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. You **must** follow the formal model, even though your database will be very small. If you do it properly, it would also work for a large database.

Part 1 (40%)

1. Implement a database containing the relations (tables) shown in Appendix A. This database uses the same relations as FOCS Figs. 8.1 and 8.2 (also seen in class) but with different data (tuples). Use the method described in Section 8.2 in the section "Representing Relations" and elaborated in Section 8.4 "Primary Storage Structures for Relations." Use hashtables to store the sets of tuples, as seen in class and in the textbook.

2. Implement the basic single-relation *insert*, *delete*, and *lookup* operations as functions. Using the implementation described in the textbook, you will need separate functions for each relation (*e.g.*, `insert_CSG`). You should support leaving some attributes unspecified for *delete* and *lookup* (denoted with "*" in the textbook, perhaps something else in your code).

3. Use your *insert* method to populate the tables with the data in Appendix A. Print the contents of the database clearly after loading the data.

4. Finally, demonstrate all three operations by performing the following operations, which are similar to those shown in FOCS Example 8.2 (p. 409). Print what is being done and print the affected table after the operation.

   (a) $lookup(\langle \text{“CSC172”}, 98789, * \rangle, \text{Course-StudentId-Grade})$

   (b) $lookup(\langle \text{“CSC173”}, \text{“CSC172”} \rangle, \text{Course-Prerequisite})$

   (c) $delete(\langle \text{“DSCC201”}, *, * \rangle, \text{Course-Day-Hour})$

   (d) $insert(\langle \text{“CSC280”}, \text{“CSC173”} \rangle, \text{Course-Prerequisite})$

   (e) $insert(\langle \text{“DSCC202”}, \text{“DSCC201”} \rangle, \text{Course-Prerequisite})$

Part 2 (30%)

For this part, use the database with all the tuples from Appendix A (that is, before any deletions or insertions). I suggest that you have a function that prepares the database (creates the relations and inserts the tuples). Then use that function twice, once for Part 1 and once for Part 2.

1. Write a function to answer the query "What grade did *Name* get in *Course*?" as described in FOCS Section 8.6 "Navigation Among Relations." The code for your function **must** look like the pseudocode in FOCS Fig. 8.8 or 8.9. (I recommend using the pseudocode as comments in your code.)

   Demonstrate this functionality with a "Read-Eval-Print Loop" or "REPL": ask the user for the query parameters (*Name* and *Course*, not an entire English sentence), perform the query, print the results informatively, until the user says to stop.

2. Write a function to answer the query "Where is *Name* at *Hour* on *Day*?" (assuming they are in some class, in which case their location is a room, as seen in the textbook and in our class). The code for your function **must** follow the model shown in FOCS Fig. 8.10 (described starting on p. 426), also seen in class.

   Demonstrate this functionality with a REPL also.

Make sure it is clear what query is being used, what values the program is asking for, and what are the results of the query. It is **your responsibility** to makie it clear for us.

Part 3 (30%)

Implement the Relational Algebra operations as described in FOCS Section 8.8. Demonstrate this by evaluating the following Relational Algebra expressions, which are similar to those in FOCS Examples 8.12–8.15. Print an informative description of the expression and print the table that is the result of evaluating it.

1. Selection: $\sigma_{\text{StudentId=67890}}(CSG)$
2. Projection: $\pi_{\text{Course}}(\sigma_{\text{StudentId=67890}}(CSG))$
3. Join: $CR \bowtie CDH$
4. All of the above: $\pi_{\text{Day,Hour}}(\sigma_{\text{Room="Wegmans 1400"}}(CR \bowtie CDH))$

If you follow the implementation in the textbook, you will probably need a different function for each different set of arguments to the operator (*e.g.*, `select_CSG_Course`, `join_CSG_SNAP`, ...). Only implement the ones that you need for the examples listed above. You may also throw in any additional examples that you feel illustrate relevant aspects of your implementation.

Note that some of the Relational Algebra operations create a relation with a different schema from that of their operands. (You should know which operations do this...) 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 code up the required examples using your implementations of the Relational Algebra operators. This is one of many differences between what you need to do for this project and a real database framework.

