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 is similar to the "Registrar" database from FOCS Figs. 8.1 and 8.2 (also seen in class). You **MUST** use the method described in Section 8.2 in the section "Representing Relations" and elaborated in Section 8.4 "Primary Storage Structures for Relations." Your **MUST** 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_SNOP` for the $SNOP$ table). 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. You **MUST** print the contents of the database clearly after loading the data.

4. Then demonstrate your implementation by performing the following operations, which are similar to those shown in FOCS Example 8.2 (p. 409). You **MUST** print what operation is being performed, for *lookup*, you **MUST** print the result(s), and for *insert* and *delete*, you **MUST** print the affected table after the operation.

   (a) $lookup(\langle 99863, \text{“319-97-1535”}, * \rangle, \text{ProjectId-SSN-Hours})$

   (b) $lookup(\langle *, \text{“H. Kim”}, *, * \rangle, \text{SSN-Name-Office-Phone})$

   (c) $lookup(\langle 12345, 67890 \rangle, \text{ProjectId-DependsOn})$

   (d) $delete(\langle 63653, \text{“101-85-4521”} \rangle, \text{ProjectId-ManagerSSN})$

   (e) $delete(\langle 72278, \text{“Facilities”}, \text{“P123”} \rangle, \text{ProjectId-Department-Account})$

   (f) $delete(\langle 72278, *, * \rangle, \text{ProjectId-Department-Account})$

   (g) $insert(\langle 23169, 57849 \rangle, \text{ProjectId-DependsOn})$

   (h) $insert(\langle 57849, 31687 \rangle, \text{ProjectId-DependsOn})$

Part 2 (40%)

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 "How many hours does *Name* work on project *ProjectId*?" which is similar to the one 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 *ProjectId*), 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 "What is the manager's SSN of projects involving *Name* from *Department* using *Account*?" The code for your function **must** follow the model shown in FOCS Fig. 8.10 (described starting on p. 426), also seen in class. You may use indexes to improve performance on this query, but it is not required.

   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 (20%)

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 relation (table) that is the result of evaluating it.

1. Selection: $\sigma_{\text{ProjectId=99863}}(PSH)$
2. Projection: $\pi_{\text{Name}}(SNOP)$
3. Join: $PM \bowtie PDA$
4. All three operators: $\pi_{\text{Department,Account}}(\sigma_{\text{ManagerSSN="210-86-2988"}}(PM \bowtie PDA))$

If you follow the implementation in the textbook, you will probably need a different function for each different set of arguments to the operator. For example: `select_PSH_ProjectId`, with arguments the relation and value to select, `project_SNOP_Name`, with argument the relation, and `join_PM_PDA` with arguments the two relations. You only need to implement the ones that you need for the required expressions.

Your implementations of the Relational Algebra operators should return **new relations**. Then the fourth expression above can be implemented by "chaining together" the results of the required sub-expressions. If you do it correctly, you will be able to use the result of at least one of the previous required expressions.

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

There are no opportunities for extra credit in this project.

Your program should create the necessary database (tables and indexes) in code. You may find it interesting or useful to implement functions for saving your database to and loading it from a file. However it is not central to the topics covered in class, so you will **not** receive extra credit for it.

The code for the FOCS "Registrar" database is obviously specific to it, and similarly for your database in this project (assuming you followed FOCS 8.2 and 8.4). A true database system, like SQLite or MySQL, allows you to represent any database schema.

You may, if you wish, implement a more general representation that would allow you 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.

You may use your "generic" database for the project, but you will **not** receive any extra credit. I advise that you **NOT** take this approach. Stick with the textbook.

Additional Requirements and Policies

The short version:

- You **must** use the following 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 three (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 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.

You may **NOT** use `#pragma`'s in your programs. This includes `#pragma`'s added by an IDE or any other way that they might get into your code.

You **SHOULD** test your program with the memory checking program `valgrind`. If you don't know what `valgrind` is or why it is A Good Thing, read C for Java Programmers Chapter 11: Debugging a C Program.

Programs that do not receive a clean report from valgrind have problems that **should be fixed** whether or not the program sometimes runs properly on some platforms. The only exception is unfreed memory errors (so-called "memory leaks"). We will not penalize you for those in CSC173.

If your program does not work for us, the first thing we're going to do is run valgrind on it. If valgrind reports errors: you have errors in your program. Period.

It is easy to run valgrind in a virtual machine or Docker container if you cannot install and run it natively. Mac and Windows users can install Docker Desktop. See How to run Linux (including via Docker) for more information about Linux and Docker.

For help with `valgrind`, please go to study session **well before** the project deadline.

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.

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 do not run **do not** 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 do not 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.

The use of generative AI tools is **NOT** permitted in this course.

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.

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

SNOP			
SSN	Name	Office	Phone
972-06-2473	H. Kim	Dewey 402	851-3472
251-12-6815	E. Price	Hutchison 754	896-4408
021-08-4448	M. Ruiz	CSB 571	360-1897
319-97-1535	H. Kim	Morey 758	704-9520
210-86-2988	J. Jackson	Lattimore 879	952-9418
098-54-7473	R. Roberts	Harkness 156	314-9972
101-85-4521	M. Ruiz	Hylan 525	470-6491

Employees have a unique SSN. Names, addresses, and phone numbers are not necessarily unique.

PM	
ProjectId	ManagerSSN
57849	101-85-4521
23169	319-97-1535
99863	210-86-2988
63653	101-85-4521
31687	251-12-6815
72278	210-86-2988

Projects have unique identifiers (numbers). Each project has one manager, but one employee may manage multiple projects.

PDA		
ProjectId	Department	Account
63653	Marketing	A082
99863	Operations	N903
57849	Support	C764
99863	Facilities	J690
72278	Facilities	M857
31687	Facilities	R837
57849	Marketing	N903
23169	Operations	J690

Projects involve employees from one or more departments using one or more accounts.

PD	
ProjectId	DependsOn
99863	72278
57849	63653
57849	31687
31687	57849
63653	23169

A project may depend on one or more other projects.

PSH		
ProjectId	SSN	Hours
57849	251-12-6815	5.5
23169	319-97-1535	6.0
63653	021-08-4448	4.0
72278	210-86-2988	7.5
31687	251-12-6815	4.0
99863	319-97-1535	6.0
23169	098-54-7473	9.0
99863	101-85-4521	5.0
31687	101-85-4521	4.0
57849	101-85-4521	7.5
99863	098-54-7473	4.5

Employees work on projects for some number of hours. Employees may work on multiple projects, but for a given employee and project there is at most one number of hours.