

Yoonsung Nam

CS 457 Project 4

5/10/2022

## **Programming Assignment 4: Design Document**

### **Abstract:**

This document will go into detail how exactly the program for this assignment functions. It will discuss, on a very high level, how the update transaction between the two running processes work with each other. The document will also clarify how exactly the code should be compiled and executed.

### **How does the program manage and implement the transactions?**

Following the given SQL file, the queries present require the program to execute commands and actions such as creating a database, inserting info, and creating tables that were also present in past projects. Thus, this project will use programming assignment 1's code as a base template.

The thing that really separates this assignment is the inclusion of the "begin transaction", "update", and "commit" commands. To execute the proper functions based on these commands, the program will wait for user input from the terminal and will call the correct function based on what the user input command was. For instance, if the user input had the words, "create database", in them, the function will call the create database function. To also achieve this with the "begin transaction", "update", and "commit" commands, new elif cases were added the main function to call the functions tied to these keywords.

The update function serves to update the currently existing status values tied to the seats in the tables. So, if the user input set the status to 1 at seat 22 with the update command, the program will first read through the existing Flights file, which held the pre-updated information, and will store that read information in parsed out variables. For example, different variables will hold different information in the read file such as the status. Then, the update function will change these values to the updated version. For instance, the status value of 0 for seat 22 in the original version will be changed to 1 with the update command. After this is changed, all the information will be put together under one string.

This one string will however, NOT be written back into the original Flights text file. Instead, the string with the updated values will be written and stored in a temporary file. The program does in to account for the commit and transaction between process 1 and process 2. If the Flights text file's contents were changed before the commit command was called, that would not be proper locking of the transaction to one process. Process 2 would just be able to access the updated version even though only the update command was given by the user and the not the commit. Thus, the temporary text file will act as a placeholder for the updated information until the commit command is given by the user.

When the commit command is given by the user, the function\_commit function will run. In this function, the temporary text file which hold the updated information will be read, and the read values will now be

written back into the Flights test file. Now, only after this is complete can other processes, such as process 2 access the updated information in Flights.

It is important to also discuss the lockProcess function which served to act when the “begin transaction;” command was given by the user. When this function runs, it will serve as a sort of key for the update function to write the updated information to the necessary file. This is because the program is designed so that the update function will only be able to write to the temporary file if the begin transaction function is first called. This is done by the global variable, transaction, and when the lockProcess function runs, the variable will be set to True, and only when it is true, can the update function write to the temporary file. Furthermore, this function also has the logic to determine if any other processes are trying to call the update function when it has already been called by process 1. With this, the program will output an error message when process 2 tries to update the status when process 1 has not committed the updated information yet.

## **How should the program be compiled and run?**

The method to running and testing this program will be different than in previous programming assignments.

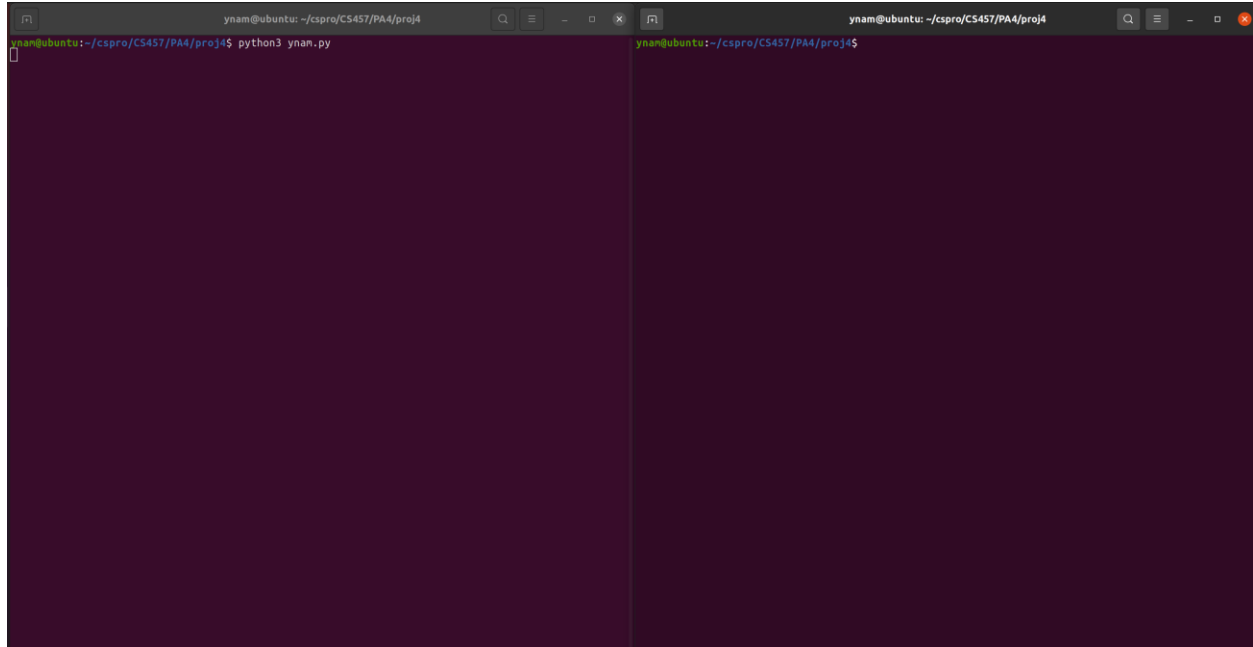
First, two terminals will be opened being in the same directory as the program.py. These terminals will represent process 1 and process 2. Now we will be running the program with the command “python3 ynam.py”, and this time, we will not be adding in the given SQL file to be read from. Instead, we will be inputting in the given commands from the SQL file directly into the terminal line by line. It is important to input each line (command) from the SQL file manually into the terminals.

