**Design Document and Final Report**

1. Title  
   A short program **title**, the **author's name**, **author's ID**, and **date**.

Title: Movie Database

Author’s Name: Jake Billings

Author’s ID: 105955110

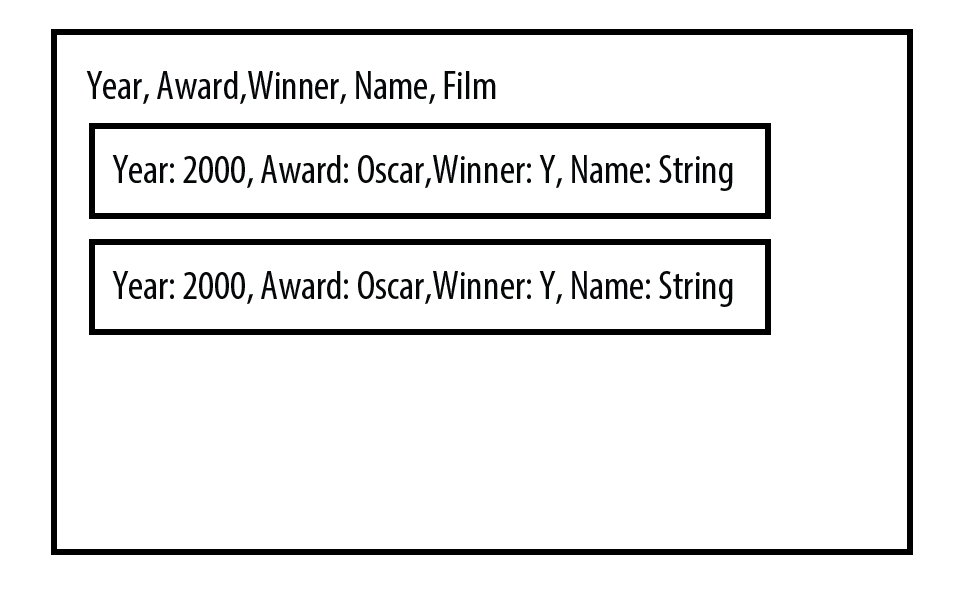
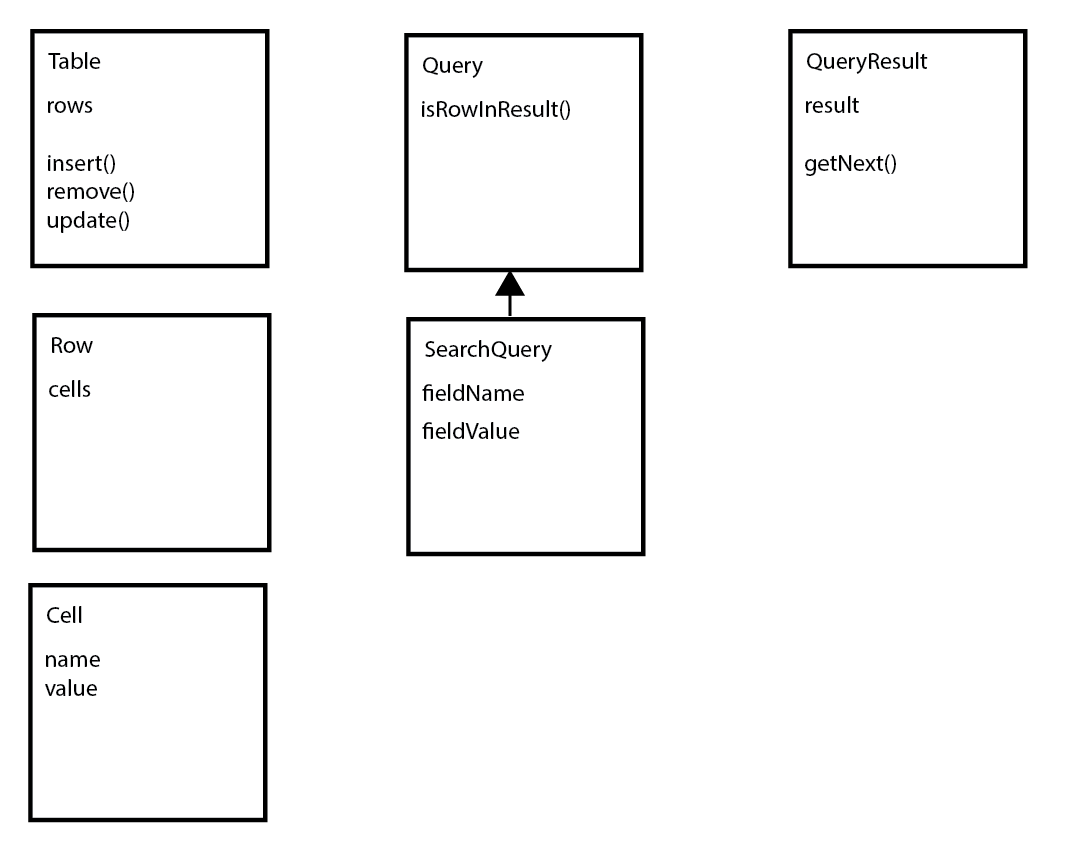
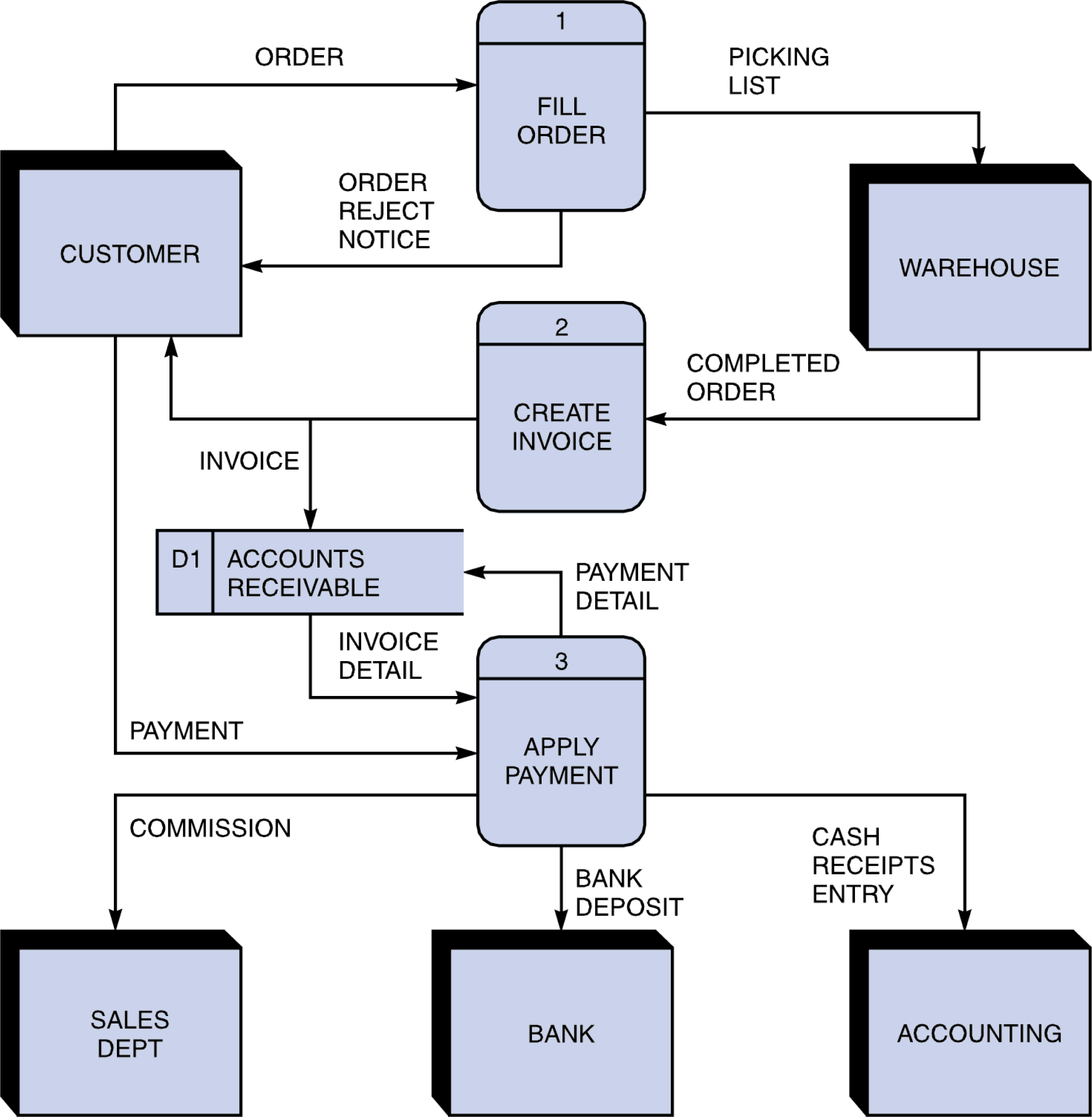
Date: May 2nd, 2018

1. Problem DescriptionA brief description of what this program does.

This program loads a amount of csv data about movies, actors, and actresses. Then, it provides a utility to sort and search the data. The database will provide the ability to add, search, modify, sort, and delete records.

