CS 457 / CS 557 – Database Software Design

Assignment 2

I. Modeling with UML

The airline AirForm proposes us to model in UML a reduced version of its information system of reservation of flight tickets. Flights are planned in advance and assigned to an airplane, an airport of departure, an airport of arrival, a departure date and an arrival date. Each airplane has a capacity in maximum number of passengers. Tickets are issued for each flight when planning, there is no overbooking.

Users buy the tickets. This purchase results in a reservation (via ticket) for the flight in question. We keep the last names, first names, addresses and phones of the users who made a reservation, as well as the booking date and the ticket price. Upon check-in (departure), passengers confirm their tickets for the registered flight. We memorize this initial confirmation.

When the flight is over, the reservations associated with it are archived, and they are deleted when the flight is canceled.

1. Write the high-level use case "Buy the tickets" initiated by a customer, and refine if it is possible this high-level use case.
2. Propose a class diagram that models the AirForm system without representing other elements than those indicated in the statement. Please, specify the attributes for each class.
3. Model the following treatments with detailed sequence diagrams.
   a. Book a ticket on a flight.
   b. Cancel a flight.
4. Suppose that the airline AirForm wants to generalize the system to offer services for high-end tourists, and social group of wealthy people who travel the world to participate in social activities unavailable to ordinary people. Which changes in the AirForm system would you recommend? Justify your answer.

II. Programming with SQL

Instructions

In this section, you are recommended to install *pgAdmin*. In fact, it is a user interface that already includes *PostgreSQL*. Here is a tutorial that illustrates how to install it. Please, proceed as follow:

- Be sure to terminate every SQL statement with semicolon ";".
- Using your favorite editor, create files named *createall.sql* to create all the tables, *dropall.sql* to drops all the tables, *populateall.sql* to populate all the tables and *Assignment2Px.sql* that contains all the queries of the problem *x*. For example, *Assignment2P1.sql* denotes the queries of the problem 1.
- Include your name (and the name if your teammates if it is applicable) in a header comment at the top of your source file.
- Make sure that the output of each query is distinguishable. Comment your code; if nothing else, mark each query with its number.
- Good luck ☺.

Problem 1

You are going to use *PostgreSQL* to design a simple University database. You will create tables and implement some queries.

Create the tables described below. Name these tables TEACHER, COURSE, STUDENT, ENROLMENT, COURSE_SCHEDULE.

TEACHER (t_id : number, t_name: text, t_status:text , t_dept: text)

COURSE(c_id : text, c_name: text,c_level :text)

STUDENT (s_id : number, s_name: text, s_status: text)

ENROLMENT (#c_id : text, #s_id : number)

COURSE_SCHEDULE (#c_id : text, #t_id : number)

Primary Keys are underlined, and the foreign keys are preceded by the symbol #.

Populate the tables you created by relying on the data provided in the file *populate.txt*.

Queries

1. Find the name(s) of all teachers(s) who are from ECE department.
2. Find the name(s) of all student(s) enrolled in CS250

3. Find the student id(s) and names(s) of all students enrolled in CS348 and either in ECE264 or in CS503
4. Find the name of the teacher teaching MA525
5. Find the name(s) of all students enrolled in one or three courses
6. Find the name(s) of all students who are being taught by Prof. Christopher Clifton.
7. Name any undergraduate course(s) being taken by graduate student(s).
8. Name any undergraduate student(s) who is taking a course with Prof. Sheron Noel

Problem 2

Suppose that you have the following university schema:

create table *classroom* (

building **varchar**(15),
room_number **varchar**(7),
capacity **numeric**(4,0),
primary key (*building*, *room_number*)
);

create table *department*(

dept_name **varchar**(20),
building **varchar**(15),
budget **numeric**(12,2) **check** (*budget* > 0),
primary key (*dept_name*)
);

create table *course* (

course_id **varchar**(8),
title **varchar**(50),
dept_name **varchar**(20),
credits **numeric**(2,0) **check** (*credits* > 0),
primary key (*course_id*),

foreign key (*dept_name*) **references** *department*

on delete set null

);

create table *instructor* (

ID **varchar**(5),

name **varchar**(20) **not null**,

dept_name **varchar**(20),

salary **numeric**(8,2) **check** (*salary* > 29000),

primary key (*ID*),

foreign key (*dept_name*) **references** *department*

on delete set null

);

create table *section* (

course_id **varchar**(8),

sec_id **varchar**(8),

semester **varchar**(6)

