Project 2. Simple DBMS

SWE3003 Database Systems - Fall 2024

Due date: Dec 6 (Fri) 11:59pm

1 Purpose of this so-called 10,000 Line Project

The purpose of this project is to understand a large project code and make modifications and improvements to a part of it. When you join a company and get involved in its project, the first task you need to do is to understand the massive code of the company's project, and then you will implement some modules within it.

So, for this project, your need to get SimpleDB project, install it, and become comfortable with using it.

2 How to Setup

The project code is in `/home/swe3003/SimpleDB` in In-Ui-Ye-Ji cluster. You need to copy the entire directory to your working directory by running the following Linux command.

```
$ cp -R /home/swe3003/SimpleDB  ~/
```

For convenience, it is recommended to set the project directory as an environment variable. Set the environment variable in the `.bashrc` file as follows:

```
$ echo 'export SDBHOME=~/SimpleDB' >> ~/.bashrc
```

The source codes are in the `src/main/java` directory, which contains three sub-directories: `derbyclient`, `simpleclient`, and `simpledb`. `simpledb` directory has the source codes for DBMS. The simpledb directory has the sub-directories - buffer, file, etc, where source codes for each module of the DBMS are located.

`simpleclient` directory has client codes. Some client programs have hard-coded SQL statements, which you will find helpful for testing and debugging modifications to your database engine. Additionally, SimpleIJ.java is a client program for an interactive SQL shell. There are two types of clients for SimpleDB. The first group, located in the `embedded` directory, access databases directly without using JDBC. This is referred to as the *embedded mode*. The second group, located in the `network` directory, operates in a *server-client mode* where a client process communicates with a SimpleDB engine process using JDBC. To use this mode, you must first run the `simpledb.server.StartServer` program before running a client program. For more detailed instructions on how to use either mode, please continue reading.

To compile the server and clients, you need to run Maven commands as follows. Apache Maven is a popular build automation tool and project management tool for Java-based software projects, similar to Makefile or CMake for C/C++ projects. It provides a way to manage a project's build, dependencies, and documentation using `pom.xml` file. Please check the provided `pom.xml` file, if you are interested. But, note that you don't need to modify the `pom.xml` for this project.

```
$ cd $SDBHOME
$ mvn compile
```

After the compilation, class files will be created in `target/classes` directory.

3 Run the Embedded Client Programs

3.1 Create the student database in embedded mode

Please scan `CreateStudentDB.java` before you run. It is okay even if you don't fully understand it. This program creates a database named `studentdb`. Please make sure to be familiar with the database schema. To run CreateStudentDB, you need to run as follows.

```
$ cd $SDBHOME/target
$ java -cp ./classes simpleclient.embedded.CreateStudentDB
```

You should see a sub-directory for the studentdb database in the target directory. Feel free to examine its contents (but, DB files are binary :D).

Now, you should read `StudentMajor.java`. This program submits a SQL query. To run this program, you should run the following command.

```
$ java -cp ./classes simpleclient.embedded.StudentMajor
```

You will see 9 records showing the names of the students and their majors.

You also need to read `ChangeMajor.java`. This program submits a query to change the MajorId value of Amy's record in the STUDENT table. To execute this program, run the following command.

```
$ java -cp ./classes simpleclient.embedded.ChangeMajor
```

To check whether Amy's MajorID has been changed, you need to run the StudentMajor program again as follows.

```
$ java -cp ./classes simpleclient.embedded.StudentMajor
```

3.2 Delete the student database

To drop database tables, you need to delete files in the sub-directory `studentdb` as follows.

```
$ cd $SDBHOME/target/studentdb
$ rm -i *.tbl
```

Please BE CAREFUL when you run `rm` command. It may accidentally delete important files such as source codes. Consider using the `-i` option to prompt for confirmation before deleting any files. This simple step can prevent irreversible data loss.

4 Run the SimpleDB Engine as a Server

You may run SimpleDB in server-client mode. To run the server, run the following command.

```
$ cd $SDBHOME/target
$ java -cp ./classes simpledb.server.StartServer
```

You need to keep this server running while you run clients in server-client mode.

To kill the server, you need to press CTRL+C. Please make sure you kill the server when you don't use it. In-Ui-Ye-Ji is a shared cluster, and don't waste the computing resource for other students.

4.1 Run the Server-based Client Programs

Look at the source codes in `src/main.java/simpleclient/network` directory. While the server is running, you can run the CreateStudentDB and StudentMajor clients. It will produce the same outputs as when running in embedded mode.

