Semester-Long Project

CS 364

For this project assignment, you will be submitting—either as an individual or as a pair—a project requirements document for the semester-long project component for this class. This document will outline the scope/functionality of the project, the associated database, and envisioned stakeholders. It will also outline basic expectations for the project, including your initial project document and final deliverables.

The Project

For the remainder of the semester, you will be working on a class project of your choosing. This document outlines some guidelines on some requirements your project should fulfill, but most of the project is left up to you. You will choose a domain to design an application for, and specify particular pieces of functionality you anticipate your application providing. The domain is entirely up to you[1], although my ability to help you make decisions on the scope of your project within a particular domain will vary depending on my familiarity with the domain. The scope of your project is also up to you, provided it fits within the requirements below. You are free to go above and beyond these requirements if you would like, but keep in mind that while I enjoy seeing your creativity on display with these projects, you will only be graded on those components outlined below (and explicitly attached points in the rubrics at the end of the document). Your project may also be written in any programming language you would like; you will not be submitting code to me, so you only need to ensure that your language of choice can connect to a relational database, and that you can run your program on your machine.

Grading for the project is spread across three different components: an initial project document, a project demo at the end of the semester, and a final project document.

The Requirements

Before we dive into the requirements, an important public service announcement:

This is not a CS 341 project!

While you probably knew that, I have found that students struggle with separating the goals of this project from what they might have done in other classes, like CS 341. Oftentimes, students will get to the final demo for this project and show me a great application - good software engineering, beautiful UI, error checking - but missing many of the core CS 364 project requirements, like advanced queries. As you work through your project and prioritize your time, please see the rubrics at the end of this document for how you will be graded.

For full credit, your project should satisfy the following requirements:

- Your application should demonstrate the ability to connect to and interact with a relational database. Note that while there are other popular database engines available (e.g., Mongo DB), you should be sure that you are working with a relational database, and not some other paradigm (e.g., NoSQL). Common acceptable options for this class are SQLite, Microsoft SQL Server, and MySQL. Check with me if you have questions about whether a particular database is relational.
- Your database is required to have at least five tables, at least three of which must be entity types. Not all of these tables must be entity types, as some tables might be representative of

[1] Within reason - you should be willing to give a presentation to strangers about whatever it is you choose!

relationships. Your description of the database in your document should be sure to identify which of those relationships in the ER diagram will manifest as tables.

I will occasionally see ER diagrams for this project that are made up entirely of one-to-one relationships. Such a database design is **not** utilizing relational databases well. Rather, it is essentially a spreadsheet, or NoSQL. Your database should primarily be made up of one-to-many and/or many-to-many relationships.

- Your database should have data in it. The data may be either real data that you have somehow imported, or fake data you developed for the purposes of this project (Mockaroo[2] is a great resource for this!). You should have enough data to demonstrate that your advanced queries (described later in the document) work. A good litmus test is that, for any given advanced query, is there enough data such that your query includes some pieces of data, but excludes other pieces of data. For example, if you had a database of students with a query that included `GROUP BY LastName HAVING count(*) > 5`, you should have multiple last names in your database (to demonstrate the ability to form multiple groups), and some of those groups should have more than five tuples, while others have five or fewer tuples.
- Your application should demonstrate the ability to add, update, and delete data. You do not need to include this functionality for every possible table. Rather, you could include all three types of modification for a single table, or you may apply each type of modification to a different table.
- You should have at least three pieces of functionality (specifically highlighted and described under "Functionality" in your document) that requires something beyond basic `SELECT-FROM-WHERE` syntax. Your description should make it explicit to me how it will employ additional SQL syntax (e.g., describing a piece of functionality and then noting that it will require a subquery); I should **not** have to speculate about how a piece of functionality will fulfill an advanced query! You must have at least one query that fits into each of the following categories (i.e., utilizing at least one of the pieces of syntax from that group):
  - **Group 1**: aggregate functions, `LIKE`, `GROUP BY`, `ORDER BY`, `LIMIT`
  - **Group 2**: `HAVING`, `OFFSET`, outer join, joining four or more tables
  - **Group 3**: subqueries, `IN`, set operators, any additional functionality outside of what was discussed in class will likely fall into this category (but please talk to me first if we did not cover this in class)

