COURSEWORK ASSIGNMENT **UNIVERSITY OF EAST ANGLIA**

School of Computing Sciences

Module: CMP-7025A Database Manipulation

ASSIGNMENT TITLE: Database application

DATE SET	:	see attached
DATE & TIME OF SUBMISSION	:	see attached
RETURN DATE	:	see attached
ASSIGNMENT VALUE	:	see attached
SET BY	:	**Jeannette Chin**
SIGNED	:	
CHECKED BY	:	**Dr Jaejoon Lee**
SIGNED	:	**Jaejoon Lee**

Aim:

Assess knowledge of the following learning outcomes:

1. write SQL queries and apply transaction processing techniques for ensuring consistency of the database
2. write a simple application program to interact with a database server and perform database manipulation

Assessment criteria

Appropriate use of SQL to complete design.

Appropriate use of SQL statements to implement specified tasks.

Correct style of Python programming.

SQL and Python program testing.

Readiness to demonstrate.

Submission of requested documentation.

Description of assignment:

See attached

Required:

See attached

Handing in procedure:

Please submit your piece of coursework electronically following instructions posted on Blackboard. Submissions will be time stamped by the system. Any late submission (after 3 pm on the day of submission) will be penalised.

Plagiarism:

Plagiarism is the copying or close paraphrasing of published or unpublished work, including the work of another student without the use of quotation marks and due acknowledgement. Plagiarism is regarded a serious offence by the University and all cases will be reported to the Board of Examiners. Work that contains even small fragments of plagiarised material will be penalised.

UNIVERSITY OF EAST ANGLIA

School of Computing Sciences

CMP-7025A Database Manipulation

Database application

Set on	**: 20th September 2020**
To be completed by	**: Wednesday 16th December 2020, 3 pm**
Returned by	**: Friday 15th January 2021**
Value	**: 60%**

1. Introduction

The Pierian Games are a four-yearly sports competition open to all the countries of the world. The games comprise a number of *events*. The events are held at specific locations, dates and times determined by the games organisers. *Spectators* are issued *tickets* to attend events. A spectator can have only one ticket for any given event and they must have been issued a ticket before travelling to the event. Occasionally, events are moved to a different location, date or time, in which case the event is *cancelled* and all the tickets for the event are cancelled. Spectators may then be issued new tickets for the re-arranged event but this is outside the scope of this exercise. Details of events, spectators, valid tickets and cancelled tickets are recorded in a database. The database is used to provide up-to-date information to spectators and to the organisers of the games. This information is accessed over the Internet and by using dedicated workstations.

Data to be held in the database and queries concerning tickets come from various workstations or active web pages. The IT team for the Pierian Games refer to these various inserts, deletes and queries as *tasks*. Their central database is held on a server computer that processes the tasks. Tasks are held in an input queue and processed by an application program on the server. The results of processing the tasks are placed in an output queue for despatch to workstations or web pages.

For this coursework exercise you are required to set up a database for the Pierian Games, write SQL statements for the tasks and develop a Python application program for running the tasks through the Python psycopg2 interface. Naturally, this exercise is greatly simplified compared to a real application. A detailed specification of the work to be undertaken and the deliverables to be produced for assessment is given below.

You will set up your 'Pierian Games database' as a set of tables. You are required to write the server application program and *not* the workstation software or the web pages. For this exercise you will simulate the queue of input tasks by reading a stream of data from a text file. Similarly, the results of the tasks will be sent to a text file that simulates the output queue. Thus, no GUI user interface programming is required.

You may, of course, use your own facilities to develop your program *but the final version must use PostgreSQL ~~and run in the CMP labs.~~*

2. Application Program Functionality

2.1 The database comprises the following tables with precisely the given, named fields:

```
event (ecode, edesc, elocation, edate, etime, emax)
spectator (sno, sname, semail)
ticket (tno, ecode, sno)
cancel (tno, ecode, sno, cdate, cuser)
```

Notes:

The `event` table holds details of each event which is scheduled for the games.
The `spectator` table holds details of spectators who have been issued tickets.
The `ticket` table holds details of tickets issued to spectators for events.
The `cancel` table is used to record details of all tickets that have been cancelled.

