CompSci 316 Fall 2014: Course Project

100 points (25% of course grade)
Assigned: Thursday, September 11

Important Dates

Milestone 1: Thursday, October 16
Milestone 2: Thursday, November 13
Early in-class demo: December 4
Demo: Monday-Wednesday, December 8-10

Overview

You have the option of doing either a "standard" or an "open" course project. The "standard" project is to build a database-driven website from the ground up. With this option, there will be some examples and instructions to help you get started. No prior experience in developing such applications is assumed. On the other hand, if you want to try something unconventional, you may choose the "open" option and build anything of your liking—provided it is related to data management. With the open option, you will need to make a detailed proposal, and the course staff may not be able to provide as much programming help and support. Generally speaking, much more work is expected, but the reward may be bigger too.

This document also describes a number of possible ideas for both project options. Feel free to talk to the course staff if you choose one of them. Of course, you are welcome to come up with your own ideas as well. Many of the "open" project ideas below can evolve into "Graduation with Distinction" projects for Computer Science majors, and the instructor will be glad to supervise continuation of successful projects as CompSci391 (Independent Study) or CompSci393 (Research Independent Study).

Submission and Grading

There will be **two milestones and a final project demo**; see *Important Dates* above for due dates. You will find the details of what to submit for each checkpoint later in this document under the sections *"Standard" Course Project* and *"Open" Course Project*. Because of the open-ended nature of course projects, certain instructions may not apply to your particular project; when in doubt, consult the instructor.

Each project will be graded on a scale of 0-100 points. A breakdown is as follows: 30 points for submitting the required work at three checkpoints; 40 points for completing the proposed work; 30 points for the quality of the work. Out of the 70 points for completeness/quality, 5 to 10 points are reserved for impressive and/or innovative work beyond what is expected. In other words, meeting the expectation will ensure a project grade in the *A* range, but *A+* will require exceptional work.

What the "required work" means may evolve over the course of the project. We will start with your Milestone 1 proposal, help you get a feel for the amount of work involved, and work with you to ensure that it meets the minimum requirements of depth and scope for a course project.

Teamwork

The project should be completed in **4-person teams**. *Any other team size requires explicit approval from the instructor.* Regardless of the team size, an equal amount of work is expected and the same grading scale will be applied. All members in a team will receive identical grades for the project. If there is any problem working

with your team members that you cannot resolve by yourself, bring it to the instructor's attention as soon as possible. Last-minute complaints of the form "my partner did nothing" will not be entertained.

Platform Issues

To develop the project you are encouraged to use the VM provided by the course. If you want to run a publicly accessible website, create a VM on Amazon using the course credit. As examples, the course staff will provide the source code (from the course `git` repository) and tutorials (on the course website) for several database-backed websites that are implemented using different technologies and deployable on the course VM. We will make an announcement during the semester once these examples are ready. Of course, there are many other ways to develop web and database applications. If you wish, you may use other languages, tools, or application development frameworks, or run servers on your own machines. Setting up the whole application/database stack is non-trivial and can be a rewarding experience. However, the course staff can only support the course VM and technologies used by the provided examples.

"Standard" Course Project

The "standard" project is to build a database-driven web application. Specifically, you will need to complete the following tasks through the course of this semester. Note that different members of a team can work on some of these tasks concurrently.