Extra Credit

1. Implement functions for saving your database to and loading it from **a single file**. Demonstrate this functionality in your program(s) and explain 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. [max 10% extra]

2. 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 the specific examples used in this project. Demonstrate your "generic" database performing the updates and queries from Parts 1 and 2 (including REPLs where neceessary).

   If you choose to do this, and are confident in your implementation, you may use it for the required parts of the project. Explain what you've done in your writeup and we will give you the extra points if it all works. You may also choose to demonstrate this separately, for example with another program. Again, be clear in your writeup so that we can give you the points. [max 10% extra]

Additional Requirements and Policies

The short version:

- You **must** use C compiler options "`-std=c99 -Wall -Werror`".
- If you are using an IDE, you **must** configure it to use those options (but I suggest that you take this opportunity to learn how to use the command-line).
- You **must** submit a ZIP including your source code and a README by the deadline.
- You **must** tell us how to build your project in your README.
- You **must** tell us how to run your project in your README.
- Projects that do not compile will receive a grade of **0**.
- Projects that do not run or that crash will receive a grade of **0** for whatever parts did not work.
- Late projects will receive a grade of **0** (see below regarding extenuating circumstances).
- You will learn the most if you do the project yourself, but collaboration is permitted in teams of up to 3 students.
- Do not copy code from other students or from the Internet.

Detailed information follows. . .

Programming Requirements

C programs **must** be written using the "C99" dialect of C. This means using the "`-std=c99`" option with `gcc` or `clang`. For more information, see 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 **must** be able to write code without warnings in this course.

With these settings, your program should compile and run consistently on any platform. We will deal with any platform-specific discrepancies as they arise.

If you are using an IDE (Eclipse, XCode, VSCode, CLion, *etc.*), you **must** ensure that it will also build as described above. The easiest way to do that is to setup the IDE with the required compiler options. There are some notes about this in the C Programming Resources (for CSC173 and beyond) area.

Furthermore, your program should pass `valgrind` with no error messages. If you don't know what this means or why it is A Good Thing, look at the C for Java Programmers document which has a short section about it. Programs that do not receive a clean report from `valgrind` have problems that **should be fixed** whether or not they appear to run properly. If your program does not work for us, the first thing we're going to do is run valgrind on it.

Submission Requirements

You **must** submit your project as a ZIP archive of a folder (directory) containing the following items:

1. A file named `README.txt` or `README.pdf` (see below)
2. The source code for your project (do not include object files or executables in your submission)
3. A completed copy of the submission form posted with the project description (details below).

The name of the folder in ZIP **must** include "CSC173", "Project 1" (or whatever), and the NetID(s) of the submitters. For example: "`CSC173_Project_1_aturing`"

Your README **must** include the following information:

1. The course: "CSC173"
2. The assignment or project (*e.g.*, "Project 1")
3. Your name and email address
4. The names and email addresses of any collaborators (per the course policy on collaboration)
5. Instructions for building your project (with the required compiler options)

6. Instructions for running your project

The purpose of the submission form is so that we know which parts of the project you attempted and where we can find the code for some of the key required features.

- **Projects without a submission form or whose submission form does not accurately describe the project will receive a grade of 0**.
- If you cannot complete and save a PDF form, submit a text file containing the questions and your (brief) answers.

Project Evaluation

You **must** tell us in your README how to build your project and how to run it.

Note that we will NOT load projects into Eclipse or any other IDE. We **must** be able to build and run your programs from the command-line. If you have questions about that, go to a study session.

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 your grade will be.** It is **your** job to make the building of your project easy and the running of its program(s) easy and informative.

For C projects, the most common command for building a program from all the C source files in the directory (folder) is:

```
gcc -std=c99 -Wall -Werror -o EXECUTABLE *.c
```

where `EXECUTABLE` is the name of the executable program that we will run to execute your project.

You may also tell use to build your project using `make`. In that case, be sure to include your `Makefile` with your submission. You **must** ensure that your `Makefile` sets the compiler options appropriately.

If you expect us to do something else, you **must** describe what we need to do in your `README` file. This is unlikely to be the case for most of the projects in CSC173.

Please note that we will **NOT** under any circumstances edit your source files. That is your job.

Projects that do not compile will receive a grade of 0. There is no way to know if your program is correct solely by looking at its source code (although we can sometimes tell that is incorrect). This is actually an aspect of a very deep result in Computer Science that we cover in CSC173.

