Phase 1: Design Document (Due 2/17/2020)

The first phase of the project will consist of developing planning documents for developing a database management system. The deliverable for this phase is the design document. This design is only for the database, not the connectivity or the GUI or the applications you will develop to access the data.

Scenario

Your development team was contracted by Texas A&M University (TAMU)'s Sport's Management department to develop an application for use in an introductory course from their proposed class in Sports Statistics. Administrators from Texas A&M University's Department of Health and Kinesiology are in the works of creating a new Sports Statistics class, and they have contracted your development team to build an application for use in an introductory statistics course.

- The new class coordinators have requested the development of a database management system (DBS) and graphical user interface (GUI) for managing and querying data from the current college football data.
- Specifically, your team has been tasked with developing an application that will allow students to query names of Players, Teams, Conferences, individual games, hometowns, statistics, records or other items of interest. The queries would be used as input for generating answers to various trivia-like questions for class assignments.
- Decide on the data you will pull in from the data we will give you. Imagine there is much more data than you need. You will plan for what you need and populate your database from the data we provide.
- In order to prepare for this task, your team will develop planning documents consisting of a database design document.

Design Document

The first planning document is the design document. The purpose of this document is to plan the high-level design of your system prior to planning its low-level development (source: *Scott Hackett's "How to write an effective design document"*). This deliverable should be turned in as a group assignment. All members should participate equally. The document should consist of the following sections.

- **1. Executive Summary.** After completing the document, re-read and give a high level summary of what you are presenting.
- **2. Purpose of Project.** Describe what the system does.
    - Problem Statement: Describe what problem it is trying to solve.
    - Relevance: Describe why the system should exist.
    - Target Users: Describe who will use it.
- **3. Needs Statement.** Describe the specific need your product will address.
- **4. High-Level Entity Design.** Describe the high-level entities that form the major constructs of your design. In terms of a database, the high level entity would be a table of tuples, or data records. It is estimated that a project of this size will likely need 10-20 high level entities.
    - Purpose: Describe in a few sentences what each entity does. Does not need to be verbose, just enough to explain its purpose.
    - Justification: Describe your reasoning for defining each entity.
    - System Role: Describe the role of each entity in the system.
- **5. Low-Level Entity Design.** Describe the interactions of the entities in detail and their corresponding relationships. This relates more to the interaction between entities and the transactions to obtain the desired information.

    - 5.1. Usage. For each entity, describe its usage and functionality. About a paragraph per entity.
        - Relationship: Describe the relationships for each entity.
        - Thought Process: Describe the thought process for why you defined each entity.
        - Benefits: Describe the benefit of each entity.
        - Risk: Describe the risk of each entity.
    - 5.2. Model. Provide a visualization and accompanying overview that models your system.
        - Diagram: Create a UML diagram that models your system. It does not need to be perfect, but it should describe what exactly is happening in the diagram. This should not be hand drawn.
        - Description: Write a text description that overviews the UML diagram.
    - 5.3. Interaction. Provide a visualization and accompanying overview that describes the system interactions.
        - Diagram: Create an interaction diagram that shows how the entities communicate with each other to perform a complex task.
        - Description: Write a text description that overviews the interaction diagram.
- **6. Expectations.** Describe the list of benefits, assumptions, risks, and issues related to your system.
    - 6.1. Benefits. Describe the list of benefits in multi-paragraph form.
    - 6.2. Assumptions. Describe the list of ALL assumptions in multi-paragraph form.
    - 6.3. Risks. Describe the list of ALL risks in multi-paragraph form.
    - 6.4. Issues. Describe the list of ALL issues in multi-paragraph form.

Phase 1 Grading

Points		**Deliverables**
100	**Design Document**	
	10	Executive Summary
	10	Section 1. Purpose of System
	5	Section 1.2 Needs Statement
	30	Section 1.3 High-Level Entity Design
	30	Section 1.4 Low-Level Entity Design
	15	Section 1.5 Expectations

Phase 2. Database System (Due 2/24/2020)

The second phase of the project will consist of building the database by downloading the IMDb and inserting the data into the relations you designed in Phase 1. Your goal will be to implement and verify the validity of your design.