1. Pick your favorite data management application. It should be relatively substantial, but not too enormous. Several project ideas are described at the end of this document, but you are encouraged to come up with your own. When picking an application, keep the following questions in mind:
    a. How do you plan to acquire the data to populate your database? Use of real datasets is highly recommended. You may use program-generated "fake" datasets if real ones are too difficult to obtain.
    b. How are you going to use the data? What kind of queries do you want to ask? How is the data updated? Your application should support both queries and updates.
2. Design the database schema. Start with an E/R diagram and convert it to a relational schema. Identify any constraints that hold in your application domain, and code them as database constraints. If you plan to work with real datasets, it is important to go over some samples of real data to validate your design (in fact, you should start Task 7 below as early as possible, in parallel to Tasks 3-6). Do not forget to apply database design theory and check for redundancies.
3. Create a sample database using a small dataset. You may generate this small dataset by hand. You will find this sample database very useful in testing, because large datasets make debugging difficult. It is a good idea to write some scripts to create/load/destroy the sample database automatically; they will save you lots of typing when debugging.
4. Design a web-based user interface for your application. Think about how a typical user would use your site. Optionally, it might be useful to build a "canned" demo version of the site first (i.e., with hard-coded rather than dynamically generated responses), while you brush up your website design skills at the same time. Do not spend too much time on refining the look of your interface; you just need to understand the basic "flow" in order to figure out what database operations are needed in each step of the user interaction.
5. Write SQL queries that will supply dynamic contents for the web pages you designed for Task 4. Also write SQL code that modifies the database on behalf of the user. You may hard-code the query and update parameters. Test these SQL statements in the sample database.

6. Choose an appropriate platform for your application. Python or PHP? To JavaScript or not to JavaScript? Start by implementing a "hello world" type of simple database-driven web application, deploy it in your development environment, and make sure that all parts are working together correctly. The course website will provide pointers to working examples for supported platforms.
7. Acquire the large "production" dataset, either by downloading it from a real data source or by generating it using a program. Make sure the dataset fits your schema. For real datasets, you might need to write programs/scripts to transform them into a form that is appropriate for loading into a database. For program-generated datasets, make sure they contain enough interesting "links" across rows of different tables, or else all your join queries may return empty results. Keep in mind that the course VM's hard drive has limited capacity: for larger databases, you may need to create a separate, bigger virtual hard drive—see course staff for help if you run into issues.
8. Test the SQL statements you developed for Task 5 in the large database. Do you run into any performance problems? Try creating some additional indexes to improve performance.
9. Implement and debug the application and the web interface. Test your website with the smaller sample database first. You may need to iterate the design and implementation several times in order to correct any unforeseen problems.
10. Test your website with the production dataset. Resolve any performance problems.
11. Polish the web interface. You may add as many bells and whistles as you like, though they are optional because they are not the main focus of this course.

Milestone 1. You should have completed Tasks 1-5 and have started thinking about 6 and 7. If you plan to work with real data, you should also have made significant progress on Task 7 (you should at least ensure that it is feasible to obtain the real dataset, transform it, and load it into your database). Submit the following via `websubmit`, under "compsci316" and "proj-ms1":

- A brief description of your application.
- A plan for getting the data to populate your database, as well as some sample data.
- A list of assumptions that you are making about the data being modeled.
- An E/R diagram for your database design.
- A list of database tables with keys declared.
- A description of the Web interface. You can write a brief English description of how users interact with the interface (e.g., "the user selects a car model from a pull-down menu, clicks on the 'go' button, and a new page will display all cars of this model that are available for sale"). Or, instead, you can submit a canned demo version of the website.
- A `.zip` or `.tar.gz` archive of your source code. The source code directory should at least contain:
    - A `README` file describing how to create and load your sample database.
    - Files containing the SQL code used for creating tables, constraints, stored procedures and triggers (if any).
    - A file `TEST-SAMPLE.SQL` containing the SQL statements you wrote for Task 5.
    - A file `TEST-SAMPLE.OUT` showing the results of running `TEST-SAMPLE.SQL` over your sample database. You can create the file by running:
      `psql` *`dbname`* `-af TEST-SAMPLE.SQL > TEST-SAMPLE.OUT`
      where *`dbname`* is the name of your database.
    - If applicable, any code for downloading/scraping/transforming real data that you have written for Task 7 so far.

