NanoDB Setup

Setting up NanoDB

Ensure your system satisfies the requirements below before starting NanoDB projects.

IDE

We recommend you either use VSCode or IntelliJ, with a slight preference towards IntelliJ.

Make (Optional)

NanoDB includes a Makefile for convenience (e.g., running build and test commands), but **make is NOT strictly required**, especially on Windows. We've also set up IntelliJ files for the test commands in each project.

If you prefer not to use `make`, you may run the equivalent commands manually (which you can see in the Makefile). Note that the Maven CLI does **not** automatically recompile if you run the test target commands. You'll need to explicitly recompile; see the `build` target in the Makefile for the command.

Check if make is already installed:

```
make --version
```

If not...

- **MacOS**: Run `xcode-select --install` in your terminal.
- **Linux**: Should come packaged with most distributions by default, otherwise follow directions for your package manager.
- **Windows**: If you want to use the Makefile, you'll need to use WSL to have a POSIX shell like bash.

JDK 21

NanoDB requires Java 21. Install and follow setup instructions for JDK 21 from: https://www.oracle.com/java/technologies/downloads/archive/

After installation, verify that everything points to JDK 21:

```
java -version
javac -version
```

Both should report Java 21. If you have multiple JDKs installed, you may need to configure your OS or shell environment (e.g., `JAVA_HOME`, `PATH`) so that Java 21 is selected by default.

Finally, ensure your IDE is configured to use JDK 21.

Maven

Install Maven following the official instructions: https://maven.apache.org/install.html

Verify with, and ensure it is running on Java 21 as well:

```
mvn -v
```

Cloning the Repo

Sign up for the repository at the Project Registration link on the course website, which will create a new repository for you on GitLab. You will work in the same NanoDB repository for the entire term. Occasionally, we may push updates to your repo, and you'll need to pull this code. We will announce whenever this happens, which is typically at the beginning of each project.

Working with NanoDB

Building NanoDB

NanoDB is built using Apache Maven. Maven uses a file named `pom.xml` to configure the build process. This file contains a lot of details, including the dependencies that NanoDB requires, custom build steps, and so forth. When Maven runs, it will download all necessary requirements for the project into a "`.m2`" subdirectory in your home directory. Maven can build various targets in a project "life cycle"—to that end we have configured a Makefile for key targets. In IntelliJ, these targets will also be available as run configurations.

- `mvn clean` will delete all generated build artifacts. Usually these artifacts are under a `target` directory.
- `make build` will build all of the project's source code and test code, and if this entire process completes successfully, a JAR (Java ARchive) file is generated in the `target` directory.
- `make run` will run the `./nanodb` script running the NanoDB CLI.

Using NanoDB

NanoDB can be started with the `make run` target. When you start up NanoDB at the console, you will be greeted with a simple prompt:

```
$./nanodb
Welcome to NanoDB.  Exit with EXIT or QUIT command.

CMD>
```

This is where you can issue SQL commands against the database. Commands are generally case-insensitive. **Note that all commands must end with a semicolon ";" character.**

As the program says, you can type "`exit;`" or "`quit;`" to exit the database. You can also exit with Ctrl-D (end-of-file) or Ctrl-C if you wish. Sometimes if NanoDB is being cranky, Ctrl-C may be your only option!

If you ever run into behavior that you think is a bug, please let the Ethan and/or the TA's know so that we can fix it for the class, and/or give you a workaround until a fix is available. NanoDB has many bugs, and you will likely encounter at least a few of them during the term.

`rlwrap` (Optional)

`rlwrap` is a utility on Linux and MacOS that helps provide more user-friendly text-editing and command-scrollback when installed. You can type `"which rlwrap"` to see if it is available on your computer, and then install it if you don't have it .

You don't need to do anything special to get NanoDB to use `rlwrap`; the `nanodb` shell script will automatically check for `rlwrap` and use it if it is present.

NanoDB Configuration Properties

Some of the database configuration is exposed as *properties*. You can see all available properties by typing:

```
CMD> show properties;
+----------------------------+--------------------------------------------+
| PROPERTY NAME              |                                      VALUE |
+----------------------------+--------------------------------------------+
| nanodb.baseDirectory       |                                ./datafiles |
| nanodb.createIndexesOnKeys |                                      false |
| nanodb.enableIndexes       |                                      false |
| nanodb.enableKeyConstraints|                                       true |
| nanodb.enableTransactions  |                                      false |
| nanodb.flushAfterCmd       |                                       true |
| nanodb.pagecache.policy    |                                        LRU |
| nanodb.pagecache.size      |                                    1048576 |
| nanodb.pagesize            |                                       8192 |
```

```
| nanodb.plannerClass         | edu.caltech.nanodb.queryeval.SimplestPlanner |
+-----------------------------+----------------------------------------------+
```

Some of these properties can be modified while NanoDB is running. Note that property names must be enclosed in single-quotes, and property names are most definitely case-sensitive.

