INF1343, Winter 2012, Final Project Requirements – UPDATED ON FEBRUARY 13, 2012.

This assignment consists of two deliverables, jointly worth 35% of your course grade.

1. The initial design is worth 10% of your course and is due at **9:10 am on Monday, March 5, 2012**.
2. The final implementation and report are jointly worth 25% of your grade and are due at **9:10 am on Monday April 2, 2012**.

There will be a penalty for lateness of half a grade (e.g. from A to A–) for any deliverable not submitted on time, with an an additional half-grade penalty deducted for any further 24 hours that the assignment is late. Work that is not handed in one week (168 hours) after it was due will not be accepted.

Both deliverables should be submitted **on paper**. Late assignments can be submitted by email as **PDF** files. Please note that the final submission also involves the implementation of the system, which must be done on the server.

The assignment can be done individually or in groups of 2 or 3. You can choose your own group. If you are having trouble finding a group, contact the instructor as soon as possible. If several students are working on an assignment together then accommodations for illness and other valid reasons will only be provided if ***all*** members of the team are incapacitated. (In other words, if you work with a partner and your partner cannot complete their part, be prepared to finish the work on your own.)

The Problem

For this assignment you will build a prototype of an online bookstore. Your store will allow customers to find books, get information about them, read or post reviews of books, and, of course, order the books.

Below are some of the specific features that you should offer, together with a number of out-of-scope items. (Out-of-scope means you specifically do *not* need to do it.) You should ideally implement all of those features. If you cannot, then do what you can. Please remember, however, that this is a course on *database design*. You will therefore be primarily evaluated on the underlying database, the queries, and your report. A team that designs and builds an excellent database but fails to implement a working website will usually get a higher grade than a team that provides a working web interface to a seriously flawed database. Also, this is *not* a web design exercise, so the aesthetic qualities of your web interface will *not* be a factor in grading. (It just need to work.)

Finding books

A visitor to the site should be able to search for books by entering a part of a title or a part of an author's name. You design should consider that there may potentially be *a lot* of books matching the user's query. Your system probably shouldn't attempt to show all of them at once. You should also think about the order in which you will display the books. (Best-selling titles first? Titles with the most positive reviews first?)

Out-of-scope items:

- **Your system only needs to support very simple search** – the kind that can be implemented using SQL's "like" keyword. So, you also only need to return those items where the search string matches *exactly* a part of a title or a author's name. So, a user searching for "guide to the galaxy" would locate "The Hitchhiker's Guide to the Galaxy," but one searching for "guide galaxy" will not. Also, if the user misspells a term ("hichhiker" instead of "hitchhiker"), then they are out of luck.

Displaying information for a book

From the list of books the user should be able to access each book's individual page. This page should show them information about the book: the title, the names of all of the authors, the price of the book, whether it is in stock. It should also show the average rating for the book by visitors to the site. The book's page should ideally also show an image of the book's cover — you can use http://openlibrary.org/dev/docs/api/covers as a resource.

Reviews

Users should be able to post reviews of books. They will need to be logged in to do this. Each review should include a rating and should show the username of the user who posted the review. When a user posts a new review with a rating, the average rating for the book should be adjusted to reflect this.

There should also be some way to see all the reviews for a particular book, with individual ratings. The reviews should be listed in reverse chronological order (the most recent at the top, the oldest at the bottom).

Ordering a book

If a user identifies a book they want to buy and the book is in stock, then the user should be able to order the book. When ordering the book the user will need to provide their name, address and credit card number — unless the system already has this information on file. (For credit card numbers, you system should accept any 16 digit number as if it were valid.) The user should be able to buy several copies of the same book at once.

Your system should never accept an order for more copies of a book than you have in stock. (That is, if you have only two copies, then the user should be able to order either one or two copies, but not three.) When an order is placed, it should be recorded in the database. Your system should also somehow keep track of the fact that there are now fewer copies of this book in stock. (If you had three copies and someone ordered two, then you have one.)

Your system also needs to provide a *manager's interface*: a page showing the list of outstanding orders, including information about the book that were ordered and the addresses to which they are to be shipped. There should be a way to mark an order as "shipped." The same page should also show the list of books currently out of stock, with an option to increment the number of copies, putting the book back in stock.