Milestone 2. You should have completed Tasks 1-8 and have made good progress on 9. Submit the following via `websubmit`, under "compsci316" and "proj-ms2":

- New assumptions, E/R diagram, and list of tables (if they have changed since Milestone 1).
- A brief description of the platform you chose in Task 6.
- Changes you made to the database during performance tuning in Task 8, e.g., additional indexes created.
- A `.zip` or `.tar.gz` archive of your source code. At this point, your source code directory should at least contain:
    - A `README` file describing how to generate the "production" dataset and load it into your database. Do not submit the production dataset itself through if it is too big; instead, submit the URL where you download/scrape the raw data (if applicable), and the code that extracts and transforms (or generates) the production dataset.
    - A file `TEST-PRODUCTION.SQL` containing the SQL statements you wrote for Task 5. You may wish to modify some queries to return only the top 10 result rows instead of all result rows (there might be lots for large datasets).
    - A file `TEST-PRODUCTION.OUT` showing the results of running `TEST-PRODUCTION.SQL` over the production dataset. If you do your development on your virtual machine, you can create the file by running:
      `psql` *`dbname`* `-af TEST-PRODUCTION.SQL > TEST-PRODUCTION.OUT`
      where *`dbname`* is the name of your database.
    - Code implementing a simple but working database-driven web application on your chosen platform, which can serve as a starting point for completing your project.

Project Demo. At the end of the semester, you will need to present a working demo of your system. Instructions on how to sign up for the demo will be given during the second to last week of the class. Prior to your demo, submit the following via `websubmit`, under "compsci316" and "proj-final":

- A project report, including a brief description of your application, the E/R diagram for your database design, assumptions that you are making about the data being modeled, and the list of database tables with descriptions.
- A `.zip` or `.tar.gz` archive of all your source code. The source code directory should also contain a `README` file describing how to set up your servers and database, and how to compile and deploy your application.

"Open" Course Project

The open option is a chance for you to build something that you really want, provided it is related to data management. You need to write a detailed project proposal, and the course staff will work with you to ensure that your project meets the minimum requirements of depth and scope. You are encouraged to build novel systems and tackle challenging problems. Your "risk factor" will be considered in grading. Because of limited time, it is important to stay focused and ensure that certain pieces of your project are completely done; it is difficult to judge a project if nothing works.

Before settling on an idea and submitting a proposal for Milestone 1, you must speak to either the instructor or the TA about your project to obtain initial feedback.

Milestone 1. Submit a project proposal via `websubmit`, under "compsci316" and "proj-ms1." The proposal should contain:

- A description of the problem you wish to solve or the application you wish to develop, and, more specifically, what you plan to demonstrate at the end of this project.
- How it is important, interesting, and/or useful.
- Initial thoughts on how to approach the problem or build the application, including the preliminary system architecture and the platform you plan to use.
- Survey of previous and/or related work and systems, including discussions of how they relate to your problem as well as their limitations and/or flaws.
- A brief summary of your discussion with the instructor or TA (which is required before submitting the proposal).

The instructor will let you know whether the proposed project is acceptable.

Milestone 2. Submit a project status report via `websubmit`, under "compsci316" and "proj-ms2." The report should contain:

- Changes/updates to your original proposal (if any).
- Summary of progress so far, e.g., components built, tasks completed.
- A list of tasks to be completed before the final due date.

Project Demo Period. At the end of the semester, you will need to present a working demo of your system. Instructions on how to sign up for the demo will be given during the second to last week of the class. Prior to your demo, submit the following via `websubmit`, under "compsci316" and "proj-final":

- A self-contained project report, including:
    - The problem description, motivation, and survey of related work as in the project proposal, but more detailed and refined.
    - An in-depth discussion of your system, including the design choices you made.
    - Detailed description of any new approaches or algorithms that you are developing.
    - Evaluation of your system, and if applicable, comparison with competing systems. Be clear about what your evaluation metric is. If you have experimental evaluation, describe the experimental setup in enough detail so that others can repeat your experiments.
    - Any open issues or directions suitable for future work.
- A `.zip` or `.tar.gz` archive of all your source code. The source code directory should also contain a `README` file describing:
    - A brief overview of how your code is structured.
    - How to compile, set up, deploy, and use your system.
    - Any limitations in your current implementation.

"Standard" Project Ideas

Below is a list of possible project ideas for which high-quality datasets exist. Of course, you are welcome to come up with your own.

