Assignment 4

Due February 21, 2018 by 11:45pm

In this assignment, you will be required to use PostgreSQL. Your solutions should include the PostgreSQL statements for solving the problems, as well as the results of running these statements. Turn in a file .sql with your solutions (where necessary, include comments explaining your solutions). Turn in a file .txt which illustrates how your solutions work. Consider a database with the following relations:

Student(sid int, sname text, major text)	sid is primary key
Course(cno int, cname text, total int, max int)	cno is primary key total is the number of students enrolled in course cno max is the maximum permitted enrollment for course cno
Prerequisite(cno int, prereq int)	**meaning**: course cno has as a prerequisite course prereq cno is foreign key referencing Course prereq is foreign key referencing Course
HasTaken(sid int,cno int)	**meaning**: student sid has taken course cno in the past sid foreign key referencing Student cno foreign key referencing Course
Enroll(sid int,cno int)	**meaning**: student sid is currently enrolled in course cno sid foreign key referencing Student cno foreign key referencing Course
Waitlist(sid int,cno int, position int)	**meaning**: student sid is on the waitlist to enroll in cno, where pos is the relative position of student sid on the waitlist to enroll in course cno sid foreign key referencing Student cno foreign key referencing Course

1. (a) Notice the primary key constraints specified for these relations. Write insert triggers to enforce these primary constraints.

   (b) Notice the foreign key constraints specified for these relations. Write insert and delete triggers on the appropriate relations to enforce these foreign key constraints. Furthermore, write delete triggers on the `Student` and `Course` relations that have cascading deletion effects on the relations that hold foreign keys that reference the primary keys of these relations.

   (c) The `HasTaken` relation is special in the sense that only inserts are permitted on it. `HasTaken` serves as a historical record of courses that students have taken in the past. Furthermore, we require that no new inserts are permitted on `HasTaken` from the moment that students start getting enrolled in the `Enroll` relation. Write trigger on HasTaken to enforce these requirements.

(d) The `Course` and `Prerequisite` relations are special in the sense that only inserts are permitted on them. In addition, we assume that `Prerequisite` and `Course` can no longer change from the moment that enrollment records get inserted or deleted in the `Enroll` relation. In particular, no new courses and/or no new prerequisites for courses can be added or updated from that point on. The idea behind these requirements is that course-data is very rigid temporarily relative to students enrolling in such courses. Write triggers to enforce these requirements.

2. Furthermore (1) inserts and deletes in the Enroll relation and (2) insert and deletes in the Waitlist are governed by the following rules:

   - A student can only enroll in a course if he or she has taken all the prerequisites for that course. If the enrollment succeeds, the total enrollment for that course needs to be incremented by 1.
   - A student can only enroll in a course if his or her enrollment does not exceed the maximum enrollment for that course. However, the student must then be placed at the next available position on the waitlist for that course.
   - A student can drop a course, either by removing him or herself from the Waitlist relation or from the Enroll relation. When the latter happens and if there are students on the waitlist for that course, then the student who is at the first position for that course on the waitlist gets enrolled in that course and removed from the waitlist. If there are no students on the waitlist, then the total enrollment for that course needs to decrease by 1.

   Write appropriate triggers to enforce these rules.

3. Consider a situation wherein we repeatedly, and randomly, throw 3 dice d_1, d_2, d_3. Clearly, the possible value for each such dice d_i is between 1 and 6.

   Define the random variable T which, when applied to a throw of 3 dice d_1, d_2, and d_3, returns the sum of the 3 dice values. I.e.,

   $$T(d_1, d_2, d_3) = d_1 + d_2 + d_3.$$

   Notice that the range of values for T is $[3, 18]$. For example $T(2, 4, 5) = T(3, 2, 6) = 11$, $T(1, 1, 6) = 1 + 1 + 6 = 8$, $T(1, 1, 1) = 3$, etc.

   The objective of this question is to compute, in an **incremental way**, an approximation of the *expected value* $\mathbf{E}(T)$ of the random variable T, as well as an approximation of the *standard deviation* $\mathbf{V}(T)$ of the random

variable T. By definition, the exact values for $\mathbf{E}(T)$ and $\mathbf{V}(T)$ are as follows:

$$\mathbf{E}(T) = \sum_{v=3}^{18} v \cdot p(T = v)$$

and

$$\mathbf{V}(T) = \sqrt{\sum_{v=3}^{18} v^2 \cdot p(T = v) - \mathbf{E}(T)^2}$$

where $p(T = v)$ is the probability that the outcome of T is the value v. Incidentally, the theoretically correct value for $\mathbf{E}(T)$ is 10.5 and that for $\mathbf{V}(T)$ is $\sqrt{8.75} = 2.95803\cdots$.

Your solution requires the use of a trigger which maintains an approximation of $\mathbf{E}(T)$ and $\mathbf{V}(T)$. More specifically, each time a throw of 3 dice is made, these approximations need to be refined in accordance with the outcome of T for this throw. Furthermore, and this needs to be a core part of your solution, the approximations are required to be maintained in an incremental fashion. In particular, you can not store the set of all throws and from that set find the approximations of $\mathbf{E}(T)$ and $\mathbf{V}(T)$.

To obtain a random value for a single dice, you can use the function call

`floor(random()*6)+1`

This works since the function call `random()` produces a random value in the range $[0, 1)$. Therefore a throw of three dices is a tuple of the form

(`floor(random()*6)+1`, `floor(random()*6)+1`, `floor(random()*6)+1`).

You need to experiment with different numbers of throws. To do this, use the following function template:

```
create or replace function runExperiment(n int)
returns record
as
$$
declare i int;

begin
   -- code to initialize the experiment;

for i in 1..n loop
    -- code to call a trigger on
    -- throw (floor(random()*6)+1, floor(random()*6)+1, floor(random()*6)+1);
```

```
end loop;

return -- (approximation for E(T), approximation for V(T));
end;
$$ language plpgsql;
```

Then execute for example

```
select runExperiment(1000);
```

This will be an experiment with `n` = 1000 throws of 3 dice each.

The following is a table with some values obtained for $\mathbf{E}(T)$ and $\mathbf{V}(T)$ for different values of `n`:

`n`	**E(T)**	**V(T)**
10	10.6	3.16859
100	10.24	2.80756
1000	10.419	3.06356
10000	10.505	2.94929
100000	10.490	2.95849