5 Run the interactive shell - SimpleIJ

To run your own SQL statement in the interactive shell, you should run the following command.

```
$ cd $SDBHOME/target
$ java -cp ./classes simpleclient.SimpleIJ
```

In the `Connect>` prompt, enter "jdbc:simpledb:studentdb", which will establish a connection to the embedded database.

```
Connect>
jdbc:simpledb:studentdb

SQL>
```

The SQL shell will now repeatedly ask you to enter SQL queries, one per line. Type the following query, which should print the name and majorid of all students.

```
SQL> select sname, majorid from student
```

To terminate the shell, type “exit”.

```
SQL> exit
$
```

In the `Connect>` prompt, if you enter "jdbc:simpledb://localhost", it will establish a connection to the SimpleDB engine server, assuming the server is still running.

6 Task 1: Improving Buffer Manager

The SimpleDB buffer manager has two major inefficiencies:

- It uses the first unpinned buffer it finds when looking for a buffer to replace, instead of using a more intelligent method such as LRU.
- It performs a sequential scan of the buffers to check if a block is already in a buffer, instead of using a data structure (e.g. a map) to quickly locate the buffer.

To fix these problems, the BufferMgr class should be modified using the following strategy:

- Keep a list of unpinned buffers. When a replacement buffer is needed, remove the buffer at the head of the list and use it. When a buffer's pin count becomes 0, add it to the end of the list. This implements LRU replacement.
- Keep a map of allocated buffers, keyed on the block they contain. A buffer is allocated when its contents are not null, and may be pinned or unpinned. A buffer starts out unallocated; it becomes allocated when it is first assigned to a block and stays allocated forever after. Use this map to determine if a block is currently in a buffer. When a buffer is first allocated, it must be added to the map. When a buffer is replaced, the map must be changed—the mapping for the old block must be removed, and the mapping for the new block must be added.
- Get rid of the bufferpool array, as it is no longer needed.

In addition, the Buffer class should be modified so that each Buffer object knows its buffer ID. Specifically, its constructor should have a third argument denoting the buffer's ID, and a method getId() that returns that ID.

The BufferMgr class should also have a method printStatus that displays its current status. The status consists of the ID, block, and pinned status of each buffer in the allocated map, plus the IDs of each buffer in the unpinned list. Here is an example of the output that the method should produce for a database with four buffers, in which blocks 0 to 3 of file "test" were pinned, and then blocks 2 and 0 were unpinned:

```
Allocated Buffers:
Buffer 1: [file test, block 1] pinned
Buffer 0: [file test, block 0] unpinned
Buffer 3: [file test, block 3] pinned
Buffer 2: [file test, block 2] unpinned
Unpinned Buffers in LRU order: 2 0
```

Note that the buffers in the above output are in seemingly random order because they were retrieved from a hash map. The information within brackets comes from calling the `toString()` method of BlockId.

To help debug the code, a test program called TestBufMgr.java has been written. It pins and unpins buffers and occasionally calls the buffer manager's printStatus method.

7 Task 2: Wait-Die Scheme

SimpleDB currently uses timeout to detect deadlock. Change it so that it uses the wait-die strategy. Your code should modify the class `LockTable` as follows:

- The methods `sLock`, `xLock`, and `unLock` will need to take the transaction's id as an argument.
- The variable `locks` must be changed so that a block maps to a list of the transaction ids that hold a lock on the block (instead of just an integer). Use a negative transaction id to denote an exclusive lock.
- Each time through the while loop in `sLock` and `xLock`, check to see if the thread needs to be aborted (that is, if there is a transaction on the list that is older than the current transaction). If so, then the code should throw a `LockAbortException`.
- You will also need to make trivial modifications to the classes `Transaction` and `ConcurrencyMgr` so that the transaction id gets passed to the lock manager methods.

Test program creates three transactions. Transactions A and C both need a lock held by transaction B. Under the wait-die scheme, transaction A should wait, whereas transaction C should throw an exception.

When you run the Test program with correct implementation, you will get the following output:

```
new transaction: 1
Transaction A starts
Tx A: request slock block 1
Tx A: receive slock block 1
new transaction: 2
Transaction B starts
Tx B: request xlock block 2
Tx B: receive xlock block 2
new transaction: 3
Transaction C starts
Tx C: request xlock block 1
Transaction C aborts
transaction 3 rolled back
Tx A: request slock block 2
Tx B: request slock block 1
Tx B: receive slock block 1
transaction 2 committed
Transaction B commits
Tx A: receive slock block 2
transaction 1 committed
Transaction A commits
```

8 Deliverables

Source codes only

9 How to Submit

Please submit your source code files in swin.skku.edu using db_submit command as follows.

```
$ db_submit term2  /your/code/directory/path
```

This command will compress and submit all files in your current directory. For example, if your codes are in the current directory, run the following command.

```
$ db_submit term2  ./
```

Note that you can submit multiple times. But only the last submission will be graded. Using the following command, you can check whether your file has been correctly submitted.

```
$ db_check_submission term2
```

10 Late Submission Policy

20% will be deducted for every 24 hours.

11 Q&A Piazza

For any questions, please post them in Piazza so that we can share your questions and answers with other students and TAs. Please feel free to raise any issues and post any questions. Also, if you can answer other students' questions, feel free to do so.