Entertainment, sports, or financial websites. Examples include those that allow visitors to explore information about movies, music, sports, games, stocks, etc. There are already many commercial offerings for such purposes. While there is less room for innovation, there are plenty of examples of what a good website would look like, as well as high-quality, well-formatted datasets. For example, *IMDb* makes their movie database available (http://www.imdb.com/interfaces); historical stock quote can be downloaded and scraped from many sites such as Yahoo! and Google Finance. This project is well-suited for those who just want to

learn how to build database-backed websites as beginners. You can always spice things up by adding features that you wish those websites had (e.g., different ways for summarizing, exploring, and visualizing the data).

Websites providing access to datasets of public interest. If you are interested in doing some good to society while learning databases, this project is for you. There are many interesting datasets "available" to the public, but better ways for accessing and analyzing them are still sorely needed. Here are some examples:

- Data.gov (http://www.data.gov/) has a huge compilation of data sets produced by the US government.
- The Supreme Court Database (http://scdb.wustl.edu/data.php) tracks all cases decided by the US Supreme Court.
- US government spending data (http://usaspending.gov/data) has information about government contracts and awards.
- Federal Election Commission (http://www.fec.gov/disclosure.shtml) has campaign finance data to download; their "disclosure portal" (http://www.fec.gov/pindex.shtml) also provide nice interfaces for exploring the data.
- GovTrack.us (http://www.govtrack.us/developers) tracks all bills through the Congress and all votes casted by its members. The Washington Post has a nice website (http://projects.washingtonpost.com/congress/112/) for exploring this type of data (in predefined ways), but you can be creative with additional and/or more flexible exploration and analysis options.
- The Washington Post maintains a list of datasets (http://www.washingtonpost.com/wp-srv/metro/data/datapost.html) that have been used to generate investigative news pieces. Most of these datasets hide behind some interface and may need to be scraped. Use this list for examples of what datasets are "interesting" and how to present data to the public effectively.
- National Institute for Computer-Assisted Reporting maintains a list of datasets of public interest (http://www.ire.org/nicar/database-library/). Use this list for examples of what datasets are "interesting"—they are generally not available to the public, but there may be alternative ways to obtain them.
- Google Fusion Table (http://www.google.com/fusiontables/Home/) hosts quite a number of datasets of public interest. It is a good place to find datasets or data sources to work on, and you can consider using it as a method of hosting your data for public access.

Your task would be to take one of such datasets, design a good relational schema, clean up/restructure the data, and build a website for the public to explore the dataset. If you are interested in this line of projects, discuss your plan with the instructor. As mentioned in class, Jun has a project on "computational journalism," so he will be happy to work with you on your efforts. Some of the datasets pose significant challenges in cleansing, analysis, and visualization; you may also consider an "open" project option to focus on these challenges.

"Open" Project Ideas

Here are some "open" project ideas. Some are very open-ended, and you need to narrow down their scope further. Some are not directly related to the materials covered in the course, and you will need to do a fair amount of research and reading on your own. Most ideas below can become Graduation with Distinction projects and Research Independent Study courses.

A number of ideas below are related to *computational journalism*, a project that Jun is working on. The decline of traditional media has led to dwindling support for the watchdog function of journalism, which is key social concern right now. The goal of computational journalism is to leverage the power of computing to

lower the cost, raise the effectiveness, and increase the participation of investigative journalism. See http://db.cs.duke.edu/projects/cj for more information.

Websites providing access to datasets of public interest. See the corresponding section under *"Standard" Project Ideas* for details. An open project would put more emphasis on tackling issues of data quality, analysis, and visualization that go beyond basic functionalities. The project idea below is one such example.