Out-of-scope items:

- **You can limit your customers to one title per order.** That is, if the customer wants two copies of *The Hitchhiker's Guide to the Galaxy* and one copy of *The Marx-Engels Reader*, then she will need to place two orders: one order for the two copies of *The Hitchhiker's Guide* and another order for *The Marx-Engels Reader*. (This is not ideal from the customer's point of view, but it will simplify the problem tremendously.)
- **Your system will not track fulfillment of orders.** That is, you just need to make sure that the customers can place orders, that you record their orders, that you don't accept orders for books that have already been sold to other customers, and that there is a way for the manager to see what orders were placed and to mark them as "shipped." You do not need to worry about what happens afterwards.
- **Manager's interface can be public.** In other words, you can make it visible to any person who knows the URL of the page. (You would never do this in a real-world situation, of course, but this is a prototype.)

The web interface

Users will access all of system's functionality through a web browser. The web interface should be implemented using Python CGI, in a manner that will be introduced in class on weeks 7 and 8. If your team would prefer to

use a different approach to web development, *please contact the instructor to discuss this*. Be advised that permission to use a different method will only be granted in cases where *all* members of the team are familiar with this alternative method.

Authentication

Your system will need to somehow recognize which users it is dealing with. You will do this using "session tokens." When a user provides your site with their username and password, your system will check if those match. If the password is correct, your system will generate a "token" — a random number that will serve to identify the user temporarily. This token will be sent to the user. Your system will remember which user is identified by this new token, as well as the expiration time for the token. When the user's browsers requests another page, it will provide your system with the token, which you will use to identify the user. If at some point the user choose to logout, your system will either delete the token or mark it as invalidated.

- For security reasons, session tokens must be *unpredictable*. So, you cannot generate them using auto_increment. Instead, you can use MySQL's function rand() to generate a random number, multiply it by a large number (e.g., 1,000,000,000,000) and then use round() to turn the result into an integer. (It's good to make the token a large number, to minimize the likelihood that the same token would be generated again. However, make sure that the column that is going to store your tokens is actually setup to store sufficiently long numbers.) Alternatively, you can use Python functions to generate the token.

Out-of-scope items:

- **You do not need to implement new user registration.** You will set up a number or accounts and it will be possible for visitors to login using usernames and passwords for those accounts. However, there does not need to be a way for visitors to create new accounts using the web interface.
- **You can require your users to login before accessing any features of the site.** This is not ideal for usability of the site, but it can simplify your task quite a bit.

Dealing with invalid input and exploits

Your system needs to be prepared to deal with invalid input. For example, you should consider what happens if a user orders 0.5 of a book or -2 books. You may also want to consider that multiple users may be ordering things at the same time. This means that while a book may be in stock at the time when a user found it, it may potentially be out of stock by the time when the order is placed. What will your system do at this point? You also need to consider how malicious users may be able to exploit your system. For instance, could it be possible for a malicious user to post reviews as if from a particular username without knowing the password? You should also make sure that your system is not vulnerable to the "SQL injection" attack that we will be discussing in class on week 8.

Your system does not need to be very "user-friendly" when dealing with invalid input. For instance, if a user manages to request a negative number of books, you can just show them an error message saying "Invalid number of copies" and expect them to go back to the previous page and correct it the problem. The most important thing is that your system should never let this kind of data get in.

Sample data

Later in the semester you will be provided with data for you to load or otherwise enter into your system.

Use of CASE tools

All tables for your database must be created using SQL CREATE TABLE statements written "by hand" by the members of the team. They should **not** be generated by CASE tools. *Submissions using CREATE TABLE statements generated by software tools will lose a substantial part of the grade.*

Formatting of the Deliverables

Both of the deliverables described below should strictly adhere to the following format:

- The main text should be typed in a 12 point serif font (such as Times New Roman) and be 1.5-spaced.
- All SQL and Python code should be in a monospace font. (`A monospace font is one in which all letters are of the same width.`) It should also be properly indented.
- All pages should have 2 cm margins.
- All pages should be numbered at the bottom.
- There should be no cover page and no artwork other than the ER diagrams.
- The names of the team members should be shown on the first page.
- The ER diagrams need to be drawn using a software tool and should have labels for all relationships.

