Overview: In this assignment, you will build a database-driven web information management system from ground up. We will give you an application domain to work on, your goal is to design the underlying database and define the application functionalities you will provide with the database, and implement this application as a web-based system.

Assignment: In this assignment you are to implement a three-tier client-server architecture.

1. **Database Back-End**, which runs the Oracle DBMS on `aloen.cs.arizona.edu`. Your job is to design the database relational schema, create tables and populate your tables with some initial data. We are requiring that you create an E-R diagram, analyze the FDs of each table and apply table normalization techniques to your schema to justify that your schema satisfies 3NF, and, if possible, BCNF.

2. **The business logic and data processing layer**, which is the middle tier that runs on an application server (`lectura.cs.arizona.edu`) running the Tomcat web server. We’ll be using the Spring framework for Java. Spring provides an MVC (Model–View–Controller) format to design this layer of the application. This layer helps you create a controller which will allow you to handle GET, POST, PUT and DELETE requests made by the client. The response generation may involve accessing, from within the controller, the back-end Oracle database you have created. The demo provided (see below) demonstrates how the HTML page and the controller would interact (Controller-View interaction).

3. **Web Front-End**, which is the client user-interface. You need to design web pages appropriately to handle all the required functionalities. Your client application can run on a CS lab workstation, or on any machine that can tunnel into lectura.

Application Domain: The problem description for the project is as follows:

You are tasked with creating an application for the Lester McCann Accidental Occidental University Residence Office.

Data stored for each SWU Student includes: ID number, name, home address, phone, email, gender, date of birth, category of student (for example: first-year undergraduate, postgraduate), major, minor, and Adviser ID number. Upon acceptance to SWU, each Student is assigned and Adviser—data held on a student’s Adviser includes ID number, name, position, department name, telephone number, and email.

The residence office manages two types of student accommodations: Rooms in a Residence Hall and furnished student Apartments. Residence Halls have a name, address, telephone number, and a hall manager (a Residence Office Staff member). Rooms in a residence hall have a room number, and a monthly rent rate, and accommodate one student. Room numbers are unique among the rooms in the same residence hall, but may not be unique across residence halls.
Student apartments provide accommodation for three, four, or five students. Each bedroom in a student apartment has a room number, an apartment number, a monthly rent rate. (The room and apartment numbers uniquely identify the student apartment.)

Students may rent rooms in residence halls or student apartments. In addition to the lease number, which uniquely identifies a Lease, the residence office also stores the duration of the lease (1 or 2 semesters), the leasing student's name and ID number, the Room and Apartment or Residence Hall numbers, the monthly cost of the accommodation at the time of lease creation, and the lease start date.

At the start of each semester, each student is sent an Invoice for the following rental period. In addition to an Invoice number (which uniquely identifies the Invoice), stored information for an Invoice includes at least the following: lease number, semester, payment due, and the date the invoice was paid.

Information stored about residence office Staff members includes: ID number, name, email, home address, date of birth, gender, job title (e.g. Hall Manager, Administrative Assistant, Cleaner), and location (e.g. Residence Office or Residence Hall).

Finally, Rooms and Apartments are subject to routine Inspections, conducted by a residence office Staff member. Information stored about Inspections includes at least the following: date of inspection, ID number of Staff member conducting the inspection, identifying Room or Apartment information, whether or not the residence was deemed in satisfactory condition, and what action was taken to address unsatisfactory conditions if any.

(Credit: This scenario is adapted from the University Accommodation Office case study in Appendix B of the textbook.)

Your web interface should be able to provide the answers to the following queries:

1. What are the manager names and telephone numbers of each Residence Hall?
2. Display a table of summary data about students with currently active leases, laid out as follows: Each column corresponds to a type of residence (residence hall or furnished apartment), and each row corresponds to a student category (first-year undergraduate, post-graduate, etc.); each cell contains how many students of that student category are leasing that type of residence. For example, if there are 10 "post-graduate" students living in "furnished apartments," then the entry in the "post-graduate" row and the "furnished apartment" column should be 10.
3. List the names of students with outstanding unpaid invoices, identifying information for the Room/Apartment associated with the Invoice (including Residence Hall if necessary), and the amounts due. In addition, calculate the total outstanding debt of all unpaid invoices.
4. Two additional non-trivial queries of your own design. See “Required functionalities” below for details.

In addition, the web interface should enable users to:

1. Add a new Student or Staff member to the database.
2. Add a new Lease to the database given information about an existing Student and Room/Apartment.
3. Update the monthly rent of a Room or Apartment. (The update should not change the monthly rent of currently active leases for that Room/Apartment, but should change the monthly rent for leases created in the future.)
4. Delete a Student from the database (along with all that Student’s Leases, past and current).

(Continued...)
This description does not describe every detail. These are the essentials; we expect that your team will create logical and conceptual designs that incorporate these features, at minimum. You are free to add additional details that you feel are appropriate.

We realize that you are not a Vice President for Resident Life at a university, but just by being university students you know the basics. Starting with that foundation, build entities and relationships that meet this school’s needs.