We will then input in the commands in the first P1 block of commands. After all the commands from the first P1 block of commands go into the terminal for process 1, we will go to the other terminal, which represents process 2, to input in the commands from the first P2 block of commands. We will do this again for the second P1 block of commands, this time in the process 1 terminal once more, and so on. We will be going back and forth between the two terminals to input in these commands.

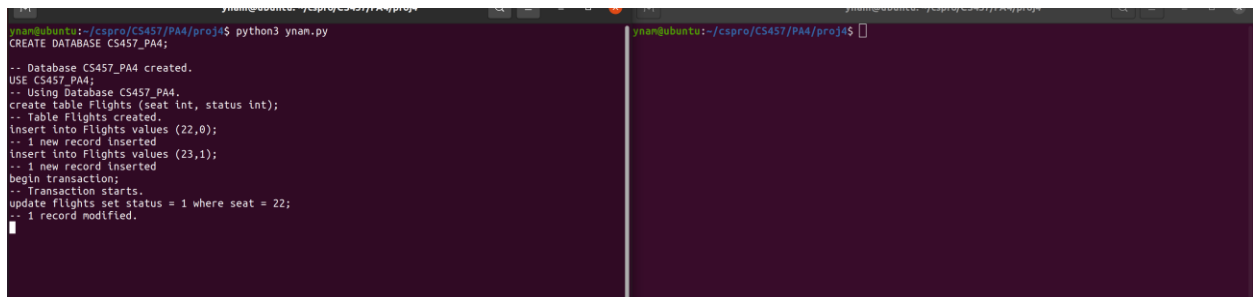
The corresponding output to each of the commands will be printed after each command is given. An example demo of the code will be given down below.

## Example demo of the code

We will first make two terminals from the same directory as where the code is.

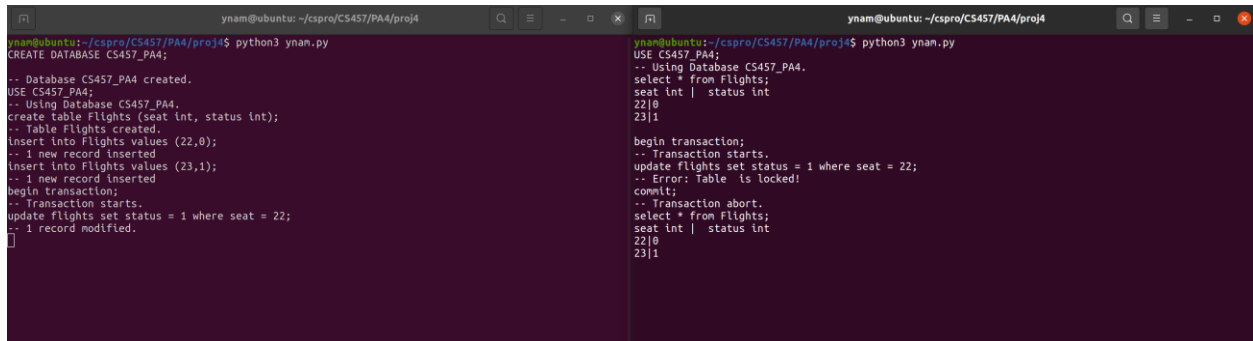


The left terminal will be process 1 and the right side will be process 2. We will first run process 1.