We will then run your program by running the executable, or as described in the project description. If something else is required, you **must** describe what is needed in your `README` file.

Projects that do not run or that crash will receive a grade of 0 for whatever parts did not work. You earn credit for your project by meeting the project requirements. Projects that don't run don't meet the requirements.

Any questions about these requirements: go to study session **BEFORE** the project is due.

Late Policy

Late projects will receive a grade of 0. You MUST submit what you have by the deadline. If there are extenuating circumstances, submit what you have before the deadline and then explain yourself via email.

If you have a medical excuse (see the course syllabus), submit what you have and explain yourself as soon as you are able.

Collaboration Policy

I assume that you are in this course to learn. You will learn the most if you do the projects **YOURSELF**.

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

- Teams of no more than 3 students, all currently taking CSC173.
- You MUST be able to explain anything you or your team submit, IN PERSON AT ANY TIME, at the instructor's or TA's discretion.

- One member of the team should submit code on the team's behalf in addition to their writeup. Other team members MUST submit a README (only) indicating who their collaborators are.
- All members of a collaborative team will get the same grade on the project.

Working in a team only works if you actually do all parts of the project together. If you only do one part of, say, three, then you only learn one third of the material. If one member of a team doesn't do their part or does it incorrectly (or dishonestly), all members pay the price.

Academic Honesty

I assume that you are in this course to learn. You will learn nothing if you don't do the projects yourself.

Do not copy code from other students or from the Internet.

Avoid Github and StackOverflow completely for the duration of this course.

There is code out there for all these projects. You know it. We know it.

Posting homework and project solutions to public repositories on sites like GitHub is a violation of the University's Academic Honesty Policy, Section V.B.2 "Giving Unauthorized Aid." Honestly, no prospective employer wants to see your coursework. Make a great project outside of class and share that instead to show off your chops.

Frequently Asked Questions

Q: In C, how can I make a hashtable 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 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 (you **must** encapsulate the casts in helper functions or your code will be unreadable and unmaintainable).

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 need 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 stored with hashtable), and they all amount to reinventing (in fact, pre-inventing) the idea of true classes.

Or 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 are 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 DEFRELATION(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.

And BTW: If your tuples are not implemented as structs as seen in FOCS, you will still need to keep track of how to hash them for a particular relation (hashtable). Actually, for a particular **index** on a relation, implemented as a hashtable, right? There are many possibilities. . .

Appendix A: Project Relations (like FOCS Figs. 8.1 and 8.2)

StudentId	Name	Address	Phone
11111	M. Subban	1 Exchange Blvd	555-1212
12345	R. Zmolek	2700 B-H Townline Rd	555-1111
67890	P. Tischke	1 Exchange Blvd	555-1234
12321	F. Cedarqvist	80 Lyndon Rd	555-2222
98789	M. Subban	123 Ling Rd	555-3333

Note: There may be more than one student with the same name (of course).

Course	StudentId	Grade
CSC171	12345	A
CSC171	67890	B
MATH150	12345	A
DSCC201	12345	C
DSCC201	11111	B+
CSC172	98789	A-
MATH150	67890	C+
CSC173	12321	B+
CSC173	98789	A
DSCC201	98789	C

Course	Prerequisite
CSC172	CSC171
CSC172	MATH150
CSC173	CSC172
CSC252	CSC172
CSC254	CSC252
DSCC201	CSC171
DSCC202	DSCC201
DSCC265	CSC262
DSCC265	CSC171

Course	Day	Hour
CSC171	T	1400
CSC171	R	1400
CSC172	T	1525
CSC172	R	1640
CSC173	T	1400
CSC173	R	1400
CSC252	T	1230
CSC252	R	1230
DSCC201	M	900
DSCC201	W	900
DSCC202	M	1650
DSCC202	W	1650
DSCC265	M	1400
DSCC265	W	1400

Course	Room
CSC171	Hutchison Hall 141
CSC172	Hutchison Hall 141
CSC173	Wegmans 1400
CSC252	Wegmans 1400
CSC254	Wegmans 1400
DSCC201	Bausch & Lomb 109
DSCC202	Dewey 2162
DSCC265	Wegmans 1400
MATH150	Harkness 115