**Required functionalities:** Including the requirements and restrictions of the provided application domain above, your system is expected to perform all of the following operations:

1. **Record insertion:** Your application should support inserting a new data record via web interface.
2. **Record deletion:** Your application should support deleting an existing data record via web interface.
3. **Record update:** Your application should support updating an existing data record via web interface.
4. **Record query:** Your application should support querying your database via the web interface for the problem description given above. You are required to implement the three given queries as well as at least two queries of your own design, but with the following restrictions: At least one must be constructed using information gathered from the user by the web interface. At least one must involve two or more relations. And, there should no trivial queries (for example, simply selecting everything from a table); your queries should be constructed to answer questions that real users would be expected to ask.

For each table you create, you need to populate a reasonable number of tuples in order to test your queries. Some data basics are provided in the application domain description; the rest are left for you to determine, based on your needs. (What is ‘reasonable’ is difficult to define; a few dozen tuples per relation certainly would be; just a handful per relation may not provide sufficient variety.)

**Working in Groups:** In industry, such a project is usually the work of multiple developers, because it involves several different components. Good communication is a vital key to the success of the project. This assignment provides such an opportunity for teamwork. Therefore, we are accepting team sizes of either two or three members. Because we want this class to include at least one team project, working alone is not permitted.

You will need to agree on a reasonable work-load distribution scheme. More importantly, you need to come up with a well-formed design at the beginning. This will minimize conflicts and debugging effort in the actual implementation.

**Hand In:** There are three due dates for this assignment:

1. **Team Composition:** By the first due date (see the top of the front page of this handout), one member of your team must email both of the TAs (Jacob (jacobcombs@email.arizona.edu) and Aakash (aakashrathore@email.arizona.edu)) with the names of the members of your team. Failure to do so by the start of class on this date will cost your team the points listed in the Grading section (below).

2. **E–R Diagram:** As stated above, in the Assignment section, your team will need to create an E–R diagram that describes your database design. Before the second due date, your team will need to prepare a draft of your E-R diagram **and** a member of your team will need to schedule a meeting with the TAs to discuss the draft. The purpose of this requirement is to allow the TAs to review your schema and make suggestions for improvement. The sooner you create your design and discuss it with the TAs, the more time you will have to refine your final E-R diagram. In a few days, we’ll let you know how to sign up for a discussion time.

(Continued...)

3
3. **Final Product**: On or before the third due date, a member of your team must submit a `.tar` file of your well-documented application program file(s) via turnin to the folder `cs460p4`. The tar file should contain all of the following:

(a) The source code for your application. The structure of it should follow that of the demo application (see below).

(b) A subdirectory called “doc”, containing a PDF document including these sections in this order:
   i. *Conceptual database design*: Final ER diagram along with your design rationale and any necessary high-level text description of the data model (e.g., constraints or anything not able to show in the ER diagram but is necessary to help people understand your database design).
   ii. *Logical database design*: Converting an ER schema into a relational database schema. Show the schemas of the tables resulted in this step.
   iii. *Normalization analysis*: Show the FDs of all your tables and justify why your design adheres to 3NF.
   iv. *Query description*: Describe your two created queries. Specifically, what questions they are answering? What is the significance of including them in the system?

(c) A `ReadMe.txt` describing:
   i. How the class staff can operate your website to see the required functionalities
   ii. The workload distribution among team members (that is, which people were responsible for which parts of the project?).

In addition, each team should schedule a time slot (10-15 minutes) to meet with a TA and demonstrate your system. Closer to the third due date, we will let you know how to sign up.

**A Simple Demo Application**: To speed up the development, we’ve provided a simple demo application. The demo contains a simple web page with a simple interface. Please read the short documentation file that accompanies it to see how to run it. We also demonstrated this demo the day the assignment was handed out.

The demo package and documentation are linked from the class web page, below the PDF link of this assignment handout. You should download, install, and run this demo yourself soon because (a) Spring is a new framework this semester, undoubtedly possessing multiple unforeseen difficulties and points of confusion; and (b) running it will help you get familiar with the components and directory structure that you are going to use for this assignment.

**Grading**: Total: 100 points

1. Team Composition (1st due date): 5
2. Complete E–R Diagram Draft and meet with TA(s) for feedback (2nd due date): 20
3. Final Submission (3rd due date):
   
   (a) Coding / Implementation: 60
   - Documentation 10
   - Style and organization 10
   - Record insertion: 5
   - Record deletion: 5
   - Record update: 10
   - Record query: 10
   - web front-end: 10

   (b) Database design: 15
   - Final E–R diagram: 5
   - Normalization analysis: 10

(Continued...)
Note: We won’t put much weight at all on the appearance of the pages; the main point of the assignment is the DB design. Put your focus on web page functionality. Don’t worry if your web pages don’t look “nice”.

Late days: Late days can be used on this assignment, but only on the third due date. How many a team has to use is determined as follows: Team members total their remaining late days, and divide by the number of members in the team (integer division), producing the number of late days the team has available, to a max of two days. (Justification: The TAs need to get grading done soon after the due date, and you need time to study for your final exams.)

For example, a team whose three members have 1, 1, and 3 late days remaining have \( \left\lfloor \frac{1+1+3}{3} \right\rfloor = 1 \) late day to use to get their project materials submitted.