CompSci 316 Spring 2017: Course Project

100 points (25% of course grade)
Assigned: Thursday, September 15

Important Dates

Project mixer (in class): Wednesday, February 1

Milestone 1: Monday, February 27
Milestone 2: Monday, March 27
Demo: Between Monday, April 24, and Tuesday May 02

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. Many 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 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 respective sections. 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 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 **4-person teams**. *Any other team size requires explicit approval from the instructor; team sizes below 3 or above 4 are strongly discouraged (and will only be allowed by the instructor on a case by case* *basis in exceptional situations).* 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. *You are required to report each team member's effort and progress in the milestone and final reports.* 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.

Note: There will be a "**project mixer**" class on February 1 (Wednesday) where you can pitch your ideas by giving a short presentation on the project that you are considering and the no. of team members you are looking for. Students looking for projects can talk to the students giving presentations to see if there is a match. Even if you do not have a concrete idea by this class, you can still give a presentation saying what domains and types you are interested in, and look for team members. There will also be a few rounds of reshuffling of seats during this class so that the students can meet other students in class and exchange ideas.

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 cloud-based VM 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 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? 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 "proj-ms1":

- 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. You should have completed Tasks 1-8 and have made good progress on 9. Submit the following electronically under "proj-ms2":

- 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.
    - 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 `.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.
    - 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 electronically under "proj-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 course-staff about your project to obtain initial feedback. Please send an email to compsci316-staff@cs.duke.edu with a short description of your ideas, and the instructors and/or the TAs will get back to you. Note that it is important to receive an early feedback for an open course work so that the course staff can tell you whether your project is likely to meet the requirements of this class, which you should receive before you start working on Milestone 1.

Milestone 1. Submit the following electronically under "proj-ms1":

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

Milestone 2. Submit the following electronically under "proj-ms2":

- 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 "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 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 (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/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.
- 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.
- 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 (send an email to compsci316-staff@cs.duke.edu first), Sudeepa 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 ***FireFly (Formal Interactive Rich Explanations on the Fly)***, a project that Sudeepa is working on. As people engage in more and more discussions on big data, they continue to seek meaningful "explanations" for trends and anomalies in the graphs that are plotted from available datasets ("Why are two graphs similar/different?" "Why is a sequence of points increasing/decreasing?" "Why is there a sudden spike or dip in a graph"). The goal is to help the users understand such query results with fast, rich, and insightful explanations. See https://users.cs.duke.edu/~sudeepa/firefly.html for some research papers on this project.

Automatically finding meaningful explanations

Here, we start with some datasets (see ***"Standard" Project Ideas*** for some pointers), explore the datasets by running queries to find interesting or unexpected observations, and try to build algorithms and tools to help people understand those observations.

As an example, consider the "DBLP" database, which records publications in Computer Science (e.g. see this link for publications in ACM SIGMOD – a premier database conference - http://dblp.uni-trier.de/db/conf/sigmod/) and is a publicly available dataset in XML format (http://dblp.uni-trier.de/faq/How+can+I+download+the+whole+dblp+dataset). Actually, many of these datasets are in XML, csv, JSON, etc. form, that you need to parse and load into a relational database system in relational form. We did this task, ran some queries to explore the datasets, and found that SIGMOD papers from academia (US schools) kept increasing over the years as expected, but the papers from industry showed a trend like a "bell curve" – with a peak around late 90's and early 2000s. Now this is surprising given the continued trend in industrial research in databases and big data systems. However, with our basic explanation framework in the FireFly project, we were able to automatically find some interesting explanations, like some industrial research labs and their senior researchers, who contributed tremendously in the development of the foundations in database research and DBMSs during that time, but some of those labs are now shut down or have a hiring slow down (so far, the interpretation has been manual). Also, we found some relatively "new" academic database research groups, who were founded later but were highly prolific since early 2000s, and thus contributed to this change. Therefore, the generic question is how we can formalize the explanations that contribute to a "change", and how can we find the ones with high impact. How can we make these explanations "rich" (by extracting deep insights from the data), "efficient" (search from the huge search space of explanations), and "meaningful" (that is easily understandable not only by the experts but by a large group by users, say by providing explanations in natural language form). Speak to Sudeepa if you are interested in this topic and she can give you some further pointers and specific project ideas.

Visualizing query answers for easy exploration

Before you explain a question from a user, a natural (and orthogonal) step would be to help the user ask a question. For this purpose, we need efficient, effective, and easy to use data visualization and data exploration tools. You may have heard of data visualization software like Tableau that helps users by providing sophisticated and easy to understand views of data. However, the need for visualization and exploration tools for understanding data and query answers still remains. As a toy example, suppose the user is running a group-by query to get some aggregates, and sorting the results with an order-by clause.

If the query looks like this:

```
SELECT A, B, sum(D) as c
FROM R
GROUP BY A, B
ORDER BY A, B
```

You can plot a barchart: where c is on the y-axis, A = a1, a2, a3, … values are on the x-axis, and for each value of A, you can plot a bunch of bars for the values of B. (For instance, A can be names of companies and B can be locations or the year, where c outputs the total sale.) The question is, if you have a query with three or more attributes in the group-by clause, then how would you plot it? Do you use a 3D plot (which is visually bad)? Then what about 4 attributes? Can you develop a user interface that nicely plots arbitrary group-by queries and help the user visualize the results? If the result set is huge, how can you detect a set of "top-k" results that you should first show to the user? How can the user interactively explore the results without getting overwhelmed with all the information? And of course, efficiency of these systems is a big concern since the exploration has to be done interactively in real time. Sudeepa can give you some pointers on related work if you are interested in this line of research.

Building tools for analyzing performance in cluster computing frameworks (like Spark)

There are two research projects that a graduate student, Sudeepa, and other faculty members are working on. More details on these projects will be sent over emails. If you are interested in learning more about "big data processing systems" and working on this topic (which is mostly out of scope of the material covered in this class), but did not receive the email sent to the class, contact Sudeepa.