  To be clear, you must have three separate advanced queries - you cannot write one query that fulfills all three groups and have it count for all three. On the other hand, you might have three queries that all use `GROUP BY`, `HAVING`, and a subquery - one of those queries would be used to fulfill group 1, another to fulfill group 2, and a third to fulfill group 3. Note that your advanced query cannot be written in a convoluted way in order to fulfill one of the advanced query requirements; basically, if I can find a simpler way to write the query without the advanced functionality, you will not get credit for that query. Consider a database of course offerings across semesters (i.e., the same course will likely appear many times). If you wanted to write a query to find unique courses, an easy way would be the following:

```
SELECT DISTINCT Course.Name
  FROM Course
```

[2]Website: `https://www.mockaroo.com/`

But a more convoluted approach that would satisfy group 1 or group 2 requirements would be the following:

```
SELECT Course.Name
  FROM Course
  GROUP BY Course.Name
  HAVING count(*) > 0;
```

This second query would be considered unacceptable for fulfilling any of the advanced queries.

For pairs only: Pairs are required to have **six** pieces of functionality that require advanced queries - two for each group. For the two queries for groups 1 and 2, they must use **different** syntax from that group. For example, you might have one query with aggregate functions, and another with `LIKE`; these two queries would each fulfill the group 1 requirement for a pair. However, if you have two queries that both use `LIKE`, only one of them will count for fulfilling the group 1 category for a pair. At your demo, you will each be expected to describe one query from each category, e.g., partner 1 discusses queries 1, 2, and 3 that fulfill groups 1, 2, and 3, while partner 2 discusses queries 4, 5, and 6 that also fulfill groups 1, 2, and 3.

- Interactions (i.e., insert, update, delete, queries) with the database that are fulfilling the above requirements should **not** be hardcoded. In other words, the user should need to provide some information via the interface that is then filled in on the query in order to enable the query to run successfully. For example, if you have a database of students, you should not simply have a button that deletes the student named "John Doe". Rather, the user should need to do something in the interface (e.g., provide an ID number, select a student in a list) to dynamically construct the query to delete some student. Another way to look at this is that you should have queries in your code that use the wildcard character, e.g., 'SELECT * FROM Students WHERE LastName = ?'.
- There should be a user interface to your application. This could be a console-based application or a GUI. See the rubric at the end of the document for more information on point values attached to the user interface component.

See the grading rubric at the end of the document for further elaboration on gradations of fulfilling these requirements and their relative weights.

Document Requirements

You will turn in two documents related to your project - an initial document at the onset of the project, and then a final document at the completion of the project that is due the last day of class. Further details on each are below.

Initial Document

Please title your document with a descriptive name[3]. Be sure to list everyone on your team beneath the title. Your document should be divided into the following sections:

- **Synopsis**: Develop a 250 word abstract that you are basing your project off of.
- **Database**: Begin with an ER diagram of your envisioned database. Below this, describe each table that you will have in your database: what is its purpose, why you are storing those particular attributes, how the data will be used, etc. **You must have both an ER diagram and a description of tables/attributes.**

[3]Clever name optional, but always appreciated!

- **Functionality**: In **itemized form**, delineate what your group envisions as the main functionality of the application. This section should also outline the advanced queries you envision having in your database; see the above section for more details.
- **Stakeholders**: Describe who you envision as the user(s) of your program and how their use of the system might differ.
- **Technological Requirements**: Briefly discuss the platform you envision implementing on (e.g., desktop application, mobile, web), the language(s) you anticipate using, the database platform you anticipate using, the connector you anticipate using (e.g., JDBC), how you plan to share code (e.g., email, GitHub), and what experience your team (or yourself) has with these components. In particular, if you are planning a website, you should have a discussion of the technology needed and whether anyone has experience with it. You are not bound to the decisions made here, but I would like to see some thought as to whether your chosen platform and language are feasible for your group.

Your initial project document is due on the day specified by the Canvas dropbox at 11 PM. Please submit your proposal as a single PDF document titled with your last name (separated by dashes if you are on a team, in alphabetical order); only a single person from your group needs to submit the proposal.

Final Document
Your final document should include updated sections for everything in your initial document detailing what your project entailed. Additionally, you should include two new sections: one titled **Screenshots** that includes two or three screenshots of your project, and one titled **Advanced Queries** that includes all of the SQL for your advanced queries. For each advanced query, you should also include a) which group it fulfills, and b) a brief description of what the query does (e.g., "This query finds last names that are shared between five to ten students.").

Your final project document is due on the last day of the semester in a dropbox on Canvas. Please submit your proposal as a single PDF document titled with your last name (separated by dashes if you are on a team, in alphabetical order); only a single person from your group needs to submit a proposal.