- **Input:**
  - ~~IMDb from class web page or from IMDb.com. (https://datasets.imdbws.com/)~~
  - Load table data into your database. Instructions on importing JSON data will be given in lab.
  - Use PostgreSQL and SQL commands to verify your data.
- **Output:**
  - Create a Database Verification document that both verifies and captures the completeness of your database. (Deliverable 1)
    - Show a list of relations as output by PostgreSQL (use \d)
    - Show the size of each table with a SQL command.
    - Show at least 10 lines from each table with a SQL command.
  - Build an input file of at least 20 SQL commands that pull data from one table, pull data and aggregate that data, queries multiple tables, uses SQL operators, and creates a temporary view. Capture this file and format the output as if it were an appendix to a design or review document to be created in the future.(Deliverable 2)

Phase 2 Deliverable #1. Database Verification

Your DBMS will be evaluated on the correctness and closeness to the design document. If changes had to be made at this point, the changes should be documented here. This document is a group effort and is mostly output from PostgreSQL. This can be used to verify the high level design from your document.

Phase 2 Deliverable #2. Query Correctness

Your Database system will be evaluated on the correctness of its DBL queries. You are to build 20 SQL queries that can be run as an input file. Capture the output, adding headings, comments, the original commands. Format this as if it will be an appendix to a design or verification document. This can be used to verify the interactions and low level design from your document. These queries should have some examples of a desired question that can be answered through this query.

Phase 2 Deliverable #3. Process and Contribution (approx. 250 words)

Write a memo that describes your team's process in getting the data into the format and design you chose and include your contribution. One per team member. The memo is to the instructor and TA's. If you changed the design of your database (additional relations, additional attributes, changes in relationships, etc.) please describe in detail the changes and then the justification for this change. This is above the listed word count.

Phase 2 Grading

Points			Deliverables
100	**Phase 2. Database System**		
	35	**Database Verification**	
		10	Document writing style, layout, and correctness.
		10	Required verification output.
		15	Compliance with design document. Documentation of variance
	35	**Command Correctness**	
		10	Document writing style (Appendix), layout, and correctness.
		10	Input file, creativity and completeness of SQL commands.
		15	Capture and relevance of data displayed.
	30	**Process and Contribution**	
		10	Memo format, writing style.
		10	Process description.
		10	Contribution description and added value.

Phase 3. Graphical User Interface (Due 3/2/2020)

The third phase of the project will consist of developing a graphical user interface (GUI) for interfacing with your DBMS. The details are listed below.
Before you start, you might want to read this short, but good article on incremental development:
https://levelup.gitconnected.com/code-less-think-more-incrementally-98adee22df9b

Phase 3 Deliverable #1. Database GUI Sketch

You will deliver a PDF drawing of the initial look of your user interface.
Use a tool to design the initial prototype of your GUI to the project Database. Create a title and the overall look and feel of your final product. Save the bottom ½ for future development. There are many free tools like framebox.org (and many others) that allow you to develop and save a prototype. The initial design should allow for several simple queries like movies by (actor, director, year, etc.) or simple joins like movies with actor A and Actor B. You should include input to record if the user would like to see the output in a text box or write to a file on your local machine. You should turn in a PDF of the output of your tool. Hand drawn sketches are not acceptable. Additional notes or comments can be included in this document. It should be written as if it is a page in an engineering notebook.

Phase 3 Deliverable #2. Validate the GUI

Using Java, complete with JDBC from your local machine, code the user interface designed above. In lab, show your working connection from your local machine to your database on db-315.cse.tamu.edu with your initial GUI. Have your original GUI sketch with your team's name, Individual's names and UIN's. Demonstrate the queries. Turn in the sketch after the demo.

Phase 3 Deliverable #3. Process and Contribution (approx. 250 words)

Write a memo that describes your team's process in getting the design on paper (or digitally) and then how you went from design to implementation. Include your contribution. One per team member. The memo is to the instructor and TA's. If you are unfamiliar with how a memo looks, please look up the format. Again, include additional (not previously documented in phase 2) changes you made to your original design of the database.

Notice:

Phase 3 WILL NOT give you a final product. It will not even give you a complete graphical user interface. By the end of Phase 3 you will have a platform that has all of the back end working, with proven connectivity and a basic user interface that shows this. It will be the start of the front end with a graphical user interface, but it will not be the complete product. You will update the look and functionality of your product in Phase 4 based on the requirement released at that time.

What you will or should get is:

- The confidence that you have connectivity from your machine (laptop or desktop, windows or Mac) to the database you created that resides on db-315.cse.tamu.edu. Your Java app should be able to run on your local machine, connecting to the database through JDBC.
- A basic, but fully functioning graphical user interface that will allow simple searches on Actors, Titles, Directors, Writers, Year of release, etc. as well as the union or exclusion of these.
- You should be able to query the data and if small, display it to the screen, but if large, offer the option of output to a file.