`ecode` is a four-character code identifying an event, e.g. A100 for the athletics event run over 100 metres.
`edesc` is a description of the event.
`elocation` is the place where the event is held.
`edate` is the date on which the event is held. Each event takes place on a single day. All events are scheduled for the month of April 2019.
`etime` is the start time of the event. No events start before 09:00 hours.
`emax` is the maximum number of tickets that can be sold for an event. Ranges from 1 to 1000.
`sno` is a numeric identifier given to a spectator.
`sname` is the name of a spectator.
`semail` is the email address of a spectator.
`tno` is a reference number for a ticket issued to a spectator for an event. The reference numbers start at 1 and are incremented by 1 each time a new ticket is sold. You should increment the `tno` manually rather than use an automatic number generator as that would make testing more difficult.
`cdate` is a timestamp showing when the cancellation of a ticket takes place.
`cuser` is the user id of the person responsible for a cancellation of a ticket being recorded in the `cancel` table.

2.2 The tasks of interest to us in this exercise are:

A. Insert a new spectator.

B. Insert a new event.

C. Delete a spectator. To be deleted, the spectator must not have any valid (i.e. not cancelled) tickets.

D. Delete an event. All the tickets for the event must be cancelled before an event can be deleted.

E. Issue a ticket for an event. A spectator may have only one ticket for a given event.

P. Produce a visual table showing the total number of spectators liable to travel to a location[1]. The table should show the total number of spectators that could travel to a location on each date an event is held at a location.

[1] Here the use of the term 'table' is to indicate the general format of the result of a database query. It does not imply that a new base table is to be created in the database. It means that the result of the query can be in the

Q. Produce a visual table showing the total number of tickets issued for each event. Present the data in order of event description.

R. As Q above but only for a given event which is specified by the event code.

S. Produce a table showing the itinerary for a given spectator. The spectator is specified by their spectator number. The itinerary should contain the spectator's name and the date, location, time and event description of each event for which the spectator has been issued a ticket.

T. Given a specific ticket reference number, display the name of the spectator and the event code for the ticket and indicate if the ticket is valid or is cancelled.

V. View the details of all cancelled tickets for a specific event.

X. A task code sent to close down the server application program.

Z. A task code sent to the server application program to empty the database tables prior to use with live data.[2]

2.3 Note that, for each task, the program should send a reply to the output queue (a text file). For example, tasks A to E, X and Z should, if successful, send a simple message showing the task type (A, B, …) and confirming that it has been completed. For example:
`'X. Pierian Games application closing down'.`

The output 'tables' from tasks P, Q, R S and T are sent to the output queue and comprise the task type followed by the relevant data in a simple tabular format. You are not expected to implement report headers, report footers, page breaks, control breaks, fancy fonts, colour, logos and so on, as might be found in a 'real' output.

A task that contains errors or would cause errors should send its task type and a user-friendly error message to the output queue.

3. Assignment Tasks

3.1. Database definition and loading

Prepare and test an SQL script to create your copy of the database. This should take, as a starting point, the table definitions given in Appendix A2. Prepare additional SQL clauses and/or statements to complete the definition of the database by specifying primary keys, domain constraints, entity and referential integrity constraints, etc. **Note that you should NOT modify the name and type of the attributes** (i.e. the information you have been given). Save all your Data Definition Language (DDL) statements in a text file.

Load a reasonable volume of test data (e.g. 5 - 10 rows) into the tables for use in your testing. The test data should be sufficient to test all of the queries with their expected output and should provide a suitable environment in which to test normal operation as well as abnormal conditions.

- Document this stage with a copy of your complete DDL statements in SQL (including any table definitions, views, triggers, comments, etc.)
- Also produce a copy of the test data you loaded for testing in a text file.

normal tabular format which in turn may have some typographical style imposed before display at a workstation or on a web page.

[2] Actually to help in repeated testing!

3.2. Interactive SQL version of the tasks

Prepare and test interactive SQL statements for the various types of tasks defined in section 2.2 above. Test these statements using the SQL editor in PostgreSQL.

The purpose is to test your SQL statements before using them in your Python program. You may need more than one execution of some of the task types to demonstrate the correctness of your work. Your .sql files need to be ready to be loaded during the demo so if you fail to demonstrate your Python program working you can at least demonstrate your prototype SQL statements through the interactive interface.

- Document this stage with a copy of the SQL statements. Your SQL statements need to be accessible as text files
- Evidence of testing of each SQL statement (e.g. copy of the output from running the query).

3.3. Python application programming tasks

Use suitable versions of your SQL statements produced for section 3.2 within a Python application program using the Python psycopg2 interface. The program should read a queue of task details from a text file and send the results and/or error messages to the output queue (another text file). The precise input format of tasks is described in appendix A1 below.

