Summer 2023 CSE 480
Project 4: DBMS - TRANSACTIONS

June 1st, 2023

---

1 A Word of Warning

Make sure you read through the submission instructions thoroughly before you submit your project. Failure to comply with the submission instructions will result in a zero for this project.

This project builds directly off of Project 3. I also highly recommend that you comment your code thoroughly so you can remember how your code works in the future. As your project 5 can build off of this project.

2 Project Due Date

This project is due before June 8th at 11:59pm.

3 Project Overview

This project will emulate the behaviour of the built-in python module, "sqlite3". Your module will be able to execute SQL statements corresponding to: dropping tables, exceptions, and transaction.

4 What You are Given

On D2L we have released the test cases, and a script to run the test cases. We have also released two 'regression' tests, which you can use to determine if your updates for Project 4 have broken any Project 3 functionality. You are responsible for creating your own test cases as well to test the robustness of your solution. There are edge cases that we did not release test cases for, and it is up to you to consider all of these and make sure your solution will work in all conditions.

Please note that this cli.py script is different than the previous scripts, as it has to test concurrency in a deterministic way.

5 What You Need To Do

You need to build the functionality for Project 4 on top of your Project 3. I highly recommend that you make a copy of your Project 3 in case things go wrong.

All of the code for this project must be in 'project.py'. There is no need to write any database output to any files. Each test will be done on a fresh database.

Your program is allowed to use any built-in modules of python, with the exception of sqlite3. sqlite3 is the reference implementation for this project, your output will be compared against it for correctness. Importing the sqlite3 module in project.py will be considered academic dishonesty for these project, which will lead to an automatic failure of this course.

You cannot use any modules of python that are not built-in, meaning you cannot use things like Numpy or Pandas.

If you use any resources that did not come directly from the class, you must specify and cite them in your code file. At the top of the starter code there will be a section to paste links to anywhere you got code or ideas from. This is to make sure that you are not plagiarising.

5.1 Changes to the connect Function

In order to ensure that your solution matches the behavior of python's SQLite3 module, the connect function should accept a total of three parameters:

- 1. database: the filename of the database
- 2. timeout: your code should ignore this parameter
- 3. isolation_level: your code should ignore this parameter

These two new parameters are used to instruct sqlite3 to not do some of the fancy optimizations that your code won't be sophisticated enough to perform. Example call to connect:

```
conn = connect("test.db", timeout=0.1, isolation_level=None)
```

5.2 Raising Exceptions

There are times when your code should raise an exception (see https://docs.python.org/3/tutorial/errors.html). The type and message of the exception don't matter (and will never matter in any project). But, not raising an exception when expected will fail the test case.

6 Test Categories

6.1 Regression

These tests should pass if you completed Project 3. They should pass automatically at the start of the project. If these fail, you've made a regression (you have broken previously functional code).

6.2 Connections

These tests check that your code accepts multiple connections to the same database (only one database per test in this project).

6.3 Create_Drop_Table

You need to implement the CREATE and DROP table commands. The CREATE TABLE command you have already implemented, but there's one minor addition. Your code should raise an exception if a CREATE TABLE statement attempts to make a table that already exists.

If the CREATE TABLE command includes "IF NOT EXISTS" (example: CREATE TABLE IF NOT EXISTS students ...), then nothing should happen if the table already exists. You also need to implement the DROP TABLE statement (which has an optional "IF EXISTS" clause to not raise an error if the table doesn't exist). Examples:

DROP TABLE students;
DROP TABLE IF EXISTS students;

6.4 Transactions

- Transaction begin with a BEGIN TRANSACTION statement. Raise an exception if a BEGIN TRANSACTION occurs within an open transaction
- Transactions end with a COMMIT TRANSACTION (ROLLBACK tests are in a different category). Raise an exception, if a COMMIT TRANSACTION is attempted without a prior BEGIN TRANSACTION.
- The only statements in the tests that will happen within transaction are SELECT, INSERT, UPDATE, and DELETE (the Data Manipulation Language statements).
- If a statement is before a BEGIN statement or after a COMMIT statement, that statement is in "autocommit mode" meaning that any locks needed are acquired before the statement runs, and released when the statement is complete. This behavior is disabled after a BEGIN statement (manual transaction mode).
- A shared lock is needed to read (SELECT). Shared locks block exclusive locks.
- A reserved lock is needed to write (INSERT, UPDATE, and DELETE). Reserved locks block exclusive locks and other reserved locks.

- An exclusive lock is needed to commit a write. Specifically, if a transaction has a reserved lock, it must be promoted to an exclusive lock upon commit. Exclusive locks block all other locks.
- Locks are released upon commit.
- If a lock can't be granted when requested, raise an exception.
- Writes are only visible to other connection upon commit (before then the transactions need to keep private copies of their writes).

Remember: sqlite only locks the entire database, not individual tables or rows.

6.4.1 Recommended Implementation

The way we recommend you handle transactions is make a copy of the database's tables when the transactions starts and do your reads and writes to the copy. Upon commit, copy the modified tables to the "real" database. If you implement the locking correctly, no other transaction should generate conflicts.

It may seem inefficient to copy the entire database for each transaction, and it is. The way sqlite actually does it is keeping private copies of all the pages of memory it writes to (and only writing to disk on commit), but that is too difficult to do from Python. So do the inefficient, but easy thing for this project, copy all the tables for each transaction.

6.5 Transaction_Modes

Transactions can occur in one of three modes (BEGIN "mode" TRANSACTION):

- DEFFERED = locks are acquired when needed by a statement in the transactions
- IMMEDIATE = a reserved lock is acquired at start of the transaction and an exclusive lock is acquired when needed
- EXCLUSIVE = an exclusive lock is acquired at start of transaction

Transactions that don't specify a mode default to DEFERRED.

6.6 Rollback

The second way a transaction can end is with a rollback. All the changes made by the transaction needs to be reversed, and all locks released.

If you implement full isolation between transactions by only modifying a local copy of the database in each transaction, rollback is very simple, just don't copy your modified tables to the "real" database, just get rid of them.

You can only rollback within a manual transaction, if a connection tries to rollback outside of a transaction, raise an error.

7 Testing Your Code

7.1 Testing Yourself

We've provided a few sample sql transactions as well as a python script to test your code.

To run your code with the testing files provided:

python3 cli.py <filename>

Where the second argument is the .sql file to test. Each .sql file is a set of sql commands that compose one transaction. We also provide a way for you to compare against the ground truth, which is python's sqlite3 module. To run the ground truth, use the following command:

python3 cli.py <filename> --sqlite

Your output should be identical to the output from the sqlite3 module. In the cli.py file we have a few print statements so you know if you are using your own code or the ground truth sqlite3 module. You can remove this code if you like, in our final testing we will not have these print statements, of course.

You can, and should create your own test cases as well.

7.2 Instructor Testing

When we test your code, we will run it in almost the exact same way, just with more test cases.

8 Assumptions and Guarantees

All tests will be legal SQL (no syntax errors, inserting into nonexistent tables or data type violations).
All select statements will have "FROM" and "ORDER BY" clauses
In "CREATE TABLE" statements every attribute will have a data type and no constraints
All table and columns names will start with a letter (or underscore) and be followed by 0 or more letters or numbers or underscores

9 Submission Instructions

You will submit a file named "project.py" to the handin (handin.cse.msu.edu). All of your code should be in the "project.py" file. If you used any external resources (something other than class material), make sure that you cite the resource in a comment at the top of your "project.py" submission file. The top of the starter code also has space to put your name, netid, PID, and an estimation for how long the project took you to complete. Make sure you fill this out fully before submitting your project.