Quantitative fact checking. As one example of an advanced analysis/visualization feature that we want to provide for datasets of public interest, consider how to dissect supposedly "factual" and "significant" claims made over such datasets. Even if a claim "checks out," it may not be meaningful. Consider the statement "Lincoln Davis voted with Nancy Pelosi 94 percent of the time," which can be checked data from GovTrack.us. This statement may be true for a subset of the bills (say, those in the last session of Congress), but how would the percentage change if we look at a different subset of the bills (say, those over the last 2, 3, or 4 sessions, or only those "controversial" bills on which average Republicans and Democrats vote differently)? It would be nice to visualize the change in percentage when we vary the claim's parameters. In this case, the visualization may be a line plot over the duration of comparison, with a second line for only the controversial bills. Another dimension you can vary is the person being compared. Instead of showing the result for Davis alone, what if we also compare every Representative with Pelosi? Where would 94% stand among all Representatives? Here, we can measure the significance of the claim by the number of Representatives who agree with Pelosi no less than 94% of the time. If there are many, then the claim is really not that interesting. It would be interesting to see a plot of how this number changes as we vary the percentage. Your task would be to develop automated analysis and visualization techniques to help measure "interestingness" of claims quantitatively over data.

There are also plenty of "lies, d—ned lies, and statistics" in sports. For example, did you know that "only 10 players in NBA history had more points, more rebounds, and more assists per game than Sam Lacey in their career"? Since there are nearly 4000 players in NBA history, "one of the 11" sounds impressive. However, we can in fact claim the exact same or stronger for 112 other players (that no more than 10 players dominate them in these three stats)! Again, your job is to develop automated analysis and visualization techniques to help find, evaluate, and possibly counter claims made from data.
Jun and a number of students are actively working on this project—if you are interested, speak to Jun so we can coordinate efforts to build something with bigger impact.

Analyzing News. Bill Adair, creator of PolitiFact.com and Knight Professor of Computational Journalism at Duke's Sanford School of Public Policy, shared several interesting project ideas with us. Can we write computer programs that automatically monitor news and press releases and to detect instances of 1) anonymous sourcing (e.g., "according to people familiar with the matter who asked not to be named"); 2) campaign promises (e.g., "a goal to lead the world in college graduates by 2020, and cut the growth of college tuition and fees in half over the next 10 years"); and 3) checkable claims (e.g., earlier example of Davis-Pelosi vote correlation for fact checking)? What features of the text (or surrounding text) are possibly indicative of these instances? Here is an interesting site that detects anonymous sourcing:
http://schaver.com/anonymous/; can you do better than that? What about (2) and (3), which are progressively harder? If computer programs fail, can crowd sourcing be used to help?

Alternatives to Amazon Mechanical Turk. While Amazon Mechanical Turk is a great place for largely anonymous workers seeking paid work, it was not designed for workers motivated by non-monetary incentives, and does little to promote development of lasting relationships between workers and requesters. An ambitious project would be to build an alternative platform for crowdsourcing with different design decisions. For example, can we build a more "social" crowdsourcing platform? Can we let workers and requesters build social network connections, and use the social network to help recruit workers (or to limit access to tasks)? Can we make it more rewarding for workers to promote tasks they like (especially those pertaining to public interest)? Can we also help connect those who are willing to donate money for public interest work? Can we design some kind of "virtual currency" as an alternative to the monetary incentive? Can the two coexist, for those who still want to be paid for their work? Can tasks be made more game-like?

Another possible project would be to research all current crowdsourcing platforms and solutions on the market today, try them out (as much as possible), compare them, and come up with ideas for improving/complementing them. You will need to design a suite of "benchmark" crowdsourcing tasks with which to evaluate these systems. A survey paper will be the result of this investigation.

RA v3.0. RA is a home-grown relational algebra interpreter Jun wrote a number of years ago specifically for teaching this course. It is currently being used at Duke and Stanford for teaching undergraduate database courses. RA works on top of an SQL-based database system. Each relational algebra query is first compiled into an SQL statement (or multiple statements) and then executed on the underlying database system. Although RA has gone a long way, it still lacks many features. For example, error messages are cryptic. Auto-completion of commands and relational operators doesn't always work, and it would have been nice to also have auto-completion of relation and attribute names. An assignment command, which allows the result of a relational algebra expression to be stored in a temporary relation, will be very useful in writing complicated queries. It would be more convenient if RA can run completely from within a browser, without requiring any download. Finally, instead of a command-line interface, RA could use a graphical user interface that allows users to build relational algebra expressions like trees by dragging and dropping relations and operations.