CPS352 - DATABASE SYSTEMS

Database Design Project - Various parts due as shown in the syllabus

Purposes: To give you experience with developing a database to model a real domain

Requirements

This project is deliberately made quite open-ended. It consists of the following three parts.

1. Choose a domain with which you are familiar - other than one of the domains used for examples in the book (university) or lectures (a library). (See discussion on "Choosing an Appropriate Domain" below.) Identify a set of requirements for a system that is appropriate for this domain. (If you wish, you may choose an appropriate subset of a larger domain.)
2. Develop an E-R diagram for a database that models this domain (or the chosen subset).
3. Turn your E-R diagram into a normalized relational database design for the (subset of the) domain - i.e. a set of tables, each with appropriate attributes, a primary key, and appropriate foreign keys. The database should be based on the E-R diagram, but one-to-one and one-to-many relationships may be implemented by appropriate attributes in the "one" entity, rather than as separate tables. Your relational database must be 4NF. Then implement your design as follows.

    a. Create a file of SQL `create` statements. This should include not only creation of tables, but also (as appropriate) creation of named constraints, triggers, and/or views.

    b. Create your database under your username as a schema in the `design` database on the server (i.e. you won't actually create the database itself, just the tables, etc. within a schema in it.) If you are using a copy of db2 on your own computer, you will need to issue the following command (just once) to specify that the database `design` is found on the server, not your own machine.

    ```
    catalog database design at node csserver
    ```

    c. Populate your database with sample data to allow testing of the schema and the various transactions. Each table should have a *minimum* of five rows

    d. Create SQL select/insert/update/delete statements that correspond to your initial requirements.

    e. Thoroughly test your queries and your constraints using your sample data. Among other things, this means ensuring that your constraints correctly catch various kinds of invalid insert/update/delete operations, while allowing legitimate ones. Be sure to test using both "good" and "bad" data. Note that the comprehensiveness of your test data will be a factor in the grading of this part of the project.

Turn in:

Your project will be turned in in three parts. You may wish to modify your approach/design for subsequent parts based on feedback from each part. (You do not need to redo the prior part unless you totally change the project.) The project grade will be assigned when the final part is turned in, but will consider earlier parts as well, though the most significant factor in the grading will be Part III.

On the due date for each of the parts, you will present your work orally to your classmates. For the first part, we will also spend some time as a class discussing the next part of the project - e.g. what might be appropriate entities and relationships for modeling this domain. (We may also have brief class discussion on the second part, though not as extensively as on the first part.)

To facilitate oral presentation and discussion, you will need to prepare appropriate material for projection or handouts.

Specific Requirements for the Various Parts

Part I - Requirements

- A description of the problem domain (written using terminology that a user of the system would use, not technical database terminology.
- A statement of requirements. This should take the form of an overall use case diagram, with explanation in "requirements language" as needed. (See example systems from CPS122).

Part II - E-R Diagram

- An ER diagram for your database, including attributes for entities and relationships.