1. Overall Software ArchitectureA brief description of major functions and their main roles in the program. You need to explain how the entire program is constructed and how the functions are related each other. You don't have to explain every little function. A diagram to display relation is very useful to get an overall picture. Here's an example of the diagram for a different program but you'll get an idea.

This program will abstract the data tables it is loading to hash maps. Each dataset will be represented as a “table.” Each table will consist of a set of “rows.” Rows are records. Rows are collections of “Cells.” Each cell is a key, value pair. Each row functions as a map. A “Query” is an abstract class that defines a subset of a table using a function. Queries will support searching and matching on any property. A table will accept a query and return a “QueryResult.” A query result will be a collection of “affected” rows. Affected rows are the results of queries. The program will load each row of the CSV file as a row and then insert it into a table. The table will keep “indexes” in the form of binary search trees for every field. This will be memory-inefficient but will result in quick query times. A menu system will generate and execute queries.

  
             
****Figure 1

* + Usually your start and end points (or program exit points) are denoted with a circle
  + D1 is an external data file (arrows point to whether it is input or output)
  + Arrows are labeled with the (general) input and output types…list major data structures or results like input of “Unsorted List and output labeled as “List Sorted by ID”
  + Not listed on the diagram above, but necessary for this project, you should list the major classes, functions and member functions in each block (under the name of the block)

1. Input RequirementsA detailed **list of all external inputs** (from files or keyboard) including a description of the **data type** and **range of valid values** for each input. For input file format and interactive user input, you need to write what data type is used for every field and valid value and length.

File Input: two CSV files to load the initial data tables

Must be CSV formatted (commas between cells and newlines between rows)

Must contain all required data types for the project

(Year,Award,Winner,Name,Film, Etc…)

Keyboard Input: input into a menu system that generates queries

Numeric values are valid for menu options

Strings are valid for searches

Field names are valid for sorts

1. Output RequirementsA detailed **list or description of all outputs (**to files) including a description of the **data type** and **range of valid values** for each output.

Output:

Strings representing menu structure (easy UI stuff)

Tab-delimited rows of sorted/searched data

These will be strings from the Table strcuture

1. Problem Solution Discussion  
   A summary description of the solution steps with algorithms analysis (1 paragraph, approximately 100 words). If any unusual techniques or algorithms are used that need further explanation, and additional paragraph may be used.

A map structure will be used to represent each row. A binary search tree (BST) index will be created for each field in each table in order to optimize searches. Traversal of this tree from top to bottom will yield O(log(n)) searches. Sorting will be performed using bubble sort if there is no noticeable delay. Otherwise, a more efficient sort will be used. Inheritance will be used to make queries generic.

1. Data StructuresA description of choice of your data structures and justification. Of course the main data structure for the database is Binary Search tree. But for some internal operations you may have to use a few sub containers. So include a brief explanation for your choice. For example, "I have considered DS1, DS2, and DS3. Their pros and cons are summarized as follow... I choose DS1 over DS2 and DS3 because ...."

I will use a map to represent each row/record because it will allow for general algorithms that perform searching and sorting without ridiculous templating and typing. I have considered using C++ classes, and I believe this will result in a large amount of duplicate code. I will use a vector to represent query results due to its STL optimizations. A LinkedList could theoretically provide great performance if the order doesn’t matter. However, STL optimizations likely beat my LinkedList. Additionally, sorting algorithms must be performed, and this will not perform well on a LinkedList.

1. User Interface SchemeUser interface scheme should show the menu items at top level and items in sub menus and how to navigate through menus.

The use interface will be as generic as possible. The goal is the build a query object, send it to the database, display the result, and repeat.

The first menu will select query type. (E.g. search/sort/insert)

The second menu will collect all required fields for the query. (E.g. field name and value)

Then, it will execute and display a table of results.

