CSc 460 — Database Design
Fall 2017 (McCann)

http://www.cs.arizona.edu/classes/cs460/fall17

Program #4: Database-driven Web Application

Due Dates:

Team Members:	November 21st, 2017, at the beginning of class
E-R Diagram:	November 30th, 2017, at the beginning of class
Final Product:	December 5st, 2017, at the beginning of class

Designed by *Theodore Sackos and Anand Vastrad*

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, which, in this assignment, will be `lectura.cs.arizona.edu` running the Tomcat web server. This layer sits in the middle, receives requests from client application and generates responses back to client application. The response generation may involve accessing the back-end database you have created. Though there are many server-side techniques available for use, in this assignment we are requiring that you use Java and JavaServer Pages (JSP).
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 in any machine within the CS department with a web browser installed.

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

> Infinite Improbability Intergalactic Inc. (I^4 to be succinct) designs, manufactures and distributes luxury space ships by contract. The company produces various space ship types. However, only one department manages building one model of ship. Each type of ship requires a particular set of parts (the same for each ship, like a hull, engine, wings, etc.). However, each ship has a unique set of luxury features (such as an Infinite Improbability Drive, Fuzzy Rear-View Mirror Dice, etc.). Each ship has a different number of luxury features, between 3 and 10, in addition to the required set of parts. The company and clients tend to like to know how much it will cost for a ship to be constructed. The company maintains information about how much each part costs, including luxury features. I^4 then sells the ship at an (exorbitant) markup based on the cost of the parts.
>
> Departments whose designs are approved (such as the Galactic Empire department's trusty Death Star) may have zero to many active contracts, but only produce one type of ship. When a department is building multiple ships at the same time, each ship can be at a different state of assembly. In other words, you need to keep track of which parts of the ship have already been assembled, and which parts remain to be added. Customers enter contracts with I^4 to have specific ships built. Some high-rollers are frequent customers and may have multiple space crafts being built at the same time.

(Continued...)

I^4 maintains a catalog of parts available through the company's suppliers. This catalog stores a record of all items the company uses in order to build their ships. Each ship starts with no progress. The web client you build must incorporate features to answer questions about the current state of the company, as well as features that allow an engineer in some department to add progress (finish adding a part) to a ship, and update the total cost of that ship up to that point.

Expected Queries that should be answered by the web interface:

1. Choosing one from a list of ship types, report the cost of all the parts.
2. List all ships that are partially built but incomplete.
3. Find the customer that has paid the most for ships from I^4, and how much they have spent. This includes the markup applied to the cost of parts. Include both ships that have been completed for this customer and ships that are currently contracted by this customer.
4. Create two additional non-trivial queries for your team to include with these on the web-interface.

DML Requirements:

- Add ships/departments/parts to the database.
- Order a ship as if the user was a customer.
- Update the cost of an item (only affects ships that buy that part in the future)
- Scrap a ship under construction (parts cannot be salvaged), including the contract for that ship.
- Add to a ship's progress by updating which parts of the ship have been built, and the current cost of construction.

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 executives of ship design firms, and as such are not knowledgeable about all aspects of the design process. Neither are we! However, you all have a basic understanding of how large projects are managed. Starting with that foundation, build entities and relationships that meet this organization's needs. We're far more interested in your DB design and implementation than in how a company like SpaceX or Virgin Galactic operates.

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.

(Continued...)

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 2-4 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 (Theodore (theodorejsackos@email.arizona.edu) and Anand (anandvastrad@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. On or before the second due date, a member of your team will need to submit your E–R diagram **and** schedule a meeting with one of the TAs to discuss your design. 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 a TA, the better. In about a week, we'll let you know how to sign up for a discussion time.

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) A directory called "ROOT", which contains all the source code for the application. The structure of it should follow exactly as what you will see in the simple demo application (see below).
   (b) A directory 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; what questions they are answering? What is the significance of including them in the system?
   (c) A `ReadMe.txt` describing how the TA can operate your website to see the required functionalities, and the work-load distribution among team members (that is, who was responsible for what?).

   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.

(Continued...)

A Simple Demo Application: To speed up the development, we've put a simple demo application under `/home/cs460/fall17/2017p4/`, please read the `HowTo.txt` within it to see how to run the demo. The demo contains a simple web page with a button you can click. By clicking the button, it will retrieve all the records from the products table and display the content on another web page. `NOTE:` You will need to create the table and modify the code to make the demo work on your machine.

Important detail: The demo requires Java 7, not Java 8, and so will your implementation. You can find Java 7 on lectura here: `/usr/local/jdk/bin/javac` (and `.../java`).

You should run this demo soon because (1) it will let you install the Tomcat web server under your account which is needed for your application to run, and (2) it will help you get familiar with the techniques 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 (2nd due date): 10
3. Final Submission (3rd due date):
   (a) Coding / Implementation: 70
      - Documentation 10
      - Style and organization 10
      - Record insertion: 10
      - Record deletion: 10
      - Record update: 10
      - Record query: 10
      - web front-end: 10

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

Note: We won't put much weight at all on the look 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 $\lfloor \frac{1+1+3}{3} \rfloor = 1$ late day to use to get their project materials submitted.