COSC3380: Processing Transactions
Ticket Reservations in an Airline

Instructor: Carlos Ordonez

1 Introduction

You will develop a program that combines SQL statements and a host language to process transactions concurrently (several transactions at the same time).

The input text file includes all the information needed to reserve tickets, your program will produce a sequence of transactions and send them to the DBMS by several threads to update several tables. The input tables will be ready to be inserted and updated in the database. The program will provide two options: without and with transactions. Processing without transactions is expected to produce inconsistent results. On the other hand, processing with transactions is expected to produce consistent output.

2 Program requirements

1. Execution:

   Your program will generate and send SQL statements to the DBMS.

2. With transactions:

   Transactions should update several tables as seen in class. You program should be prepared to handle any errors.

3. Without transaction:

   you can send each statement in isolation, relying on the DBMS automated locking mechanisms. You do not have to handle any errors, letting execution proceed.

4. Languages and OS.

   The OS for our DBMS and for testing is Linux (any distro is fine). However, it is feasible to develop programs in Windows and MacOs and later test them in Linux. The programming language is: Python The connection library is psycopg2. Java or JavaScript require instructor permission and require JDBC, a different and somewhat more complicated library.The DBMS is: PostgreSQL.

5. You cannot export or read tables row by row, doing processing outside the DBMS. You cannot do process data outside the DBMS with the host language low-level processing mechanisms (i.e. locking, text files). The bulk of processing must be done with SQL statements, with high-level control in the host language. The only exception are threads.

2.1 Programming details

1. You need to program transactions to book ticket reservations assuming many customers try to book the same flight. In the worst case the flight gets sold out and reservation is declined.

2. You need to combine the information given in the text file and records in the tables to finish the reservation.

3. The program is a multi-threaded program. Each thread reads from a input file, do the reservation process. We will proved a sample to show how to do multi-threading programming in Python.

4. Tables:

   List of tables that will be updated (INSERT or UPDATE in SQL): bookings, ticket, ticket_flights, and flights

5. Initial database state:

   zero reservations, flights ready to be booked, flights available for 3 months in the future, all tickets have same price and same fare condition (economy).

6. We added two new columns in the flights table, seats_booked and seats_available. So we ignore the seat number for each customer, we only care the total number of seats that are available or booked. Thus, you do not need the boarding_passes table.

7. When start a new reservation, you should first generate a book_ref number, update the table bookings. Then, generate a ticket number and insert a new record in the ticket table and ticket_flights table if there are available seats in the specified flight and the date. And also update seats_booked and seats_available column in the flights table. The reservation is successful. If there is no available seats, only a book_ref number is generated and the the table bookings is updated.

8. The input file have 1 reservation per line. However, one customer may reserve multiple tickets.

9. Note that there might be several threads try to update the same table and the same column(for example, the seats_available column in the flights table).

10. Your program must be prepared to be stopped by the TA at any moment, closing the terminal without warning (killing the process, control C, etc).

11. Your program should generate a plain SQL script file (transaction-bookings.sql) for all the queries used in your program (correctly formatted as shown in the textbook and seen in class, following SQL format/indentation standards).

12. You should not create any additional tables. Any query should be self-contained.

13. Failed transactions: (1) When customer id is invalid (2) When flight id is invalid.

    Unsuccessful transactions: When the flight is full, but "book_ref" is still issued.

    Successful transactions: When the passenger get a successful reservation.

14. You need to generate unique "book_ref" and "ticket_no" for each reservation. You can do this using locking or no-locking. Anything is okay.

15. Python: You do not have to worry about Python speed or memory usage since the "heavy OLTP" is done by the DBMS.

16. SQL:

    Create ”transaction-bookings.sql” with the main transaction SQL code.

    SQL keywords are not case sensitive. Therefore, convert all your input SQL query, tables names, and so on to either lower or uppercase. It is preferred SQL keywords are in uppercase and table/column names in lowercase.Notice strings (char) are case sensitive.

3 Program call and output specification

Here is a specification of input/output. There are the input parameters: input file name, transaction=y/n, and # of threads.

- syntax for Python call:
  $python3\ transaction\text{-}bookings.py$ “$input = trans.txt; transaction = [y|n]; threads =< int >$”.

  Typical test: python3 transaction-bookings.py “input=trans.txt;transaction=y;threads=10”.

- If the connection to the DBMS fails, the program must display an error message.

- Input text file

  Each line in the text file contains passenger_id and flight_id, other information such as email and phone will be null values to simplify the reservation. A sample of the input text file is given below:

```
trans.txt
==================
passenger_id,flight_id
8559460462,1001
7339561341,1005
=================
```

- Input database/tables checking: Since you will work on a fixed database it is assumed tables and columns are OK. Therefore, it is expected the SQL statements in your transaction are correct as well.

- Output:

  # of records updated per table. # of successful transactions. # of unsuccessful transactions. The precise output format will be specified by the TAs, but correctness will be checked based on table contents.

4 Grading

Correctness: In this case there is not a unique database state. Column values may differ due to differences in transaction schedules, random choice of flight and the logic to handle any potential errors. The theory explanation is in the textbook.

Testing: The TAs will check the updated tables with SQL queries to check ACID properties, especially consistency across all updated tables. Again, the main goal is to observe the impact of transactions in correct output. The TAs will manually check the SQL code behind each transaction, which in general is more concise than SELECT statements in queries (say 10-20 lines).

Sizes: You can expect tables to grow to no more than 10000 rows and to have no more than 1000 updated rows, to make SQL development faster and easier to check.

Score: A program not uploaded to the UH linux server will get 0 (zero, the TAs have no obligation to run testing outside this server). A program with syntax or data type errors will get 10 (either SQL or Python). A working program, producing correct for some cases, will get a score 30 or better. The TAs do not have the duty to try to make the program run if it fails, unless this document has a wrong specification. Any resubmission will have a 20% penalty for regrading, no matter how small the fix is.

Code originality: The Python code and SQL statements will be checked for plagiarism with source code analysis tools (we cannot disclose them): any significant similarity with some other student code will be reported to the instructor. Notice reformatting code, changing variable names, changing order of functions, changing for loops for while loops, repackaging into different files, adding spurious variables; all of them can be easily detected since the ”logic” solving the problem is the same. Warning: copying/pasting code from the Internet (stack overflow, github, tutorials) may trigger a red flag: the instructor will decide if there is cheating or not.