CompSci 316 Spring 2020: Course Project

100 points (20% of course grade)

Posted: Saturday, January 25

Important Dates

Project mixer (in class): Tuesday, January 28

Milestone 1: Thursday, February 20

Milestone 2: Thursday, March 19

Demo: April 21-25 (details on this later)

Final report submission: due by the scheduled demo slot; can be updated until April 26.

In addition, there will be private piazza threads for each project and each member needs to write a short update of what s/he did in a week on the project in that thread. More details below.

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, 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. Some of the “open” project ideas below could evolve into Graduation with Distinction projects for Computer Science majors, and the instructor will be glad to supervise continuation of successful projects as CompSci 391 (Independent Study) or CompSci 393 (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 respective sections. Only one member needs to submit through Gradescope on behalf of the entire project team; *make sure all members are added* to the submission. 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:

- 24 points for submitting the required work at three checkpoints (as a team);
- 6 points for posting weekly updates (as an individual);
- 40 points for completing the proposed work (as a team);
- 30 points for the quality of the work (as a team).

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 of at least *A*-, but *A* and *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 **5 person teams**.The project should be completed in 5-person teams. Any other team size requires explicit approval from the instructor (please make your request through Piazza private questions); team sizes below 4 or above 5 are strongly discouraged. Regardless of the team size, an equal amount of work is expected and the same grading scale will be applied. You are required to:

- In each milestone/final report, report each team member's effort.
- Between the Milestone 1 due date and final project demo date, post a brief weekly progress update for yourself.

Details of how to post these updates will be announced by email. By default, all members in a team will receive identical grades for the project, except the 6 points for posting weekly updates as an individual. 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 help.

- You are strongly encouraged to use a shared coding platform like **"git"** to work on the project together. For your report/proposal/design etc., you can use a shared editor like a google doc, or for "latex" documents (that is used for writing research papers), you can use online platform like overleaf.com where everyone can edit at the same time.

If you have a project idea, you can post a **piazza message** by giving a short description and the number of additional team members you are looking for. In the **project mixer class**, if we have time, you are also welcome to tell the class if you have an idea and recruit members. Even if

you do not have a concrete idea by the time of the mixer, you can say what domains and types you are interested in, and look for teams. We will try to have 1-2 rounds of discussions in student groups in the project mixer class so that you can talk to your classmates about project.

Platform Issues

To develop the project, you are encouraged to use the VM provided by the course. We will post some options of cloud-based VM soon. As examples, the course staff 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. 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 over 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? JavaScript or plain HTML? We encourage you to explore **Flask-based platforms** for website or app -- you can receive help and support from **Duke Colab** for this platform (guest lecture in the project mixer class). While you are welcome to use any platform as you wish, we might not be able to provide you support if you run into problems. 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.
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 electronically under "Project Milestone 1":

- A progress report containing:

    - 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 text file members.txt, listing the members of your team, and for each member, a description of effort and progress made by this member to date.
- 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. **Y**ou should have completed Tasks 1-8 and have made good progress on 9. Submit the following electronically under "Project Milestone 2":

- A progress report containing:
    - 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. Not much details are needed if you are using Flask-based platforms.
    - Changes you made to the database during performance tuning in Task 8, e.g., additional indexes created.
- A text file members.txt, listing the members of your team, and for each member, a description of effort and progress made by this member since the last milestone.
- A 5-mins video with a voiceover showing the operation of your (simple) website or the app in the current stage. You should show how the frontend works and what the user can do and what happens "behind the scene" -- how the request goes to the backend for a sample action in the frontend, how the backend operates and sends response back to the frontend. Your website or app at this stage should be able to complete at least one operation requested by the user in the frontend, complete it in the backend on the

dataset you are using, and return the result to the frontend. You should also say what you plan to improve or add in the final version.

- A .zip or .tar.gz archive of your source code. At this point, your source code directory should at least contain:
    - 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
    - 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.

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 electronically under "Project Final":

- A final 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 text file members.txt, listing the members of your team, and for each member, a description of effort made by this member throughout the semester, highlighting the effort since the last milestone.
- 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 the instructor asap if you want to explore or go over an open project, and then talk to your instructor/TAs about your project to obtain initial feedback.