Phase 3 Grading

Points			Deliverables
100	**Phase 2. Database System**		
	40	**Database GUI Design (Sketch and comments)**	
		10	Document writing style, layout, and look and feel of GUI.
		10	Required sketch output.
		15	Inclusion of required features
	30	**GUI validation**	
		10	Java file compiles, runs in a GUI not in a shell.
		10	Database connects opens and closes from remote computer.
		15	Features are at least those in sketch.
	30	**Process and Contribution**	
		10	Memo format, writing style.
		10	Process description.
		10	Contribution description and added value.

Phase 4. Large Dataset Processing (Due 3/16/2020)

The fourth phase of the project will consist of completing the tool with the final functionality described below. To finalize the project, you will turn in your final source code, demo that code in Lab, and write Retrospective document and have a Retrospective meeting in lab or agreed outside time.

Phase 4 Deliverable #1. Large Data Processing Task

At this point in the project, it is expected that you have a fully functional prototype GUI that connects via JDBC to the database you created on db-315.cse.tamu.edu. This database is populated with data from the IMDb web site or the JSON data set provided. You will add the following functionality that will require you to develop a strategy that can arrive at the desired solution. The goal is to understand how the information pulled from multiple queries can be combined to solve for questions that cannot be answered from simple SQL commands. Also note that with a dataset that is too large to bring into memory and solve with standard search algorithms, you may have to make assumptions or multiple attempts at a solution. **Teams with 4 members must do at least one bonus to get full credit.**

- Question 1: Give 2 actors, List the shortest list of movie titles that connects the 2. (The x degrees of separation problem) (Bonus: Be able to exclude a particular actor in the chain) Example: If I enter Elvis Presley and Kevin Bacon, you output could look like:
  - Elvis Presley, Edward Asner, Change of Habit (1969)
  - Edward Asner, Kevin Bacon, JFK (1991)
  - There are 2 degrees of separation between Elvis Presley and Kevin Bacon.
- Question 2: Given 2 years, be able to list the shortest list of (Directors, or Actors) that had movies released. Be able to exclude names as well. (or run again, finding another answer) Example: 2005 - 2015
  - Robert Di Niro (one answer of many) then output the list of films. (he actually had 30 in this time frame)
- Question 3: Team Choice: show a solution to a question a student or film buff might ask that cannot be solved with a simple query. This should require using the results of one portion of the database to query somewhere else in the database and result in an answer or answer set that is of interest. This would be an item that sets your tool apart from the competition.
- Question 4: (Bonus) Given a genre, seed year, favorite actor, suggest 5 other movies to watch that do not include the favorite actor but have another reason besides genre and year in common.
  - Terminator, 1984, Arnold Schwarzenegger is my input. Rambo, 1982 Sylvester Stallone could be in the list of possible suggestions. The output should also contain the connection (common costars, common directors, etc.)

This deliverable is 2 parts, 1) The code will be kept in Git throughout the project, but the final *.java files will be put on eCampus to evaluate style. 2) You will demonstrate the functionality in Lab.

Phase 4 Deliverable #2. Group Retrospective Survey and Document

There are 3 parts to this document. It can be contained in one file with page breaks, or 3 files. Everything should be PDF files. These documents will be a group effort. Each team member should contribute. The same documents can be turned in for each member in PDF format. Each member must turn in a document to get credit.

At the conclusion of a project, one of the first things to do is complete a retrospective survey. The survey goes out to the full project team. The goal of the survey is to gather team feedback while the project is fresh. Typical questions are focused on project satisfaction, the client and industry, and the work and results:

- Was the project fun and interesting?
- Did it provide opportunities to innovate?
- What went well? What didn't go well?
- What lessons did you learn that you would share with your team?
- What one topic do you want to make sure we address in the retrospective meeting?

We use the survey responses to determine discussion topics for the meeting (especially the last one!) and also to create the retrospective document. The raw survey responses are usually only seen by the project's PM. In this case share all the data. Collate the results as your first document.