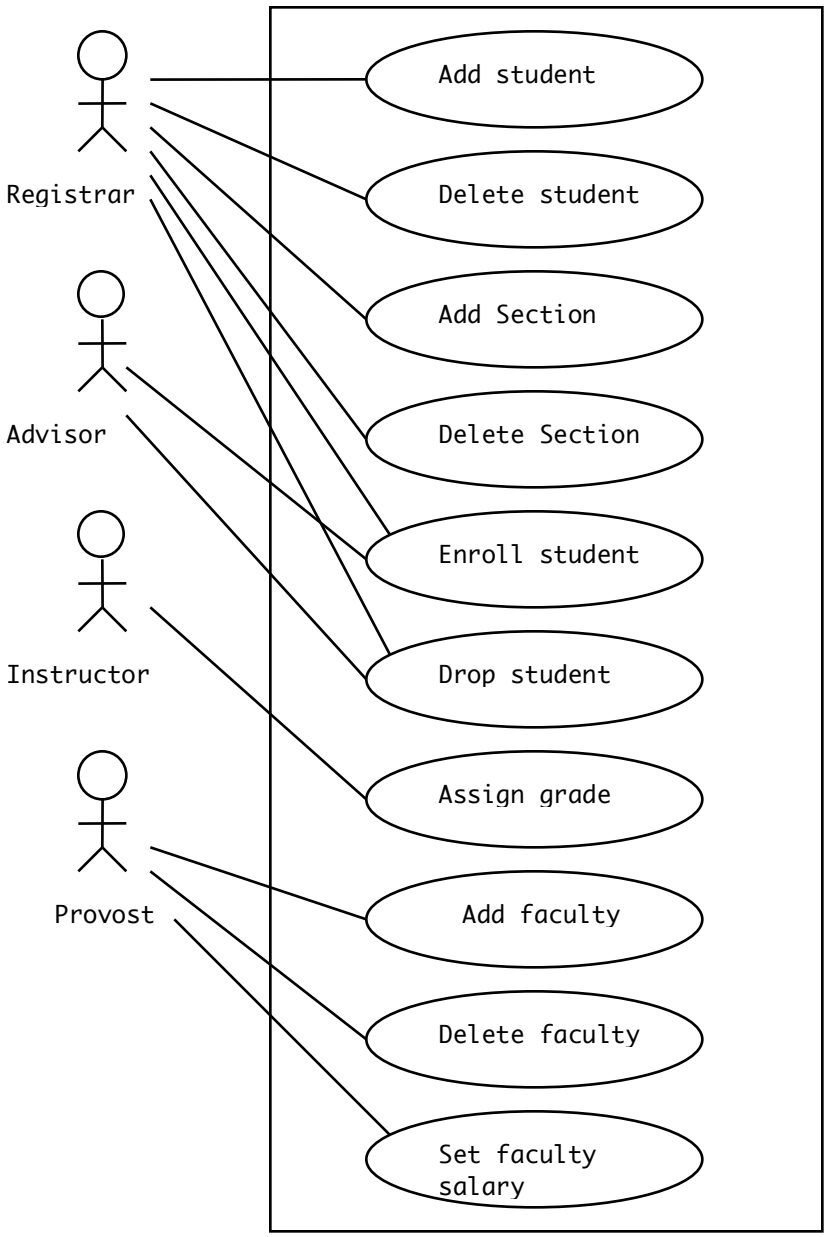
Part III - Relational Database Design, Implementation, and Sample Data

- A list of the functional and multivalued dependencies for your scheme.
- A schema diagram for your database, with primary and foreign keys specified appropriately.
- A printout of a file containing your SQL `create` statements that create the database.
- The SQL statements corresponding to each of your requirements. (If your requirements call for more than a dozen or so statements, you can do a subset). Your SQL statements should require a variety of SQL capabilities, such as various kinds of join, aggregate functions, etc. (This presupposes a good initial domain choice.)
- Documentation of testing
  - You must exercise each of your SQL statements. Your testing should be related to your original requirements - that is, you should include tests that address each of the original requirements (or a subset if there are many).
  - You must also exercise each of your triggers.
  - You must also exercise each of your constraints, being sure it correctly catches errors while allowing legitimate data. (Note: you do not need to test `not null` constraints.)
  - You must turn in printouts of the results of these tests indicating that the SQL statements for your requirements worked correctly and that your constraints correctly allowed good data and caught bad data. **Your printouts must include comments (handwritten or entered as SQL comments in a test file) that indicate what each test is testing - i.e. a specific requirement (if "good" data) or a specific problem you were catching (if testing a constraint on "bad" data).**

An Example:
(Note: this example is *much* simpler than what you would do for the actual project - it's only meant to illustrate what the various pieces of the project need to include.)

Part I

Suppose you chose a subset of the domain of university organization, as used for examples in the book. (As noted above, you can't actually choose this domain.) Section 1.6.2 on pages 16-17 of the *Database System Concepts* book describes this system; you would also need to spell out the requirements, perhaps by using a use case diagram like the one below. (Many possible cases are omitted, including those involved in managing departments, buildings, and classroom space, as well as managing the college catalog and historical information; this diagram is for illustration purposes only.) Since the nature of each requirement is clear from the use case name, no further specification is needed. *Something that looks like section 1.6.2 in the book plus the use case diagram below is what you would turn in for Part I of the project.*



