Lab 1: Storage

Release Date: September 12, 2024

Checkpoint Due date: September 18, 2024

Due Date: September 25, 2024 by 11:59 pm ET

1 Introduction

In the lab assignments in 6.5830/6.5831 you will implement a basic database management system called GoDB incrementally. For this lab, you will focus on implementing the core modules required to access stored data on disk; in future labs, you will add support for various query processing operators, as well as transactions, locking, and concurrent queries. This handout goes through the set-up steps you will need to complete and the logistics of the lab. You will find detailed instructions on specific tasks you need to complete in `lab1.md` in the course GoDB repository.

2 Getting Started

We will be using git, a source code control tool, to distribute labs in 6.5830/6.5831. This will allow you to incrementally download the code for the labs, and for us to distribute any hot fixes that might be necessary.

You will also be able to use git to commit and backup your progress on the labs as you go. The course git repositories will be hosted on GitHub, a website that hosts git servers for thousands of open source projects. In our case, your code will be in a **private** repository that is visible only to you (and course staff if you elect to).

This section describes what you need to do to get started with git, download 6.5830/6.5831 labs via GitHub, and backup your progress on GitHub.

If you are not a registered student at MIT, you are welcome to follow along, but we ask you to please keep your solution PRIVATE and not make it publicly available

2.1 Learning Git

There are numerous guides on using Git. We encourage you to explore then and learn via trial-and-error before using git on large chunks of your work. GitHub has compiled a list of such resources here.

If you have no experience with git, you may find this Missing Semester lecture helpful. And if you want to build an intuition with git commands, try this card game.

2.2 Setting Up Git and GitHub

Before you start following the lab-specific instructions, complete the following set-up steps:

1. Install git. (See below for suggestions).
2. If you don't already have an account, you can sign up for an MIT enterprise account which allows you to host private repositories with all the features for free. You can also sign up for a personal account on GitHub if you prefer.

2.2.1 Installing Git

On Unix-like environments, installing git should be a simple `apt-get install`, `yum install`, or something similar with your package manager.

General instructions for installing git on different systems can be found at GitBook: Installing.

If you are using an IDE like IntelliJ/VSCode, it likely comes with git integration. The set-up instructions below may be slightly different than the command line instructions listed, but will work for any OS. Detailed instructions can be found in IntelliJ Help, VSCode Version Control Guide, and/or VSCode Github Guide.

2.3 Getting the Lab

You should have Git installed from the previous section. The instructions below use the **https** protocol remote url for the course repository. If you are working on a server where you cannot authenticate on GitHub with a browser and 2FA, you may need to setup SSH authentication (documentation). In that case, use the **ssh** version of the remote url where applicable.

1. Clone the course lab repository with the following commands from the directory where you'd like the lab to be located on your local machine:

   ```
   git clone https://github.com/MIT-DB-Class/go-db-2024.git
   ```

   This creates a **local** copy of the code repository that you can work on from the **remote** repository we are hosting on GitHub.

   If you do not wish to set up your own remote repo to version control, you can stop here and start working on the labs. In the future, every time a new lab or patch is released, you can `git pull` to get the updates. That said, we strongly encourage you to use git for more than just downloading the labs. In the rest of the guide we will walk you through on how to use git for version-control during your own development. *Using or not using version control and backup is at your own discretion - we will not in general provide grading accommodations for lost progress.*

2. Notice that you just cloned from the public course repo, which means that it will be inappropriate for you to push your code to it. If you want to use git for version control, you will need to create your own repo to write your changes to. Do so by clicking 'New' on the left on the GitHub web interface, and make sure to choose **Private** when creating, so others cannot see your code! **Note the remote url as [your-remote-url] of your new repository as you'll need it later.** Next, you are going to change the local repository you just cloned to point to your personal remote GitHub repository.