Your program does not need to check that the values of individual fields in tasks are of the correct type and format. However, the program must make checks against the database to ensure that task details are valid. For example, an event code field used in a task type E will be a four-character alphanumeric string but there may be no such actual event stored in the database. Where possible, the program should handle errors by reporting them and then continue with the next task.

The *deliverables* for this part of the assignment are:

- A commented program source listing (your .py file/s) which demonstrate the use of SQL in Python

- Copies of input and output files used to show that the program works as specified.

The objective is to demonstrate the good use of SQL and database features and *not* complex Python coding. *Source programs should be consistently laid out in a format which is easy to read.*

3.4 Off Line Demonstrations

All marks will be awarded according to the demonstration of your working code. However, demo marks will be penalised and may even go down to zero if the supportive evidence is not submitted as requested, is incomplete or if it is submitted late. You will only receive marks for what is submitted. In the case of SQL statements, you need to submit the statement as well as evidence that you tested them as instructed (by showing the output of execution from the pgAdmin 4 interface). The off line demonstration will involve loading a set of provided test data and carrying out a standard set of tests. The SQL and program demonstrated must match the version submitted electronically.

Important Notes

- No additional report is required.
- The electronic documents should be well presented to aid checking.
- This is an individual piece of coursework NOT a group project. Collusion checks may be carried out on the electronic submissions.

- As you will be given a SQL script on Friday week 11 to load test data into your tables, it is vital that you do not change the table names, field names, field types etc.

Summary of Deliverables

- Electronic submission to Blackboard (All files to be collected in a folder named with your student number and zipped):
  - Part 1: A copy of your SQL data definition statements, annotated as necessary, together with the test data.
  - Part 2: Your SQL data manipulation statements (annotated with comments if necessary) for each of the requirements submitted electronically together with evidence of testing as instructed in the OffLineDemo.docx document (will be provided on Friday week 11).
  - Part 3: Your Python source code files (.py) and input and output files.

Appendix

A1. Task formats

Each task comprises one line, each line being terminated by an end-of-line character. The line always comprises a single character giving the task type. Where the task type requires more details to be input, the input is separated by a white space ‘ ’ following the transaction type on the same line. If there is more than one input, then the following inputs are separated by a comma ‘,’. The text file simulating the input queue will comprise a series of such tasks terminating with a type X task.

Task Type	**Fields**	**Example**
Insert spectator (A)	spectator number, spectator name, email address	A 100, A. N. Other, ano@somewhere.net
Insert event (B)	event code, event description, location, date, time, maximum number of tickets	B A100,100 metres sprint,Stadium 1,2016-04-01,14:00,1000
Delete spectator (C)	spectator number	C 100
Delete event (D)	event code	D A100
Issue ticket (E)	event code, spectator number	E 1, A100,100[3]
Travel query (P)	(none)	P
Total tickets query (Q)	(none)	Q
Total tickets for an event (R)	event code	R A100
Itinerary for spectator (S)	spectator number	S 100
Ticket details query (T)	ticket number	T 1234
View cancelled tickets for event (V)	event code	V A100
End of requests (X)	(none)	X
Empty database tables (Z)	(none)	Z

[3] Note that the ticket reference number is **not** part of the input data; it is calculated by the program.

A2. Minimal database definition[4]

The following SQL script provides a minimal definition of the database for use in part 3.1. A copy of this text can be found in the file `Cw_Schema.txt` in Blackboard. To do part 3.1 of the assignment you will probably want to *add* SQL clauses or statements. However, to enable my testing of your program during the demonstration, you must use the same tables, columns and column datatypes in your version.

```
CREATE TABLE event (
    ecode         CHAR(4),
    edesc         VARCHAR(20),
    elocation     VARCHAR(20),
    edate         DATE,
    etime         TIME,
    emax          SMALLINT );

CREATE TABLE spectator (
    sno           INTEGER,
    sname         VARCHAR(20),
    semail        VARCHAR(20) );

CREATE TABLE ticket (
    tno           INTEGER,
    ecode         CHAR(4),
    sno           INTEGER );

CREATE TABLE cancel (
    tno           INTEGER,
    ecode         CHAR(4),
    sno           INTEGER,
    cdate         TIMESTAMP,
    cuser         VARCHAR(128) );
```

A3. Help!

If you do not know where to start in the Python programming part, there is a 'starter program' which might help. This is on Blackboard as `starter.py` This program reads a text file, calls a method and writes to a text file. It does not do any database processing. The method called is just a 'stub' but the program gives a possible working framework to start your own development.

[4] Of course, the number of columns and their lengths is not realistic. The objective is to have enough data to show the principles of database programming and not to show a realistic design for a set of sporting events.