CS 120: Intro to Computer Programming II

# In-Class Activity - 08 Top-Down Design - Day 2

## Background: Databases

A **database** is a storage system which allows you to store a very large amount of data, organize it, and access is efficiently and safely. They have all sorts of cool features - which you can study in other classes - which make them ideal for the backing storage for gigantic systems with many users, or many different programs (such as a website).

For this Activity, what you need to know is that databases organize their data into **tables.** A table is a little bit like a spreadsheet - it's organized into rows and columns; each row represents on piece of data (a "record") and each column represents a field. In a given table, all records must have all the same fields (although sometimes we will leave a field blank). While a database can insert new records into a table ludicrously quickly (and search them very quickly, too!), it's pretty painful to change the set of fields in a table. This means that database programmers must carefully plan out their table designs, ahead of time.

For almost any realistic system, you will use more than one table. Just like you might define different classes in a program to handle different types of objects, a realistic database system has multiple tables to hold different types of records. (Each table can have its own set of fields; they don't all have to be the same.) Also, realistic databases have all sorts of cool tricks for connecting these tables together - data in one table can have a reference (called a "foreign key") which connects it to data in another table.

For our Activity today, you will only have a single table, for simplicity. But you're going to have to define what it does, and how you will access it.

## Overview

Today, we're going to do another design decomposition activity. This time, you're going to be writing a set of programs will access the same database - a database of whale sightings, to be used to help keep track of the populations as they migrate, grow, and change.

This Activity will involve three different programs. The first program will be used for data entry - new record(s) will be added to the table. You will have to think about what fields might be important to store, and also if you want to do any error checking about what is being added.

The second program will be a visualizer. It will present a GUI, which gives a map of the world, and users will connect to it; they will give various search parameters. Each time that the user changes the search parameters, the tool will search the database for sightings which match the search, and then will draw information on the screen.

The third program will be an automatic notification tool. This program will run for a long time. It will check the database from time to time, looking for new information, and sending an email (or perhaps a notification on a cell phone) any time that new records, matching some sort of search pattern, are found.

**As with the previous Activity, don't worry too much about how far your group gets. Just do as much as you have time for, and turn that in!**

(activity continues on the next page)

## First Things First: Choose Your Fields

Today, since our programs will all be centered around a database, the first step is to define what you will store. For simplicity, let's assume that all of your data will be in a single table. (If you want to violate this rule, and hanve multiple tables, I'm OK with that. That will be more realistic, but it will also be more work!)

Decide, as a group, what fields your table will store. **REMEMBER:** Your database cannot store **anything** that you don't encode as a field in the table. But the more fields you add, the more complex you programs will have to be.

Make sure that you turn in the definition of your fields to your TA.

## Program 1 - Input

Your first program will be handling data entry. It will add one or more records to the table.

Your first decision is, how many records should it add? Should it add only a single record - meaning that the user runs it many times to add many different records - or should it add many of them, in a loop? What sort of interface will you present to the user - text, graphical, website, etc. ?

Second, you should do some error checking. What are some common mistakes that a user might make, which you want to prevent? For instance, do you sanity-check that the species of whale is something you recognize? (If so, how do you know what the valid species are, and how do you update that list? Also, what if the user misspells the name, or just uses the wrong capitalizations? What about shortnames and nicknames - will you turn a common name into a more formal one, or just leave it as-is?)

Will you double-check that the location given, in the sighting report, is actually in the ocean?

Will you require some sort of authentication to add a record? If so, how will you store the list of allowed users?

Turn in, to your TA, a discussion of your design decisions, and a rough overview of how the program will run. (If you find that you have lots of extra time, you can make this overview more detailed.)

**NOTE:** Of course, I don't expect you to know all the details of how a database is actually accessed. You will almost certainly want to leave the "database access" part as a big TODO of your program.

## Program 2 - Visualizer

Your second program will present a visual interpretation of the data. You will create a GUI program (or a website) which displays a map of the world. There will be some fields and option boxes, which will allow the user to specify search parameters. Make the search options (a little bit) flexible, so they can search for different parameters. (For example, you might allow your users to search by species, date ranges, or the username who posted the report.)

Draw a picture of what the main window of the program will look like. Will it be completely filled with the map, or will there be buttons on the sides? What will happen when the user moves the mouse? Can they click on things? Can they scroll or zoom?

Once you have a (very) rough idea of what the user interface will look like, sketch out how the code would work. What do you need to chcek for? How would you get the picture of the world on the screen - and are you handling zooming and panning? How would you search the database? If you allow zooming, do you try to limit your searches to just those that would fit into the current window, or do you search for records across the whole globe?

Turn in to your TA, the picture(s) you drew of the interface, and any information you have about how the program will run.

(activity continues on the next page)

## Program 3 - Notifier

In a big database, it's often useful to be notified when something changes that you might be interested in.

This program will be one that runs for a long time; perhaps some user will run it in the background.[1] From time to time, this program will query, the database, looking for records that match a certain search pattern, defined by the user. When it finds a new record or records, it will automatically send an email (or maybe an app notification) to the user, telling them that there is something interesting in the database.

There are several key problems you need to solve. First, you don't want to send emails about old records - and you certainly don't want to send many emails about the same record. How will this program make sure that it doesn't send duplicate information to its user? And just as important, **what will it do when it first starts** - will it send emails about all of the old, existing records? Surely the user won't want to hear about the history, right? How would the program tell what to notify about, and what not?

Second, you are probably using a "polling" design - which means that you periodically query the database, looking for new records. Presumably, you don't want to do this thousands of times a second - discuss with your group why this might be a bad idea! So how often will you check? What are the tradeoffs here - what happens if you wait a long time between checks?

Moreover, databases are **really** good at searching through their data - and when you ask the database for data, you always give it search terms, so that it returns some of the records in the table, but not all of them. Could you come up with some search terms so that, each time you poll the database looking for more information, it will **automatically** skip the old records for you, so that you don't even have to look at them?

Turn in to your TA, a discussion of the design decisions your group made - along with any overview of the code design as well (if you got that far).

[1] More realistically, this is probably something that would run once in a while, or be integrated into the database system itself - but again, that would be more complexity than you currently know how to handle.