Deliverable 1: The Preliminary Design (10% of your course grade)

Your first deliverable is the preliminary design. It should include the following components:

1. **The list of team members.** Include the name and email address of each team member.
2. **A functional specification.** Provide a detailed description of how your system is going to behave from the customers' and the managers' points of view. This should be around 1,000 words long. Please note that he point of this section is not to explain how the system will be *implemented*, but rather how it will *behave*. Writing this section should help you think through the different situations and will provide a basis for judging the adequacy of your ER design. Your functional specification should describe specific use cases. (For example. "The customer wants to buy a copy of Harrington's *SQL*. She goes to the website and sees … She enters … into … She then clicks on … ".) Make sure to explain any non-obvious choices. If you are proposing any additional features, please describe them here.
3. **An ER diagram and notes.** Include an ER diagram and attach explanations of how it is going to support your design. *Make sure to label the relationships in your ER diagram.* Explain any non-obvious decisions — if your diagram does not make immediate sense to the instructor, you notes would hopefully provide an explanation. Make sure your diagram matches your functional specification. (In other words, your ER diagram must be able to support the behavior described in your functional specification.)
4. **The implementation plan.** Describe the schedule for implementing and testing the system. (Make sure to allow time for *testing* and fixing the problems discovered in the course of testing.) State clearly who is responsible for each piece and by when they will need to finish it. The primary goal of this is to make sure all team members are on the same page about who is doing what and when.

Deliverable 2: The Final Report and the Implementation (25% of your course grade)

Your final deliverable will consist of two parts: the final report and the actual implementation that you will setup on the database server.

The **implementation** should include:

1. **The database.** Your database should be adequate for the functionality of the system. It should properly declare all primary and foreign keys. It should be populated with the data that will be provided to you.

2. **A web interface.** The web interface should include a front page through which it should be possible to access all other functionality.

The **final report** should include the following elements, ***in this order***:

1. **The list of team members.** If this list is different from the one in your proposal, attach a note explaining why. (If you expect a change in team composition, please discuss this with the instructor as soon as possible.)
2. **The name of the database.** (Your instructor needs to be able to locate your database to evaluate it.)
3. **The main URL of the web interface.** This is the URL from which all of the functionality should be accessible. If you did not manage to get the web interface to work, please provide here a detailed explanation of what you tried to do to get it to work, what problems you ran into, and why.
4. **A summary of features.** Explain whether your implementation supports all the required features and whether it matches your functional specification. If not, explain why not. Also, if you implemented any additional features, explain here what they are and how they work. (You do not need to do this, but if you do, please make sure to explain.) Also explain here what needs to be done to verify that your system works as described. This should include search terms that one could enter into the search box to get meaningful results.
5. **The final ER diagram and explanation.** Include an ER diagram and an explanation of how it satisfies the needs of your system. *The diagram should match your implementation.* If it does not, for whatever reason, please explain this here.
6. **CREATE statements.** Include the CREATE TABLE statements that you used to create your database. Those statements should all be written "by hand" by members of the team and *not* generated by CASE tools. *They should match your database.* Those should properly setup the necessary primary and foreign keys and should satisfy the conditions for 1NF, 2NF, and 3NF. Provide an explanation for any non-obvious decisions. You generally should not use ALTER TABLE statements to setup your tables. If you do, however, please include those also and explain why they were necessary.
7. **Queries.** Include in your report *all* SQL query templates used by your system. This should include all SELECT, UPDATE, and INSERT queries used inside Python files. Explain any non-obvious decisions.
8. **Additional comments.** You can optionally provide additional explanations.
9. **Statement of responsibilities.** Describe briefly the contribution of each team member.
10. **Credits.** The ER diagram and the SQL used in this assignment should be written exclusively by your team. For other components (e.g., images, CSS, HTML, Python functions) you can use what's available on the web, if you wish. If you do so, state what you got where, including specific URLs. You can only use components that are publicly available and they cannot be custom-made for your project. (In other words, if you find some CSS you want to use, go for it. But do not *ask* someone to write CSS for you.)

If your system does not work as you think it should, please provide the following additional information in order to assist your instructor in giving you partial credit:

- In section 4 please explain what does and does not work, what you attempted to do to make it work, and what you think might be the problem.
- If your Python files do not work, please make sure to still include in section 7 all query templates that you think would be needed and explain how those queries would work. (In particular, you should explain what information would need to be inserted into the templates before they can be sent to the database, where this information would come from, and what the result would be.)

Happy hacking!