Grading
Documents will be graded on the database design, whether the design meets the functionality criteria outlined in the document, adherence to ER diagram standards, whether the specifications outlined in the document and database requirements are met, whether the document is well-written (i.e., no grammar/spelling mistakes), and whether the document is presented in a professional fashion (e.g., submission format, organization). There are example project documents on Canvas that demonstrate what a professional document looks like. Note that these are from earlier iterations of the class, when project requirements may have been different; do not consider these to be examples of a good project, only a good project document. I will also provide feedback on the scope and technological considerations.

Demo Requirements
You will sign up for a 15 minute demo slot with me during the last one to two weeks of the semester[4].

During the demo, you will walk me through your project in whatever order you see fit, and I will ask questions as you go to ensure I see everything necessary to asses your project for a grade. In

[4]The number of demo slots will be determined by the number of individual vs pair projects, as well as the number of students in the class.

general, you will need to demonstrate each of the different components listed as part of the project requirements. See the rubric at the end of this document for more information.

Further Notes, Checks, and Pitfalls

Tools

Although we are learning about SQLite and JDBC (associated with Java) in this class, you are free to use other relational database implementations, database connectors, and programming languages. Additionally, your team might choose to use some sort of code repository system to share code. Popular options include GitHub[5] and Bitbucket[6], both of which have free versions available.

GUIs tend to be a pain point for students, which is understandable. Note that a console-based application is fine for a very slight reduction in points earned. Websites are a common approach, if you are comfortable with web design. Students also report using a number of different GUI builder tools, such as the WindowBuilder plug-in for Eclipse.

Planning for the Semester

It can be daunting to tackle a project of this scope, with or without a partner. While you are free to organize completion of your project in any way you like (e.g., a different order than I suggest, waiting until the last week to do everything), below is a suggested ordering:

- Complete the initial project document first!
- Rather than wait for feedback on your document, try setting up and testing the tools/technologies you plan on using. For example, if you'll be using a repository manager like GitHub, make sure everyone has set up accounts and try sharing some code. If you are going to be doing a website, set up the tools you need for that. This will allow you to be productive independent from the specifics of the project.
- Sketch out the interfaces you will need in order to achieve the functionality outlined in your document. Maybe brush up on your GUI skills if they are a weak point, or find a tool to help you design a GUI.
- For pairs, consider how you might split up the work (e.g., interface design, database implementation) to play to people's strengths. How will you split up work such that you can work on components independently but still be able to combine them? Will you check back in weekly to make sure you're both still on track? When will you combine everything?

Weekly Check-Ins

Once project documents are submitted, by each Monday evening, I would appreciate a single email from you, or one member of your pair (with your team member CCed) split into four sections: progress made in the past week, goals for the next week, any problems encountered (either in the past week, or envisioned in the future work), and questions for me (even if it is none). An action plan for resolving problems should be discussed in the problems section as well. While these are not required (i.e., you will not be graded on them), consistently emailing me check-ins can help me to understand your progress throughout the semester, particularly in the event of unexpected setbacks that might hinder your grade at the end of the semester.

Group Work

Group work is an important part of working in industry. Some of the most valuable skills you will learn in pursuit of your degree are not graded and are not the focus of classes—communication (both within your pair and with me through documents), professionalism, dividing and combining

[5] Website: `https://github.com/`

[6] Website: `https://bitbucket.org/`

work, and so on. You should consider this project an opportunity to work on these skills, and I highly encourage you to seek out a partner. Our graduating seniors regularly report that they wish that had more opportunities for teamwork; I also recognize that teamwork can be a challenge in a school setting given individual circumstances, hence the choice.

In the event that there is a problem within your pair, I encourage you to find a time to talk to me, either during office hours or by setting up an appointment. Although grades earned for the project portion of this class should be the same across team members, I reserve the right to adjust grades for individual components of the project—up to and including the whole project—for individual members based on behavior and contribution. Note that this is reserved for extreme cases. While I do encourage you to bring problems to my attention, I may choose not to modify grades unless the offense is egregious.

Keep scrolling for rubrics
for each graded component of the project!

