface than Sqlite3
- Same as homework #1: standard input, standard output (or files if you prefer)
    - This assignment the program will be tested using two (2) interactive terminals (to emulate two concurrent processes/clients).

Testing

- We will test your program on Ubuntu (version 14 or above)
- If your program cannot compile on our testbed, we may ask you to demo your program
    - Try not to use many exotic libraries.
    - The TA will probably not spend a whole day to setup an environment as yours.
- A full test script will be provided
    - We will NOT simply redirect the provided .sql file as standard input to your program
        - Because there will be two clients/processes/terminals
    - Please, carefully read the comments in the given .sql file, and understand what the expected output means
    - We will not to test your programs with any other scripts/commands
        - However, it's always good to consider more edge cases

Grading (20 points)

- This is an individual assignment
- Design document that clarifies the followings: (5 points)
    - At a very high level, how you implement different joins.
    - Be very specific on how to compile and execute your code
- Source code (15 points)
    - Coding style and clarity, 5 points
        - Appropriate parenthesis locations, indention, etc.
        - Always write comments at the beginning of any files
            - Author, date, history, etc.
        - Always write comments at the beginning of any non-trivial class/function
            - What this class/function/subroutine does, high-level algorithm if needed
        - Write in-line comments for non-trivial blocks/lines of code
    - Functionality, 10 points
        - Refer to the test script for detailed breakdowns

Submission

- WebCampus
- Compress all your source code and documents into one package in this format:
    - <your_netid>_pa4
- Late penalty: 10% per day