1. Status of Application  
   Note what IDE you developed the Final Project, and whether it compiled and operated properly on the csegrid. Note any requirements that were not met (per the project description). Note any known bugs or issues. If you did extra credit, note the status and what kinds of reports you provided.

This application has not yet been written. This is only a design document.

**Final Report**

Requirements:

**R1:** You will read in actor-actress.csv which is formatted as a csv file (Ask the user for the file name). Then place the items in a Binary Search Tree, sorted by name.

This is a common format and is comma separated (instead of being on separate lines). So you will have comma's between the values. Blank values will just have a comma noting to go to the next field. (so you may have value,, indicating a blank field.) Each line ends in a newline, not a comma. If you want to view the file, often this will be opened by a spreadsheet unless you specifically open it with a text editor. Do not open it with Microsoft Word, as this may change the format. Consider using getline with three parameters, as an easy way to read in files The first line of a CSV file notes the data descriptions as follows:

Year,Award,Winner,Name,Film

The Winner field has a one if they won and a zero if they did not win.

**R2:** Then you will read in information about the movies (called pictures by the Academy of Motion Pictures) that have won best picture award. Place these items in a Binary Search Tree, sorted by name. This is also formatted as a .csv file

The first line of pictures.csv contains the data fields including:

name,year,nominations,rating,duration,genre1,genre2,release,metacritic,synopsis

**R3/R4:** Choose either the movie or actor database and add a record

**R5/R6:** Choose either the movie or actor database, search for a record, and modify the fields.

**R7/R8**: Choose either the movie or actor database, search for a record and delete the fields.

**R9/R10:** Choose either the movie or actor database andsort by any single (sortable) field

**R11/R12:** Choose either the movie or the actor database and do a complete search on any “complete” searchable field. It is unlikely that you would have an exact match on an entire description, so that would not be listed to search.

**R13/R14:** Choose either the movie or the actor database and do a partial search on any searchable field. A partial search is any substring within a field.

**R15/R16:** Choose either the movie or actor database, ask for a file name, and print out a .csv file of the latest database (after adds, deletes or modifies). Remember that the first line of a .csv file lists the name of the fields separated with commas, ending in a newline. Then the following lines are the information from the fields separated with commas, ending in a newline.

Requirement implementation by number:

1. Load works and is verified by testing
2. Load works and is verified by testing
3. Add functionality works for all tables in database
4. Add functionality works for all tables in database
5. Modify functionality works for all tables in database
6. Modify functionality works for all tables in database
7. Exact search functionality works for all tables in database
8. Exact search functionality works for all tables in database
9. Sort functionality works for all tables in database
10. Sort functionality works for all tables in database
11. Exact search functionality works for all tables in database
12. Exact search functionality works for all tables in database
13. Partial search functionality works for all tables in database
14. Partial search functionality works for all tables in database
15. Any table can be printed as a CSV
16. Any table can be printed as a CSV

User Interface

The user interface is intended to give wizard-like easy to data manipulation. Each manipulation function (Load, Delete, Etc..) is selected at the root menu. Then, clarifications are made until the operation can execute. For instance, calling add results in a question asking which table then asking for each field.

Since the database uses dynamic maps, you must load from CSV to get the titles before adding, viewing, or modifying data.

The entire program is complete and functional. The extra credit is approximately 50% complete. I only load nominations as a title. I don’t calculate statistics.

Design

Task: Implement a database using any data structures you want; although, using a binary search tree is required. It was also required to use some provided files for the binary search tree.

I implement the database as an unordered map of unordered maps of binary search trees of unordered maps

Layer 1: Table: the data shape/type Key: Table Name (E.g. Actors, Nominations)  
 Layer 2: Index: the pre-sort column of the binary tree Key: Column Name (E.g. Name)  
 Layer 3: BST: the sorted BST for a given column name Key: Cell Value Name (E.g. John Smith)  
 Layer 4: object/row inspired by javascript objects Key: Column Name (E.g. Name)

Any exact, single-element lookup should return in Avg: O(1) \* O(1) \* O(log(n)) \* O(1) = O(log(n)). This could theoretically be more efficient if the BST was replaced with another unordered map; however, we would lose sort indexing, and the project requirement of having a BST. I perform Unit, Integration, and E2E styles of testing in test() in test.cpp. This verifies the functionality of the database software and of the UI. I only included nominations.csv into the project. All nominations can be viewed, edited, etc. just like the other files. I did not implement statistics.