Our retrospective document is the lasting documentation from this process. It's written by the Team and pulls in data from our time and effort spent together, and the team survey responses. Create a title page with credits to the team and the document will consist of a few sections:

- Project overview (A summary of the work and the team)
- Engagement analysis (How was the engagement as a whole? Was the team satisfied? What were the challenges? Did we have the opportunity to innovate?)
- Product analysis (How was our overall final product? What did we learn that we could apply to future work? How could we have done better?)
- Work analysis (Are we proud of what we accomplished? What went well? What could have gone better?)
- Key points to remember for future projects (What did we learn? What would we do differently the next time? What lessons would we share back with our individual teams? The class as a whole?)
- Work effort analysis (How'd we do our time together and time individually? The timeline? What was our breakdown of task per member and how did that differ from our initial estimates?)

This serves as a great way to think critically about a project and also capture a snapshot of the project as a whole. Typically the retrospective documents are shared with the entire company and often refer back to them when starting new projects with similar traits to previous ones. Finally, when the document is complete, review it and create an agenda for a final retrospective meeting.

Phase 4 Deliverable #3. Retrospective Meeting Memo

This memo is written to the TA’s and the Instructor. (In industry, this usually goes to the Project manager and Development Manager). Team members should be copied on the memo. Each team member should write their own memo looking at the project from their own view and experience, but include thoughts on the retrospective meeting. The meeting should attempt to follow the items listed in the agenda previously created. The document should be turned in in PDF format.

The retrospective meeting is our chance to get the entire team together to talk about the project. Everyone from the project team attend the meeting. Choose a few discussion topics in advance and spend time chatting about those topics, the projects as a whole, and things that we learned that we can apply to future work.

The meeting is not a vent session on the project. Work hard to make sure this meetings is the venue for healthy, productive discussion. The meeting is also not a walk-through of the retrospective document. Set a specific agenda for the meeting and focus in on a few key topics. If someone became the leader in your group, they can lead the meeting, but the goal of the meeting is to hear from the team. Coming out of the meeting, each team member is responsible for writing a memo of team and individual lessons learned with their team. In your memo, record the time, place, and attendance of the meeting.

Phase 4 Grading

Points			**Deliverables**
100	**Phase 4. Large Database**		
	50	**Large Data Processing Task**	
		15	Predetermined Query 1, Given 2 actors…
		5	Predetermined Query 1, Bonus, exclude an actor from the list above.
		15	Predetermined Query 2, Given 2 years…
		10	Predetermined Query 3, Team choice…
		5	Predetermined Query 4, Bonus, Given a genre…
		10	Coding Style
	30	**Group Retrospective Document**	
		5	Survey document.
		5	Retrospective Document – content and insight into project.
		10	Retrospective Document –analysis and lessons learned from the project.
		5	Retrospective Document – formatting and writing style.
		5	Agenda document.
	20	**Database Retrospective Meeting Memo**	
		10	Review of your product, teamwork process, and the project.
		10	Review of your roll in the project and the roll of your teammates.
		5	Agenda usage and meeting minutes.

Appendix A. Database Language Grammar

We will base our DBMS on relational algebra, and only implement the parser and the database (DB) engine that responds to queries.

- Relational algebra is a formal system for manipulating relations. It consists of only six primitive operations.
- Each of the operations takes relations as arguments, and produces a relation as a result. The operations thus compose freely.
- The upside of using relational algebra is that the implementation effort of the DBMS stays manageable.
- The downside is that queries tend to be more verbose and maybe a bit harder to construct than, say, with "real" SQL.

Relational Algebra

The six operations that are core to relational algebra are:

- **Selection:** select the tuples in a relation that satisfy a particular condition.
- **Projection:** select a subset of the attributes in a relation.
- **Renaming:** rename the attributes in a relation.
- **Set union:** compute the union of two relations; the relations must be union-compatible.
- **Set difference:** compute the set difference of two relations; the relations must be union-compatible.
- Cross product: compute the Cartesian product of two relations.

An extended operation in relational algebra is:

- **Natural join:** compute the combination of all tuples in two relations, say, R & S, which are equal on their common attribute names.
    - The common attributes only appear once in the result.
    - Implement "&" separately, not as a computationally expensive Cartesian product filtered by conditions.

Queries

The communication with the DBMS takes place using a domain-specific language. The grammar of queries in this language is as follows. (The | symbol means "or" and {} mean zero or more repetitions. Thus an identifier is a letter or underscore followed by zero or more letters, underscores, or digits.)