3. By default the remote called `origin` is set to the location that you cloned the repository from. If you run `git remote -v` from the local repository now, you should see the following:

   ```
   origin https://github.com/MIT-DB-Class/go-db-2024.git (fetch)
   origin https://github.com/MIT-DB-Class/go-db-2024.git (push)
   ```

   We don't want that remote to be the `origin` of your local repository. In the local repo, run

   ```
   git remote rename origin upstream
   ```

   to set the course public remote repository to be the `upstream` instead of the `origin`.

   Now running `git remote -v` should show the following

   ```
   upstream https://github.com/MIT-DB-Class/go-db-2024.git (fetch)
   upstream https://github.com/MIT-DB-Class/go-db-2024.git (push)
   ```

4. Now we need to give your local repository a new `origin`. Issue the following command in the local repo, substituting in the remote url you obtained from Step 2:

   ```
   git remote add origin [your-remote-url]
   ```

   If you have an error that looks like the following:

   ```
   Could not rename config section 'remote.[old name]' to 'remote.[new name]'
   ```

   Or this error:

   ```
   fatal: remote origin already exists.
   ```

   This appears to happen to some depending on the version of Git being used. To fix it, just issue the following command:

   ```
   git remote set-url origin [your-remote-url]
   ```

   For reference, your `git remote -v` should now look like the following if everything is setup correctly:

```
 git remote -v
upstream https://github.com/MIT-DB-Class/go-db-2024.git (fetch)
upstream https://github.com/MIT-DB-Class/go-db-2024.git (push)
origin [your-remote-url] (fetch)
origin [your-remote-url] (push)
```

5. Push all the current code on the main branch of the course repository to your private GitHub remote repo by issuing the following:

   ```
   git push -u origin main
   ```

   You should see something like the following:

   ```
    Counting objects: 59, done.
   Delta compression using up to 4 threads.
   Compressing objects: 100% (53/53), done.
   Writing objects: 100% (59/59), 420.46 KiB | 0 bytes/s, done.
   Total 59 (delta 2), reused 59 (delta 2) remote: Resolving deltas: 100% (2/2), done.
   To git@github.com:[your-repo].git * [new branch] main -> main Branch main set up to track remote
   branch main from origin.
   ```

   You should now also be able to see the code on the GitHub webpage of your own repository. Check again that the repository is not visible to the public. Now you are good to start working on the lab!

As you work on the lab, **staging** your changes and making **commits** allows you to snapshot your code and inspect or roll back to previous committed versions. Using **branches** allows you to try different implementations or only have clean, working code on your `main` branch. **Pushing** code to your remote repository backs it up and allows you to **pull** it from other devices and continue to work there. **checkout** allows you to switch between different versions of code, and **(caution)** to roll back unstaged (buggy) local changes.

Git is a distributed version control system. This means everything operates **offline on your local** until you run `git pull` or `git push` to synchronize with the **remote**. This is a great feature. However, one consequence of this is that you may forget to `git push` your changes. This is why we strongly suggest that you check your own GitHub repo to be sure that what you want to see matches up with what you expect after finishing a batch of work for backup purposes.

Git is in general immensely useful for managing a large, complex piece of code with many moving parts like GoDB, especially if you are collaborating with someone on the codebase. If you are new to git, please do not hesitate to post questions on Piazza about what to do when you are unsure.

2.4 Getting Bugfix Patches and Newly Released Labs

Pulling in future labs or any bug fixes we release should be easy as long as you set up your repository based on the instructions in the last section. All new labs from now on will be posted to the same class repository. Check it periodically as well as Piazza's announcements for updates on when the new labs are released. Once a lab or bugfixes are released, pull in the changes to the `main` branch of your local repository like this:

```
git pull upstream main
```

And sync the changes to your own remote repository:

```
git push origin main
```

If you've followed the instructions regarding git in each lab, you should have no **merge conflicts**. If you encounter merge conflicts and are not sure how to resolve them, please reach out to course staff.

3 Submitting Your Work

After you finish all the exercises, you will need to submit

1. All the code of the lab.

2. A short lab report writeup.

We will be using Gradescope to autograde the code of the labs. Check Piazza for the access code for Gradescope. If you are still unable to join the course in Gradescope, please reach out the course staff. You may submit your code multiple times before the deadline; we will grade the latest version as determined by Gradescope. Place the write-up in a file called **lab1-writeup.txt** with your submission.

