B561 Advanced Database Concepts

Assignment 4 (Triggers)

Consider a database that maintains the relations `Person`, `Company`, `worksFor`, and `Knows`. The schemas for these relations are as follows (primary keys are underlined):

```
Person(pid : integer, name:text)
Company(cname : text, city : text)
worksFor(pid : integer, cname:text)
Knows(pid1 : integer, pid2 : integer)
```

- The `city` attribute in `Company` indicates a city in which the company is located. (A company may be located in **multiple** cities.)
- The relation `worksFor` stores the unique company (identified by `cname`) for which a person works. (It is possible that a person in the `Person` relation does not work for any company. It is also possible that a company has no persons who work for it.)
- The relation `Knows` maintains a set of pairs (p_1, p_2) where p_1 and p_2 are pids of persons. The pair (p_1, p_2) indicates that the person with pid p_1 knows the person with pid p_2. (We do not assume that the relation `Knows` is symmetric: it is possible that (p_1, p_2) is in the relation but that (p_2, p_1) is not.)

We assume the following constraints:

- `pid` is the primary key of `Person`
- (`cname`, `city`) is the primary key of `Company`
- `pid` is the primary key of `worksFor`
- (`pid1`, `pid2`) is the primary key of `Knows`
- `pid` is a foreign key in `worksFor` referencing the primary key `pid` in `Person`
- `cname` in `worksFor` references a `cname` that appears in `Company`. Note that this is technically not a foreign key constraint. Still it is a constraint that should be adhered to in the database.
- `pid1` is a foreign key in `Knows` referencing the primary key `pid` in `Person`
- `pid2` is a foreign key in `Knows` referencing the primary key `pid` in `Person`

You should create the relations `Person`, `Company`, `worksFor`, and `Knows` but without specifying the corresponding primary and foreign key constraints. You should also not yet populate the relations with data.

Solve the following problems:

1. Develop appropriate insert and delete triggers that implement the primary and foreign key constraints for the `Person` and `Knows` relations. If an insert or delete cause a violation for such a constraints, your triggers should report an appropriate error message.

   For this problem, implement the triggers such that foreign key constraints are maintained using the **cascading delete** semantics.

   Provide test cases that show that your triggers work properly.

2. Develop a trigger for the constraint that states that each `cname` in `worksFor` must references a `cname` in `Company`. Since the trigger resembles a foreign key constraints, use the cascade delete semantics for its implementation.

   Provide test cases that show that your triggers work properly.

3. Consider the following view definition

```
create or replace view PersonKnowsNumberofPersons as
  (select p1.pid, p1.name,
         (select count(1)
          from   Person p2
          where (p1.pid, p2.pid) in (select k.pid1, k.pid2
                                     from Knows k)) as ct_Knows_Persons
    from Person p1);
```

   This view defines the set of tuples (p, n, c) where p and n are the pid and name of a person, and c is the number of persons who are known by the person with pid p.

   You should not create this view! Rather your task is to create a relation `PersonKnowsNumberofPersons` that maintains a materialized view in accordance with the above view definition under insert and delete operations on the `Person` and `Knows` relation.

   Your triggers should be designed to be **incremental**. In particular, when an insert or delete happens, you should not always have to reevaluate all the numbers of persons known by persons. Rather the maintenance of `PersonKnowsNumberofPersons` should only apply to the person information that is affected by the insert or delete.

   Provide test cases that show that your triggers work properly.

4. 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

The `Student`, `Prerequisite`, and `hasTaken` are relations that are **immutable**. No inserts, deletes, or updates, are permitted on these relations. For these relations, no triggers need to be developed.

The attributes `cno`, `cname`, `max` in `Course` are **immutable**. On the other hand, the attribute `total` in `Course` is **mutable**: it can increase or decrease. At any time, during insert or delete operation on the `Enroll` and `waitList` relations, this attribute has as values the current total number of students who have enrolled in the affected courses.

Inserts and deletes are permitted on the `Enroll` and `Waitlist` relations. These insert and delete operations 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 value for that course needs to be incremented by 1 provided that this increment does lead to a value that exceed the `max` enrollment value for that course.
- 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 may disenroll from course, either by removing him or herself from the `Waitlist` 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.

Now for the problem:

Write appropriate triggers to enforce these rules.

Provide test cases that show that your triggers work properly.