Seminar 4 rehandin

Edvin Frosterud

January 2023

Changes made will be italicized.

1 Introduction

Seminar 4 is all about programmatic access to queries and databases. I worked with Yas Asghari again, but this instance our work was more split up due to the Student Nobel Nightcap she was working, so some stuff might be different. Anyhow, this report will be pretty quick due to time constraints by this course and our other course that requires a lot of time.

2 Method

We're using Java, VSC and appropriate extension and library for PostgreSQL, pgJDBC. Queries were made by the author of this, while the code was handled together with Discord or personally. Some functions required more than one query, in this case the renting of instruments, so for that we split it up into mulitple queries, both for ease of reading and because that way we can run a smaller query before even starting to insert values in a second query. The primary reason was really that I couldn't get one query that works by it self.

First, the view components of the commandline from the lectures was used as our view, as recommended by Leif. After the views were there, we needed to write an integration, Controller, and modify the existing views BlockingInterpreter for taking the expected commands for the required actions. Command enum was also changed to be the correct commands. To follow the MVC pattern we needed to not print and have logic in the integration, so a lot of changes needed to made in the later stages of the project to account for this.

In my opinion the DAO was the hardest, especially converting string sqls to variable SQL statements that can be edited and don't have locked table/column names, because it caused a lot of hard to read errors because it doesn't specify line or what the problem was. A handleException was directly taken from lectures too, since only variable names there need to be changed, the whole thing can most likely just be swapped for a catch that just rollbacks.

3 Result

`https://github.com/Froosty11/IV1351` *Github has been updated with instructions from canvas.* All relevant files for this assignment can be found in this github repository, you will need to create the database first(found in sem2), then run the fill-database.sql script using psql import. Here is a simple output, following someone terminating two already existing guitars and then renting one of them on student ID 3. This image above doesn't take ACID implemen-

Figure 1: First, user asks what commands exist. They then ask for all non rented guitars. Two leases are terminated, this is using the lease ID and then the user asks to rent a specific guitar of the 3 that now exist.

tation into account, as all queries are working without a cancellation. The acid is implemented in a very similar way to the lectures. The idea of acid is to not let a single operation be half done, therefore we're rollbacking if an exception happens or if another query fails to find the answer we're looking for. In practice this just means don't commit until the end, and disable autocommit as that is default on. When the transaction is executed it is fully consistent and all values that relate to eachother change in according to the original change. Isolation is kept by seperating transactions from eachother with multiple SQL statements for some functions. Values aren't moved between transactions, and are instead checked again after transaction is done. Data is persistant.

I think result wise the command returns/prints could be slightly cleaner, perhaps giving the user an idea that their command didn't fail when a return happens. Currently the successes don't print anything, while the failures do print, but this is unclear. I would also like to add syntax support to help, as to be able to know what to put where. I'll include in a readme with the syntax in the repos.

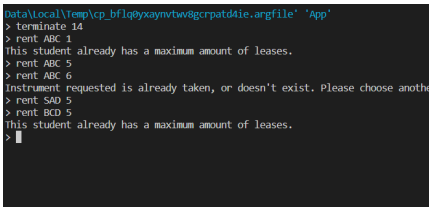
4 Discussion

As far as I'm aware the written code corresponds very easily and is correct according to MVC model. Something that I have discussed with peers is the

correctness of the current renting system. Currently, in the DAO there is a small if check that controls whether or not to fire the secondary update queries. This could be handeled with just the try-catch and throwing an exception if a selection is wrong, for example if the student selected has two leases on them. We've discussed whether or not this counts as logic, but in the DAO on the bank system designed in lecture the bank system did a similar solution, so I do think it's correct. If not, I do not think we should be taught the wrong operation as a bait.

Autocommit is disabled as soon as the DAO has started a connection. This allows us to fallback before commiting. A commit only happens after a try catch has completed, and all inner private methods/queries do throw an exception to make sure the above method knows to rollback.

Select for update is used to block other users from accessing the same database. I've also discussed with Leif about our implementation of the lock but it seems fine after some back and forth. Another criticism we got I've completely ignored as it was wrong. Previously, it wasn't possible to add more instrument leases than two, even though Niharika said it was. Maybe I'm confused to what she was saying but I haven't changed anything in that perspective, but see this following code for a clear representation of how it work now, regardless or not if the version on github previously was broken?



All requirements are in there, but I would like to improve the requirements for the removal of a lease, as the requirements for that are a bit loose I use the leaseid which is allowed, just a bad choice since it requires database access.

A terminated lease is stored by changing the endtime to the termination time, and setting terminated to true. Other queries ignore terminated queries. InstrumentItems leaseid property is reset so that it can be found as unused. I would like to store the old instrumentID somehow, but as of writing this is not supported. If I have time before deadline I'll fix this.

I do believe the advanced higher grade task requirements are followed, but as previously debated that does require that the if statement doesn't count as logic. This would be fixed by moving this to controller- but it is very vague and my peers seem to be unsure whether or not to do that.

5 Sources

Youtube channel Leif Lindbäck(Link: https://www.youtube.com/channel/UCs3aBf$_5KdxjKnNVFk_HrSQ$)