CSC173: Project 4
The Relational Data Model

In this project you will gain experience with databases and relational algebra by implementing your own database system. We will build on the presentation (and code) in the textbook.

Note: Your writeup for this project will be more extensive than previous projects. In addition to the usual instructions for building and running your project, the requirements below ask you to describe your work in your writeup. You should submit a nicely-formatted PDF document for this (no plain text or Word files).

Part 1

1. Implement a database containing the relations (tables) shown in FOCS Figure 8.1 and 8.2 (also seen in class). Use the method described in Section 8.2 in the section "Representing Relations" and elaborated in Section 8.4 "Primary Storage Structures for Relations." Describe your implementation briefly but clearly in your writeup.

2. Implement the basic single-relation *insert*, *delete*, and *lookup* operations as functions. If you use the implementation described in the textbook, you will need separate functions for each relation. You should support leaving some attributes unspecified for *delete* and *lookup* (denoted with "*" in the textbook, perhaps something else in your code). Describe your implementation briefly but clearly in your writeup.

3. Use your *insert* method to populate the tables with (at least) the data given in the figures. Then demonstrate all three operations by performing the operations shown in Example 8.2 (p. 409). Be sure that your program explains itself when run (using informative printed messages).

4. Implement functions for saving your database to one or more files, and loading from the same. Demonstrate this functionality also. If the user needs to perform multiple steps, make sure this is clear in your writeup. Note: you can do all the other parts of the project even without this functionality, so don't let it stop you or slow you down.

Part 2

For this part, use the database with all the tuples from Figures 8.1 and 8.2 (no deletions).

1. Write a function to answer the query "What grade did *StudentName* get in *CourseName*?" as described in Section 8.6 "Navigation Among Relations." Demonstrate this functionality by performing the query and printing the results informatively.
2. Write a function to answer the query "Where is *StudentName* at *Time* on *Day*?" (assuming they are in some course). Demonstrate this functionality also.

Part 3

1. Implement the Relational Algebra operations as described in Section 8.8 as necessary to do the operations on the "registrar" database described in Examples 8.12 (Selection), 8.13 (Projection), 8.14 (Join), and 8.15 (all three). Describe your implementation briefly but clearly in your writeup.
    - If you follow the implementation in the textbook, you will probably need a different function for each different set of arguments to the operator. You need only implement the ones that you need for the examples listed above. You may also throw in any additional examples you feel illustrate relevant aspects of your implementation.
    - Note that some of the Relational Algebra operations create a relation with a new schema from that of their operands. (You should know which these are...) But if tuples are implemented using structs, how do you create instances of these new "on-the-fly" relations? The answer is that you should solve the problem yourself by hand so that you know the schemas needed by your examples. Then add appropriate structure definitions to your program to allow you to do the examples. This is one of many differences between what you need to do for this project and a real database framework.

Extra Credit (20% max total)

1. The code you wrote for the "registrar" database is obviously specific to it. A true database system, like SQLite or MySQL, allows you to represent any database schema. Generalize your code to represent arbitrary databases consisting of arbitrary relations. Note that you do not need to understand SQL for this. What you

need to do is create “generic” representations of tuples and tables and then use them to write code for some specific example. You can use the “registrar” example or the DMV database (Exercise 8.3.2) or some other application. Explain what you’ve done and how it will be demonstrated by the program(s) in your writeup.

2. Identify a subset of SQL that you can support with your implementation. Write a simple parser for that language, and use it to create and query one or more example databases. Explain what you’ve done and how it will be demonstrated by the program(s) in your writeup.

Project Submission

Your project submission MUST include the following:

1. A README.txt file or PDF document describing:
   (a) How to build your project
   (b) How to run your project’s program(s) to demonstrate that it/they meet the requirements
   (c) Any collaborators (see below)
   (d) Acknowledgements for anything you did not code yourself (you should avoid this, other than the code we’ve given you, which you don’t have to acknowledge)
2. All source code for your project, including a `Makefile` or shell script that will build the program per your instructions. Do not just submit Eclipse projects without this additional information.

We must be able to cut-and-paste from your documentation in order to build and run your code. **The easier you make this for us, the better grade your will be.**

Programming Policies

You must write your programs using the “C99” dialect of C. This means using the “`-std=c99`” option with `gcc` or `clang`. For more information, check Wikipedia.

You must also use the options "`-Wall -Werror`". These cause the compiler to report all warnings, and to make any warnings into errors that prevent your program from compiling. You should be able to write code without warnings in this course.

With these settings, your program should compile and run consistently on any platform.