If you are working with a partner, only one person needs to submit to Gradescope. However, make sure to add the other person to your group. Also note that each member must have their own lab report and work on it individually. Please add your Kerberos username to the file name and in the writeup itself (e.g., `lab1-writeup-username1.txt` and `lab1-writeup-username2.txt`).

The easiest way to submit to Gradescope is uploading `.zip` files containing both your code and the lab report writeups. On Linux/MacOS, you can do so by running the following command from the parent directory of your local repository:

```
zip -r submission.zip godb/ lab1-writeup.txt
```

If you are working with a partner:

```
zip -r submission.zip godb/ lab1-writeup-username1.txt lab1-writeup-username2.txt
```

This year, we also ask you to make a *checkpoint submission*, detailed in subsection 3.2.

3.1 Lab Report

Your lab report should be brief (maximum of 2 pages) and contain the collowing

- Describe any design decisions you made. These may be minimal for Lab 1.
- Discuss and justify any changes you made to the API.
- Describe any missing or incomplete elements of your code.
- Describe how long you spent on the lab, and whether there was anything you found particularly difficult or confusing.

3.2 Checkpoint submission

This year, to encourage you to start the lab early, we are setting up a checkpoint submission. To complete the checkpoint submission, upload your progress to the same Gradescope assignment. Your submission should pass 15% of the testcases on the autograder. Moreover, the checkpoint submission should contain a preliminary lab report writeup. You only need to report if there is anything you find particulatly difficult or confusing so far in this preliminary writeup (or just say ”everything is good so far”). A few days after the checkpoint submission, we will host a lab bootcamp to review relevant concepts and address issues that people commonly have trouble with. This checkpoint submission is so that you can get the most from the bootcamp and we have plenty of time to help you with the lab.

There will be no grading penalty if you do not complete the checkpoint submission. However, if you do pass 15% of the visible autograder tests and submit the preliminary lab report, you'll receive a sweet treat in the next week's lecture. We'll also take this into account as a bonus in the hollistic review portion of your participation grade.

3.3 Submitting a bug

Please submit bug reports to 6.5830-staff@mit.edu or via a private Piazza post. When you do, please try to include:

- A description of the bug.
- A `.go` file with test functions that we can drop into the ‘godb‘ directory, compile, and run.
- A `.txt` file with the data that reproduces the bug.

If you are the first person to report a particular bug in GoDB, we will give you a (larger) sweet treat!

4 Grading

75% of your grade will be based on whether or not your code passes the system test suite on autograder. These tests will be a superset of the tests we have provided. Before handing in your code, you should make sure it produces no errors (passes all of the local tests) when you run ‘go test‘ in the ‘godb‘ directory.

Before testing, Gradescope will replace the go test files with our version of these files. This means you should make sure that your code passes the unmodified tests.

You should get immediate feedback and error outputs for failed visible tests (if any) from Gradescope after submission. There may exist several hidden tests (a small percentage) that will not be visible until after the deadline. The score given will be your grade for the autograded portion of the assignment. An additional 25% of your grade will be based on the quality of your writeup and our subjective evaluation of your code. This part will also be published on Gradescope after we finish grading your assignment.

5 Lab 1

You will need to fill in pieces of code that is not implemented. For this lab, you will need to implement database tuples, heap pages, heap files, and complete the buffer pool implementation so that the buffer pool can get heap pages. You may find the lecture and reading material on database internals particularly useful. `lab1.md` in the course repository walks you through the code and the tasks you will need to complete.

6 Getting Help

If at any point you need help with with lab, feel free to post on Piazza or reach out to one of the TAs or the instructor. Their contact information can be found on the course homepage. There will be office hours throughout the next two weeks and a bootcamp session where we review the concepts covered in the lab and common problems people run into together. Look out on Piazza and the course website for these plans.