CSc 460 — Database Design
Fall 2020 (McCann)

http://u.arizona.edu/~mccann/classes/460

Program #4: Database–driven Web Application

Due Dates:

Team Members:	November 24th, 2020, at the beginning of class
Draft E–R Diagram:	December 1st, 2020, at the beginning of class
Final Product:	December 8th, 2020, at the beginning of class

Designed by *Zheng Tang and Chenghao Xiong*

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 `aloe.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 Maven 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:

Black Friday is coming, you are tasked with designing an application for a local shopping mall in Tucson. (Yes, they still exist!)

The shopping mall provides discounts only for its members. Each member has his/her member ID, first and last name, date of birth, address, phone number, and reward points. To encourage members to purchase more, the mall offers cash–back bonuses. Members will get one reward point for every dollar they spend in the mall. Members can redeem $1 store credit using 100 reward points. This is not a free membership; each member needs to pay a monthly member fee of $5. (You can assume the monthly member fee is $5 charged on the first day in the month, and members can redeem $1 store credit using 100 reward points and only in multiples of 100. For a further implementation, you can store them into a table.)

There are three groups of employees for this shopping mall: The sales group, the stock group, and the manager group. Each employee has his/her employee ID, first and last name, gender, address, phone number, group, salary. In reality, one employee can belong to multiple groups depending on their duties, but in this project each employee belongs to only one group.

(Continued...)

Data stored for each product includes: product ID, name, retail price, category, membership discount(dollar amount) and stock info.

You should also maintain information about the suppliers, including supplier ID, name, address, and contact person.

This shopping mall has a large warehouse for its products. For this project, you need to keep track of the incoming products. The necessary information includes supplier ID, product ID, incoming date, purchase price, and amount.

The most important records for a shopping mall are the records of the sales. Just like the usual receipt we get every day, each sale record must contain information such as sale ID, date, payment method, total price, and member ID. (Total price here is the transaction amount. Notice that people can buy multiple products in one transaction.) It also needs a list of sub–records of each individual product, which has sub–sale ID, product ID, sale ID, price and amount.

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.

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.)

We realize that you are not an expert in this domain, but you have dealt with similar systems in your life. Hopefully, you have enough experience that this problem description makes sense. If you have questions, please ask, and the TAs will help you clear things up.

Required functionalities: Within the framework provided above, your system is expected to perform examples 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. *Queries*: Your application should support querying your database via the web interface for the problem description given above. You are required to implement three provided queries as well as at least one query of your own design. Details are provided below.

(Continued...)

Specifically, the web interface should enable users to:

1. Add, update or delete a member, employee, product, and supplier. When adding, the user cannot leave the ID, name and phone number empty. However, other records can be left empty and be updated later. When updating, the user is allowed to update everything except the ID. When deleting, the entire row needs to be deleted.
2. Add or update supply records. When adding, the supply ID and date cannot be empty. You cannot add a supply record with a non-exist product ID or supplier ID. Also, the date cannot be ahead of the current date.
3. Add or update sale records and sub–sale records. You need to initialize a sale record with non–empty sale ID and a total price of 0. Then you need to update the total price each time you add a new sub-sale record related to this sale. You also need to apply the membership discount during the calculation if it is a member purchase.

Here are the queries that your application should be able to answer:

1. Write a query that displays the member by searching the phone number or the ID. The results should display the member name, date of birth, and reward points.
2. Write a query that displays the profit of the current month (so you do not need to worry the member and employee changes). The number can be calculated as the sale income plus the member fees minus the supply charge and employee salaries. To calculate the sale income, for your convenience, we assume the customers will use their reward points first and pay for the rest. Some reward points will be given according to how much they pay for the rest.
3. Check the current the most profitable product over all the products in the database. The profit should be calculated as amount sold * (retail price * discount(if any) - purchase price). You should display the product name and the total profit.
4. One additional non-trivial query of your own design, with these restrictions: it must involve at least two tables.

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 (with documentation!) named `program4-files.zip` is in the class directory (`/home/cs460/fall20/`) on lectura. You should download, install, and run this demo yourself soon because (a) the Spring framework is probably new to you, and (b) running it will help you get familiar with the components and directory structure that you are going to use for this assignment.

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 an opportunity for just this sort of teamwork. However, we realize that our on–line semester makes it harder for you to get to meet and know your classmates, to learn if they'll be reliable teammates. Therefore, we are accepting team sizes of between one and four members (inclusive), but definitely recommend working in teams due to the complexity of the project.

You will need to agree on a reasonable workload distribution plan. 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.

(Continued...)

Hand In: Here are the 'deliverables' for each of the assignment's three due dates:

1. *Team Composition*: By the first due date (see the top of the front page of this handout), one member of your team must create a PRIVATE post on Piazza using the "program 4" tag 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 Criteria 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 submit it through `turnin`. 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. If TAs need further explanation of your E–R Diagram, they'll send out an email to make an appointment to have an additional meeting.

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 created query. Specifically, what question is it answering? What is the significance of including such a query 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 (∼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.

(Continued...)

Grading Criteria: Total: 100 points

1. Team Composition (1st due date): 5
2. Complete E–R Diagram Draft (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

Grading Notes:

1. Unless we receive verifiable complaints about inadequate contributions, each member of a team will receive the same score.
2. We won't put much weight at all on the appearance of the web pages; concern yourselves with the web page functionality instead. The main point of the assignment is the DB design.

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 late**. (Justifications: The TAs need to get grading done soon after the due date, you need time to study for your final exams, and the department has a rule about things needing to be due before the start of finals.)

For example, a team whose three members have 1, 1, and 3 late days remaining have $\lfloor \frac{1+1+3}{3} \rfloor = 1$ late day to use, if needed.