Furthermore, your program must pass `valgrind` with no error messages. Of course this might differ for different executions, but it should be clean for any fixed examples requested in the project description. Any questions, ask the TAs *early*.

Projects in this course will be evaluated using `gcc` on Fedora Linux (recent version, like 25 or 26). Here are several ways you can setup this environment for yourself:

- Visit `getfedora.org` and download and install Fedora on a spare computer or separate partition of your main computer.
- Don't have a spare computer? No problem. Visit `virtualbox.org` and download VirtualBox. Then setup a new virtual machine running Fedora Linux (you may need to download the installer as above). VMWare is a commercial virtualization app if you want to buy something.
- Not confident setting up a VM or don't have space? No problem. Visit `docker.com` and download and install Docker for your computer. You can easily setup an image with Fedora Linux and `gcc`, or see the appendix below for more.
- Don't want to install anything? No problem. You need an account on the Computer Science Department's undergraduate cycle servers. Ask the TAs about it. Do not wait until the last minute. This will not happen overnight.

Mac and Windows users especially beware: You must test your code under Fedora Linux, or expect problems (and low grades).

Late Policy

Don't be late. But if you are: 5% penalty for the first hour or part thereof, 10% penalty per hour or part thereof after the first.

Collaboration Policy

You will learn the most if you do the projects YOURSELF.

That said, collaboration on projects is permitted, subject to the following requirements:

- Groups of no more than 3 students, all currently taking CSC173.
- You must be able to explain anything you or your group submit, IN PERSON AT ANY TIME, at the instructor's or TA's discretion.
- One member of the group should submit on the group's behalf and the grade will be shared with other members of the group. Other group members should submit a short comment naming the other collaborators.
- All members of a collaborative group will get the same grade on the project.

Appendix: Using Docker

"Docker provides operating-system-level virtualization [...] to allow independent "containers" to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines." (Wikipedia)

Getting Started with Docker

I have built a docker image based on Fedora linux and containing `gcc`, `make`, `gdb`, `valgrind`, and `emacs`, as well as manpages. You can download it from here:

http://www.cs.rochester.edu/u/ferguson/csc/docker/

Once you have installed Docker, load the image to create the `fedora-gcc` image in your Docker installation:

```
docker load -i fedora-gcc-docker.tgz
```

Then you can do, for example:

```
docker run -it --rm -v "`pwd`":/home -w /home fedora-gcc bash
```

This runs `bash` (the command shell) using the `fedora-gcc` image. The `-it` option means to run interactively; `--rm` tells Docker to remove the container when the process exits; `-v` and the bit with the colon says to make your current working directory (outside docker) available on `/home` within docker; and `-w` tells docker to make `/home` the initial working directory for the `bash` process. Whew!

You can do a lot with Docker. You'll have to read the docs but it's a great option. Please report any problems with the image ASAP.

Frequently Asked Questions

Q: In C, how can I make a hastable that can hold any kind of tuple?

A: How would you do it in Java? Think about it. . .

Answer: Generics. And indeed `java.util.Hashtable` is a generic class.

Now, does C have generics? Answer: no. So this is an example of the kind of thing added to Java (and most object-oriented languages) as it evolved from C, in order to capture common programming patterns more effectively. But if all we have is C, what can we do?

For some dynamic data structures, you can just manage "generic" objects of type `void*` and use casts as needed (or, preferably, helper functions).

The problem is that unlike a linked list, a hashtable needs to know something about the objects it is managing. Why? Because it has to hash them, which means accessing some of their components. So you would have to be able to tell the "generic" hashtable code which hash function to use for each type of struct it will be managing. There are ways (*e.g.*, function pointer in tuple or hashtable), and they all amount to reinventing (in fact, pre-inventing) the idea of true classes.

You could just cut and paste the code for different flavors of hashtable. So the routines use the right type of struct for arguments and they know what hash function to use. Is this ugly? Yes. Is it bad software engineering practice? Yes. Is it ok for this project? Yes. This is similar to the business about the schemas of tuples produced by JOIN operations, as described in the project description Part 3.

If you were ambitious you might observe that the cut-and-paste method is purely mechanical, other than writing the hash function itself. That is, you basically go through and change all occurrences of the element type name and that's really all you need to do. Given that, you could define a macro or macros that expanded into the code. Something like:

```
#define RELATION(ETYPE,HFUNC) ...
```

for element type `ETYPE` and name of hash function `HFUNC`. Using macros is a bit of a power C programmer technique. But it is how we did "generic" code back in the day, and as you may know, C++ grew out of just this kind of macro-powered code.

You could also implement tuples using something other than structs, as described in the first Extra Credit problem. There are many possibilities. . .