- `query ::= relation-name <- expr ;`
- `relation-name ::= identifier`
- `identifier ::= alpha { ( alpha | digit ) }`
- `alpha ::= a | … | z | A | … | Z | _`
- `digit ::= 0 | … | 9`
- `expr ::= atomic-expr | selection | projection | renaming | union | difference | product | natural-join`
- `atomic-expr ::= relation-name | ( expr )`
- `selection ::= select ( condition ) atomic-expr`
- `condition ::= conjunction { || conjunction }`
- `conjunction ::= comparison { && comparison }`
- `comparison ::= operand op operand | ( condition )`
- `op ::= == | != | < | > | <= | >=`
- `operand ::= attribute-name | literal`
- `attribute-name ::= identifier`
- `literal ::= intentionally left unspecified`
- `projection ::= project ( attribute-list ) atomic-expr`
- `attribute-list ::= attribute-name { , attribute-name }`
- `renaming ::= rename ( attribute-list ) atomic-expr`
- `union ::= atomic-expr + atomic-expr`
- `difference ::= atomic-expr - atomic-expr`
- `product ::= atomic-expr * atomic-expr`
- `natural-join ::= atomic-expr & atomic-expr`

Commands

Queries generated from the above grammar compute new relations based on existing relations. Queries can also name those new relations. We need, however, some ways to create some initial relations (constituting a database), update the relations within the database, store the results of queries back to the database, and delete tuples from relations. We use the following commands for these purposes:

- `command ::= ( open-cmd | close-cmd | write-cmd | exit-cmd | show-cmd | create-cmd | update-cmd | insert-cmd | delete-cmd ) ;`
- `open-cmd ::= OPEN relation-name`
- `close-cmd ::= CLOSE relation-name`
- `write-cmd ::= WRITE relation-name`
- `exit-cmd ::= EXIT`
- `show-cmd ::= SHOW atomic-expr`
- `create-cmd ::= CREATE TABLE relation-name ( typed-attribute-list ) PRIMARY KEY ( attribute-list )`
- `update-cmd ::= UPDATE relation-name SET attribute-name = literal { , attribute-name = literal } WHERE condition`
- `insert-cmd ::= INSERT INTO relation-name VALUES FROM ( literal { , literal } )| INSERT INTO relation-name VALUES FROM RELATION expr`
- `delete-cmd ::= DELETE FROM relation-name WHERE condition`
- `typed-attribute-list ::= attribute-name type { , attribute-name type }`
- `type ::= VARCHAR ( integer ) | INTEGER`
- `integer ::= digit { digit }`

Starter

The start rule is necessary to run the entire grammar.

- `program ::= { ( query | command ) }`

Grammar Example

```
CREATE TABLE animals (name VARCHAR(20), kind VARCHAR(8), years INTEGER) PRIMARY
KEY (name, kind);

INSERT INTO animals VALUES FROM ("Joe", "cat", 4);
INSERT INTO animals VALUES FROM ("Spot", "dog", 10);
INSERT INTO animals VALUES FROM ("Snoopy", "dog", 3);
INSERT INTO animals VALUES FROM ("Tweety", "bird", 1);
INSERT INTO animals VALUES FROM ("Joe", "bird", 2);

SHOW animals;

dogs <- select (kind == "dog") animals;
old_dogs <- select (age > 10) dogs;

cats_or_dogs <- dogs + (select (kind == "cat") animals);

CREATE TABLE species (kind VARCHAR(10)) PRIMARY KEY (kind);

INSERT INTO species VALUES FROM RELATION project (kind) animals;
```

```
a <- rename (aname, akind) (project (name, kind) animals);
common_names <- project (name) (select (aname == name && akind != kind) (a *
animals));
answer <- common_names;

SHOW answer;

WRITE animals;
CLOSE animals;

EXIT;
```

Additional Information

- Note that we made a distinction between queries and commands in the grammar of the DML. The result of a query is a view. A view is not stored in the database. Rather, it is a temporary relation whose lifetime ends when a DML program finishes. So only the updates caused by the commands persist from one DML program execution to another.
- The relations themselves should be saved in a file in plain ASCII text, using the same DML described above (e.g., CREATE ... INSERT ... INSERT ....). To make it simple, let us assume that each database file can only store one relation and the filename is the same as the relation name with the suffix ".db".
- To load a relation from a database file, use the OPEN command. Opening a nonexistent file will result in nothing.
- To add a new relation to a file, use the WRITE command (the filename will be by default "relationname.db").
- To save all changes to the relation in a database file and close, use the CLOSE command.
- To exit from the DML interpreter, use the EXIT command.
- To print a certain relation or a view, use the SHOW command.