Part II

Your design might be based on an E-R diagram like figure 7.15 on page 282 of the *Database System Concepts* book. *Something that looks like figure 7.15 in the book is what you would turn in for Part II of the project.*

Part III

a. The following FD's and MVD's hold on the E-R diagram. (The names of ID attributes have been changed to explicitly indicate the table to which they belong)

```
course_id → title, credits, dept_name
dept_name → building, budget
instructor_id → name, salary, dept_name
student_id → name, tot_cred, dept_name, instructor_id (advisor)
section_id, semester, year, course_id → building, room_number, time_slot_id
building, room_number → capacity
time_slot_id → day, start_time, end_time
student_id, section_id, semester, years, course_id → grade
course_id ->> course_id (prereq)
instructor_id ->> section_id, semester, year, course_id
section_id, semester, year, course_id ->>  instructor_id
student_id ->> section_id, semester, year, course_id
section_id, semester, year, course_id ->>  student_id
```

b. A database schema based on these dependencies might look like figure 2.8 on page 47 of the *Database System Concepts* book. No tables are needed corresponding to the relationships `course_dept`, `inst_dept`, `stud_dept`, `sec_class`, `sec_time_slot` in the E-R diagram - these relationships are folded into the `course`, `instructor`, `student` and `section` tables because each relationship is one-to-many with total participation required. There is no table corresponding to `sec_course` because `section` is a weak entity dependent on `course`, so its primary key is needed in the table for `section`.

c. This database could be created with SQL statements like those in Figure 4.8 on page 132 of the *Database System Concepts* book (though more are needed to create the complete schema).

- Notice how appropriate `primary key`, `foreign key` and `check` constraints have been specified.
- Notice how cascading delete and update can be specified for `course`. (See page 133.)
- Triggers might be used as shown in Figures 5.8 (page 182) and 5.9 (page 183).
- DB2 automatically creates indexes for each primary key (but on some systems it might be necessary to explicitly do so.) However, it might be desirable to create an index on `name` for `student` if we expect to frequently have to do name lookups.
- A view could be created to allow each student to look at his/her course enrollments.

d. This example does not include SQL for all requirements - but just to illustrate the point:

   - For the requirement "Add student":

   ```
   insert into student values(student_id, name, dept_name, 0)
   ```

e. This example does not include test data for all requirements and constraint - but just to illustrate the point, the following would test the primary key and foreign key constraints on `student` (assuming that the `CS` department was already created).

   ```
   insert into student (student_id, name, dept_name, tot_cred)
       values (12345, 'Anthony Aardvark', 'CS', 0);
   insert into student (student_id, name, dept_name, tot_cred)
       values (12345, 'Zelda Zebra', 'CS', 0); (fails PK)
   insert into student (student_id, name, dept_name, tot_cred)
       values (98765, 'No such', 'ZZ', 0);          (fails FK)
   ```

   Documentation for this test would consist of a screen shot or "cut-and-paste" command line output showing DB2 accepting the first insert and giving an error message for each of the two invalid inserts, with suitable comments.

   The trigger shown in figure 5.9 could be tested by the following SQL statement (assuming the appropriate student and section already exist):

   ```
   update takes
      set grade = 'A'
      where student_id = 12345 and course_id = 'CPS352' and sec_id = 'GS'
             and semester = 'SP' and year = '2010';
   ```

   this would result in an increase of 4 in the `tot_cred` attribute for student 12345. Documentation for this test would consist of a screen shot showing two select statements, with the update statement between them.

Something that looks like all of the above is what you would turn in for Part III of the project.

Choosing an Appropriate Domain

The domain (or subset of a domain) that you choose for your project should have the following characteristics:

1. Size. An appropriately sized domain will result in a database having about a dozen tables (more or less).
2. The entities comprising your domain should be interrelated.
3. Your schema should include some attributes that make it possible to include some transactions that involve aggregate functions. (For example, the schema developed above would allow for queries to calculate the enrollment in each section, or the average enrollment in courses for a given department, or the total courses being taught by each instructor, etc.), and should also make interesting constraints and triggers possible.

Important: look over the requirements for Part III when choosing a domain, to ensure it will allow you to do a good job on the final part!