For example, to change the default page-size used by NanoDB, you can type:

```
CMD> set property 'nanodb.pagesize' = 4096;
Set property "nanodb.pagesize" to value 4096
```

Page sizes must be a power of 2 between 512 and 65536; if you type an invalid value, NanoDB will tell you:

```
CMD> set property 'nanodb.pagesize' = 4000;
ERROR:  Specified page-size 4000 is invalid.
```

Other properties are read-only during normal execution:

```
CMD> set property 'nanodb.baseDirectory' = './datafiles2';
ERROR:  Property "nanodb.baseDirectory" is read-only during normal operation, and should only be
set at start-up.
```

Properties like these can be set either by editing the NanoDB startup script (`nanodb`), or by specifying configuration at startup in the Makefile:

```
$ ./nanodb -Dnanodb.baseDirectory=./datafiles2
Welcome to NanoDB.  Exit with EXIT or QUIT command.

CMD> show properties;
+-----------------------------+----------------------------------------------+
| PROPERTY NAME               |                                        VALUE |
+-----------------------------+----------------------------------------------+
| nanodb.baseDirectory        |                                   ./datafiles2 |
| nanodb.createIndexesOnKeys  |                                          false |
| nanodb.enableIndexes        |                                          false |
| nanodb.enableKeyConstraints |                                           true |
| nanodb.enableTransactions   |                                          false |
| nanodb.flushAfterCmd        |                                           true |
| nanodb.pagecache.policy     |                                            LRU |
| nanodb.pagecache.size       |                                        1048576 |
| nanodb.pagesize             |                                           8192 |
| nanodb.plannerClass         | edu.caltech.nanodb.queryeval.SimplestPlanner |
+-----------------------------+----------------------------------------------+
```

Redirecting SQL Scripts into NanoDB

You can also redirect SQL files into NanoDB. When this is done, NanoDB will suppress its human-friendly prompts so that you don't see an annoying sequence of "`CMD> CMD> CMD> CMD> CMD> CMD> CMD>`" prompts on the console. Examples of how to do this are given in assignment guides, but a simple example might be:

```
$ ./nanodb < somedata.sql
```

Notes on NanoDB

As you explore NanoDB this term, there's a few quirks that you should be aware of.

TODOs

NanoDB is definitely still a work in progress, and as such, there are numerous places where the code has comments like "`// TODO: Implement some thing`", or "`// BUGBUG: Fix some thing`". **You only have to implement the features specified by each assignment.** We usually provide "`TODO`" comments or "`UnsupportedOperationException`"s to show you where to put your code.

If you ever have any questions about what you're responsible for, please ask Ethan or a TA. We want to make sure you only do the work that you're expected to do!

Exceptions

If you are familiar with Java then you are probably also familiar with the difference between checked exceptions and unchecked exceptions. Checked exceptions are checked by the compiler; if a piece of code can throw a given exception, then the code must either handle that exception, or declare that the exception may be thrown. Unchecked exceptions are not checked by the compiler, and generally indicate abnormal issues not expected to occur in normal operation (e.g. due to programming bugs). In Java, any exception that is castable to `java.lang.RuntimeException` is a **runtime** exception. Any exception that is derived from `java.lang.Exception` but is not derived from `java.lang.RuntimeException` is a **checked** exception, such as `java.io.IOException`.

Databases are an unusual kind of software. Many things can fail, for many different reasons. For example, the simple act of allocating a buffer can cause an IO error if the Buffer Manager must evict data pages to free up space, and the write of one of those data pages fails for some reason. This means that basically all of the code can fail.

This is not actually a problem, because if a failure occurs, we try to roll back the current transaction. This means that all failures are handled at the very top level of command execution (in the `NanoDBServer.doCommand(Command)` function). So, since most of the code can fail, and most failures are handled at the top level of the code, we simply catch and convert checked exceptions to runtime exceptions for most parts of NanoDB. It's an easy solution, and doesn't require us to write "`throws Exception`" everywhere in the code.

Logging

NanoDB uses the Apache Log4j 2 framework extensively to provide numerous details of database execution. You will see the result of this logging in the files named "`nanodb*.log`", with "`nanodb.log`" being the most recent logs. The config file that controls what components and logging levels are included is called "`log4j2.properties`". We encourage you to edit this file to tune the specific information that is included. If you have a confusing failure, the logs are a good option for understanding what was going on along the way.

Java API Documentation

NanoDB includes extensive Javadoc documentation in the codebase, which can be a very handy reference for understanding what different classes do. You can generate the documentation on your local system by running "`mvn site`" (or building the `site` step through IntelliJ IDEA).