Appendix

Initial Document Grading Rubric (11 points)

	3 points	2 points	1 point	0 points
synopsis (x 0.5)		Gives a concise synopsis of the project.	Synopsis is present, but not concise.	No synopsis.
database (x 1)	ER diagram is correct, fulfills the needs of the application, and there is a description of the tables/attributes.	One of the items from the previous column is missing.	Two of the items from the previous column are missing.	No table/attribute descriptions or ER diagram.
functionality (x 1)	Functionality listed, including clear discussion of all required advanced queries and how particular pieces of functionality fulfill the add/update/delete requirements.	Functionality listed, but missing either clear discussion of advanced queries or clear discussion of add/update/delete requirements.	Functionality listed, but missing clear discussion of advanced queries and clear discussion of add/update/delete requirements.	No functionality listed.
stakeholders (x 0.5)			Stakeholders outlined.	No stakeholders outlined.
technological requirements (x 0.5)	Discusses 1) platform (e.g., desktop), 2) anticipated programming language and familiarity, 3) anticipated RDBMS and familiarity, and 4) code sharing mechanism (if applicable).	Discusses two of the three points (or three of the four if a group).	Discusses one of the three points (or one or two of the four if a group)	No discussion of technological requirements.
professionalism (writing) (x 0.5)		Writing is free from grammatical/spelling mistakes, and uses professional language.	Writing contains many grammatical/spelling mistakes, or does not use professional language.	Writing contains many grammatical/spelling mistakes, and does not use professional language.
professionalism (presentation) (x 0.5)		Presentation/formatting makes the document easier to navigate (e.g., clear section headers).	Presentation/formatting is usable, but not something that could be shown to a client.	Presentation/formatting make the document unusable.

Final Document Grading Rubric (12 points)

	3 points	2 points	1 point	0 points
synopsis (x 0.5)		Gives a concise synopsis of the project.	Synopsis is present, but not concise.	No synopsis.
database (x 1)	ER diagram is correct, updated, and has a description of the tables/attributes.	One of the items from the previous column is missing.	Two of the items from the previous column are missing.	No table/attribute descriptions or ER diagram.
functionality (x 1)	Implemented functionality listed, including listing SQL for advanced queries and add/update/delete requirements.	One of the items from the previous column is missing.	Two of the items from the previous column are missing.	No functionality listed.
stakeholders (x 0.5)			Stakeholders outlined.	No stakeholders outlined.
technological requirements (x 0.5)	Discusses 1) platform (e.g., desktop), 2) programming language used, 3) RDBMS used, and 4) code sharing mechanism (if applicable).	Discusses two of the three points (or three of the four if a group).	Discusses one of the three points (or one or two of the four if a group)	No discussion of technological requirements.
GUI (x 1)			Includes 2-3 representative screenshots of the project.	No screenshots included.
professionalism (writing) (x 0.5)		Writing is free from grammatical/spelling mistakes, and uses professional language.	Writing contains many grammatical/spelling mistakes, or does not use professional language.	Writing contains many grammatical/spelling mistakes, and does not use professional language.
professionalism (presentation) (x 0.5)		Presentation/formatting makes the document easier to navigate (e.g., clear section headers).	Presentation/formatting is usable, but not something that could be shown to a client.	Presentation/formatting make the document unusable.

Demo Grading Rubric (34 points)

For those projects which involve pairs, each advanced query will be worth half of what it is worth below. For example, advanced query #1 is worth 3 points for an individual, but half that, i.e., 1.5 points for pairs, since pairs will have two of advanced query #1.

	3 points	**2 points**	**1 point**	**0 points**
tables built/data entered (x 2)			Tables built with data.	No tables or data.
database connection (x 1)			Demonstrates a connection from an application.	No connection from an application.
functionality (x 1)			Demonstrates the ability to interact with the database.	Does not demonstrate the ability to interact with the database.
add/update /delete (x 3)	Demonstrates the ability to add, update, and delete data based on input.	Demonstrates two of the three items.	Demonstrates one of the three items or uses hardcoded input.	Does not demonstrate the ability to add/update/delete.
advanced query #1 (x 1)	Query fulfills group 1 and there is sufficient data in the database to demonstrate that the query works.	Query fulfills group 1 but there is not sufficient data in the database to demonstrate that the query works.	Query fulfills group 1 but has hardcoded input.	No query with necessary components from group 1.
advanced query #2 (x 2)	Query fulfills group 2 and there is sufficient data in the database to demonstrate that the query works.	Query fulfills group 2 but there is not sufficient data in the database to demonstrate that the query works.	Query fulfills group 2 but has hardcoded input.	No query with necessary components from group 2.
advanced query #3 (x 3)	Query fulfills group 3 and there is sufficient data in the database to demonstrate that the query works.	Query fulfills group 3 but there is not sufficient data in the database to demonstrate that the query works.	Query fulfills group 3 but has hardcoded input.	No query with necessary components from group 3.
GUI (x 1)	Well-designed GUI.	Basic GUI.	Command line or console interface.	No interface to the database.