1. Briefly describe the artifact. What is it? When was it created?

The artifact, like before, is a text based game in which the player that has the player collecting items while avoiding the boss enemy. Collecting all items before boss encounter wins the game, while encountering the boss before that loses the game. Player can choose to exit early, and has to make decisions in a turn-based gameplay loop. It was created around the start of my time at SNHU, and has been remade as part of this course.

1. Justify the inclusion of the artifact in your ePortfolio. Why did you select this item? What specific components of the artifact showcase your skills and abilities in software development? How was the artifact improved?

In this category, I planned to:

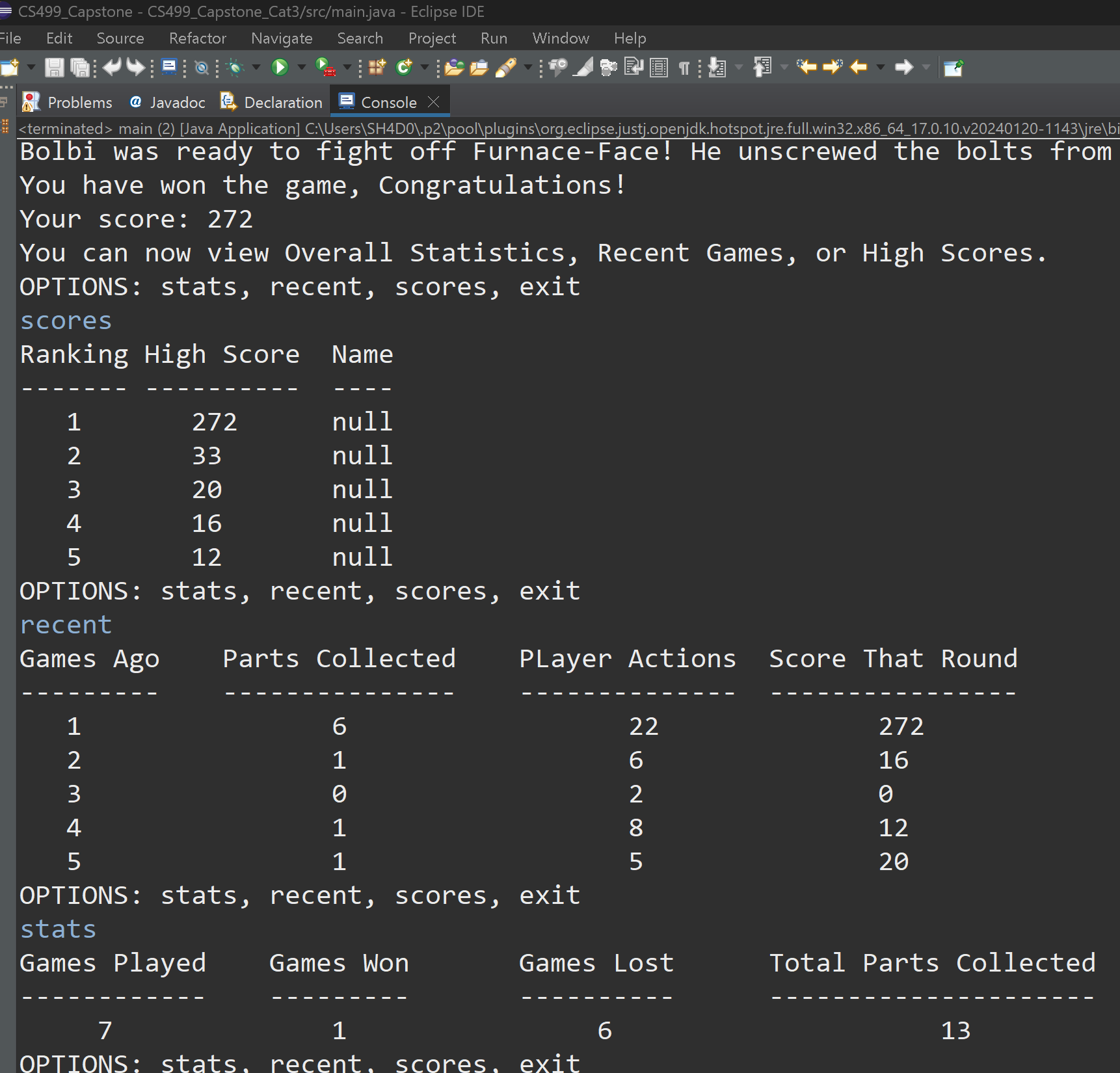
(1) Incorporate a database to store statistics about the game