The first P1 block of commands is ran, each command is given line by line and the corresponding output is given.

Next, we will run the first P2 block of commands.

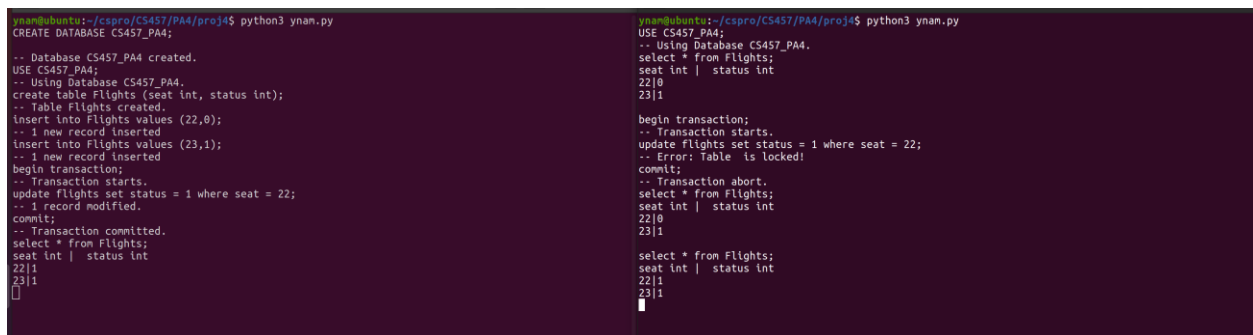


```
ynam@ubuntu:~/cspro/CS457/PA4/proj4$ python3 ynan.py
CREATE DATABASE CS457_PA4;
-- Database CS457_PA4 created.
USE CS457_PA4;
-- Using Database CS457_PA4.
create table Flights (seat int, status int);
-- Table Flights created.
insert into Flights values (22,0);
-- 1 new record inserted
insert into Flights values (23,1);
-- 1 new record inserted
begin transaction;
-- Transaction starts.
update flights set status = 1 where seat = 22;
-- 1 record modified.
[

ynam@ubuntu:~/cspro/CS457/PA4/proj4$ python3 ynan.py
USE CS457_PA4;
-- Using Database CS457_PA4.
select * from Flights;
seat int | status int
22|0
23|1
begin transaction;
-- Transaction starts.
update flights set status = 1 where seat = 22;
-- Error: Table is locked!
commit;
-- Transaction abort.
select * from Flights;
seat int | status int
22|0
23|1
```

On the right side, the first block of P2 commands is ran, it also outputs after each command.

Next, we will run the second block of P1 commands and the second block of P2 commands.



```
ynam@ubuntu:~/cspro/CS457/PA4/proj4$ python3 ynan.py
CREATE DATABASE CS457_PA4;
-- Database CS457_PA4 created.
USE CS457_PA4;
-- Using Database CS457_PA4.
create table Flights (seat int, status int);
-- Table Flights created.
insert into Flights values (22,0);
-- 1 new record inserted
insert into Flights values (23,1);
-- 1 new record inserted
begin transaction;
-- Transaction starts.
update flights set status = 1 where seat = 22;
-- 1 record modified.
commit;
-- Transaction committed.
select * from Flights;
seat int | status int
22|1
23|1
[

ynam@ubuntu:~/cspro/CS457/PA4/proj4$ python3 ynan.py
USE CS457_PA4;
-- Using Database CS457_PA4.
select * from Flights;
seat int | status int
22|0
23|1
begin transaction;
-- Transaction starts.
update flights set status = 1 where seat = 22;
-- Error: Table is locked!
commit;
-- Transaction abort.
select * from Flights;
seat int | status int
22|0
23|1
select * from Flights;
seat int | status int
22|1
23|1
```

Basically, this is the case when both of the processes are finished executing their commands.

This is also the content present inside of the Flight file after all the commits are done.



```
Open [icon] Flights
~/cspro/CS457/PA4/proj4/CS457_PA4
1 seat int | status int
2 22|1
3 23|1
```