Milestone 1. Submit the following electronically under "Project Milestone 1":

- A project proposal containing:
    - 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).
- A text file members.txt, listing the members of your team, and for each member, a description of effort and progress made by this member to date.
- A .zip or .tar.gz archive of your source code.

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

Milestone 2. Submit the following electronically under "Project Milestone 2":

- A progress report containing:
    - 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.
- A text file members.txt, listing the members of your team, and for each member, a description of effort and progress made by this member since the last milestone.
- A .zip or .tar.gz archive of your source code.

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 electronically under ""Project 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 text file members.txt, listing the members of your team, and for each member, a description of effort made by this member throughout the semester, highlighting the effort since the last milestone.
- 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 (https://www.usaspending.gov/) has information about and database downloads of government contracts and awards.
- Federal Election Commission (https://www.fec.gov/data/) has campaign finance data to download as well as 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 (albeit outdated) website (http://projects.washingtonpost.com/congress/113/) for exploring this type of data (in predefined ways), but you can be creative with additional and/or more flexible exploration and analysis options. Vote Smart (https://votesmart.org/) has a wealth of additional, useful information on votes, such as issue tags, synopses and highlights.
- Each state legislature maintains its own voting records. For example, you can find North Carolina's here: http://www.ncleg.net/Legislation/voteHistory/voteHistory.html. Some states provide records in already structured formats, but for others, you may need to scrape their websites.
- 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.
- Stanford Journalism Program maintains a list of curated transportation-related datasets (http://www.datadrivenstanford.org/).
- 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.

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. 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.

RA and SQL Debugging

For Homework 1, you probably used the relational algebra debugging tool (https://ratest.cs.duke.edu/) developed by Zhengjie Miao (http://www.miaozhengjie.com/), a PhD student in Computer Science. Under the hood, this tool solves a challenging constrained optimization problem to find the "smallest" subset of the test database that still shows the difference between two queries. Do you have ideas of making this tool better (in terms of

interface, functionality, or performance)? or help build a similar tool for debugging SQL queries? Or provide more intuitive explanations and how to fix a wrong query?

Explaining Query Answers

When you run a SQL query, compute some aggregates, and perhaps plot some graphs, you may find some answers surprising or interesting – e.g.,

- A Rank-5 grad school in CS raised more than double NSF funding than a Rank-1 school – why?
- Database researchers from industrial labs had a lot of publications in a top database conference (SIGMOD) until about year 2000, and then had a decline, whereas researchers from US schools had an increasing trend throughout – why?
- A prolific researcher in data mining suddenly had a drop in publications in one top conference in one year – why?

Sudeepa has a project called FIREFly (Formal Interactive Rich Explanations on-the-Fly) where with collaborator and students, she is building tools and techniques to answer such questions. For instance, one idea is using “intervention” – make changes to a database, if that changes your observation, that is a good explanation. E.g. if Researcher X was responsible for raising most of the funding in the Rank-5 school above, if we remove him from the database with all his funding, the difference in funding between two schools will decrease. One idea is “counterbalance” -- why did a researcher had a drop in data mining publications in a year? Probably because the researcher published more in databases that year or published in data mining more in adjacent years. Note that all these explanations are answered automatically by the system! There are several other problems under this project about (1) methodology for explanations, (2) algorithms for explanations and their efficient implementation, (3) building effective user interfaces, possibly using natural language processing (NLP) that would show and explain to the users why the explanations make sense, etc. If you are interested in this topic in general, contact Sudeepa.

Data Cleaning

Noisy data is everywhere (that does not satisfy the constraints/format, or has missing values), and data cleaning is a crucial step in almost any real application. Sudeepa is working on rule-based systems for data cleaning working like “SQL triggers” (to be covered in class) but doing more than triggers as well as how to measure repairs. For instance, can we build a system that can show how “noisy” the database is and how close it is to become a “clean database”. If you are interested in working on a project related to data cleaning, contact Sudeepa.

(If your group is interested in pursuing an open project, but not interested in the above topics, please contact Sudeepa as well.)