check (*semester in* ('Fall', 'Winter', 'Spring', 'Summer')),

year **numeric**(4,0) **check** (*year* > 1701 **and** *year* < 2100),

building **varchar**(15),

room_number **varchar**(7),

time_slot_id **varchar**(4),

primary key (*course_id*, *sec_id*, *semester*, *year*),

foreign key (*course_id*) references *course*

on delete cascade,

foreign key (*building*, *room_number*) **references** *classroom*

on delete set null

);

create table *teaches* (

ID **varchar**(5),

course_id **varchar**(8),

sec_id **varchar**(8),

semester **varchar**(6),

year **numeric**(4,0),

primary key (*ID*, *course_id*, *sec_id*, *semester*, *year*),

foreign key (*course_id*,*sec_id*, *semester*, *year*) **references** *section*

on delete cascade,

foreign key (*ID*) **references** *instructor*

on delete cascade

);

create table *student* (

ID **varchar**(5),

name **varchar**(20) **not null**,

dept_name **varchar**(20),

tot_cred **numeric**(3,0) **check** (*tot_cred* >= 0),

primary key (*ID*),

foreign key (*dept_name*) **references** *department*

on delete set null

);

create table *takes* (

ID **varchar**(5),

course_id **varchar**(8),

sec_id **varchar**(8),

semester **varchar**(6),

year **numeric**(4,0),

grade **varchar**(2),

primary key (*ID*, *course_id*, *sec_id*, *semester*, *year*),

foreign key (*course_id*,*sec_id*, *semester*, *year*) **references** *section*

on delete cascade,

foreign key (*ID*) **references** *student*

on delete cascade

);

create table *advisor* (

s_ID **varchar**(5),

i_ID **varchar**(5),

primary key (*s_ID*),

foreign key (*i_ID*) **references instructor** (*ID*)

on delete *set* **null**,

foreign key (*s_ID*) **references** student (*ID*)

on delete cascade

);

create table *time_slot* (

time_slot_id **varchar**(4),

day **varchar**(1),

start_hr **numeric**(2) **check** (*start_hr* >= 0 **and** *start_hr* < 24),

start_min **numeric**(2) **check** (*start_min* >= 0 **and** *start_min* < 60),

end_hr **numeric**(2) **check** (*end_hr* >= 0 **and** *end_hr* < 24),

end_min **numeric**(2) **check** (*end_min* >= 0 **and** *end_min* < 60),

primary key (*time_slot_id*, *day*, *start_hr*, *start_min*)
);

create table *prereq* (
course_id **varchar**(8),
prereq_id **varchar**(8),
primary key (*course_id*, *prereq_id*),
foreign key (*course_id*) **references** *course*
on delete cascade,
foreign key (*prereq_**id***) **references** *course*
);

After you create the previous tables in *pgAdmin* by importing the file *createTables.sql*, write the following queries in *SQL*, using the university schema. We suggest you to populate your tables by importing the file *PopulateData.sql* in *pgAdmin*.

1. Find the titles of courses in the Comp. Sci. department that have 3 credits.
2. Find the IDs of all students who were taught by an instructor named Einstein; make sure there are no duplicates in the result.
3. Find the highest salary of any instructor.
4. Find all instructors earning the highest salary (there may be more than one with the same salary).
5. Find the enrollment of each section that was offered in Autumn 2009.
6. Find the maximum enrollment, across all sections, in Autumn 2009.
7. Find the sections that had the maximum enrollment in Autumn 2009.

Suppose you are given a relation grade *points(grade, points)*, which provides a conversion from letter grades in the takes relation to numeric scores; for example an "A" grade could be specified to correspond to 4 points, an "A−" to 3.7 points, a "B+" to 3.3 points, a "B" to 3 points, and so on. The grade points earned by a student for a course offering (section) is defined as the number of credits for the course multiplied by the numeric points for the grade that the student received.

Given the above relation, and our university schema, write each of the following queries in SQL. You can assume for simplicity that no takes tuple has the null value for grade.

8. Find the total grade-points earned by the student with *ID* 12345, across all courses taken by the student.
9. Find the grade-point average (GPA) for the above student, that is, the total grade-points divided by the total credits for the associated courses.
10. Find the ID and the grade-point average of every student.
11. Increase the salary of each instructor in the Comp. Sci. department by 10%.
12. Delete all courses that have never been offered (that is, do not occur in the section relation).
13. Insert every student whose *tot_cred* attribute is greater than 100 as an instructor in the same department, with a salary of $10,000.

Submission

Be sure to submit one pdf file that contains your answers to **Modeling with UML** and .sql files you used to solve the problems of **Programming with SQL**. Do not use .zip file nor .rar file. *Only submissions on Moodle will be accepted*.