(2) create a scoring system based on gameplay

(3) potentially customize the naming of the high scores

The database component required a working knowledge of SQL to implement, as well as understanding of database structures in order to create one that suited the needs of this program. Both of these were demonstrated as I had to plan out what data was to be stored, along with the CRUD methods that would be used to create, retrieve, update, and delete that data.

The database was successfully implemented, and **now players** not only **have a score** at the end, but they **can see their statistics compared to previous games**, with the **best of their scores being saved until they are surpassed**. The previous randomization combined with the new scoring system gives players a reason to replay, to improve upon past performance and see the results of their efforts reflected in the scoreboard.



1. Did you meet the course outcomes you planned to meet with this enhancement in Module One? Do you have any updates to your outcome-coverage plans?

Yes, the 3rd outcome was addressed because I used algorithmic principles to design a solution (database) while managing trade-offs between different approaches and data structures. The 4th outcome was also fulfilled because I had to use innovative techniques to implement computer solutions that deliver value. The 5th outcome was addressed by having a security mindset, where I mitigated design flaws (fixing a game breaking-bug with adversarial potential) and ensured that the database was only open when in active use and closed to potential tampering outside of that intended use.

1. Reflect on the process of enhancing and modifying the artifact. What did you learn as you were creating it and improving it? What challenges did you face?

The process was similar to my previous “sprint” where I identified what needed to be done and outlined what I would like to accomplish within the timeframe I had. I added notes onto the original outline as I worked to keep my thoughts together.

*Improvements planned for Database category:*

* Database:
  + Access at endgame condition
  + Create if not present
  + Store one entry global (total) stats GLOBAL\_STATS
    - * CREATE once, UPDATE every run
      * Games
      * Wins
      * Losses
      * Parts Collected
      * Player Actions
  + Store 5 most recent stats under RECENT\_GAMES
    - UPDATE 1st recent, shift others +1, cutoff at 5
      * Parts collected
      * Player Actions
      * Score
    - Initialize 5 rows with (1... , Null, Null, Null)
    - QUEUE: FiLo, 6=5;5=4;...1=current; DELETE 6 too complex, inefficient
    - instead loop (GAMES\_AGO+=1) then DELETE 6, insert GAMES\_AGO=1
  + Store 5 high scores under HIGH\_SCORES
    - READ (select \*) every run (option to skip), compare, then UPDATE if better than
      * Criteria:
        + Total actions to victory: less = better?
        + ( Parts collected that round / Actions that round ) \* 100 for failure or \* 1000 for victory
        + Theoretical limit 500 points... 6 items, 2 moves required for each assuming no gaps and no extra moves. 6/12 \* 1000 = 500.
        + BE CAREFUL WITH DIVISION

Make 0 divisor impossible

Flow error not likely, answers always between 0 and ~500

* + - * + Ranking, decide to start with 0 or 1
    - Initialize (1..., 0, AAA...)
    - INSERT WHERE DISTINCT currentScore > MIN(SCORE) unnecessary
    - ResultSet rs = stmt.executeQuery (SELECT \* FROM HIGH\_SCORES ORDER BY SCORE DESC)
    - for (int i = 0; i<MAX\_TO\_KEEP; i++) {(if currentScore>rs.getInt(“SCORE”) //insert here with this ranking?

*If there’s time:*

* Allow User-defined names with high scores (VARCHAR 3)
* Fix rare bug that crashes game when Player and boss tried moving simultaneously

I decided on using SQLite JDBC library for Java: <https://github.com/xerial/sqlite-jdbc>

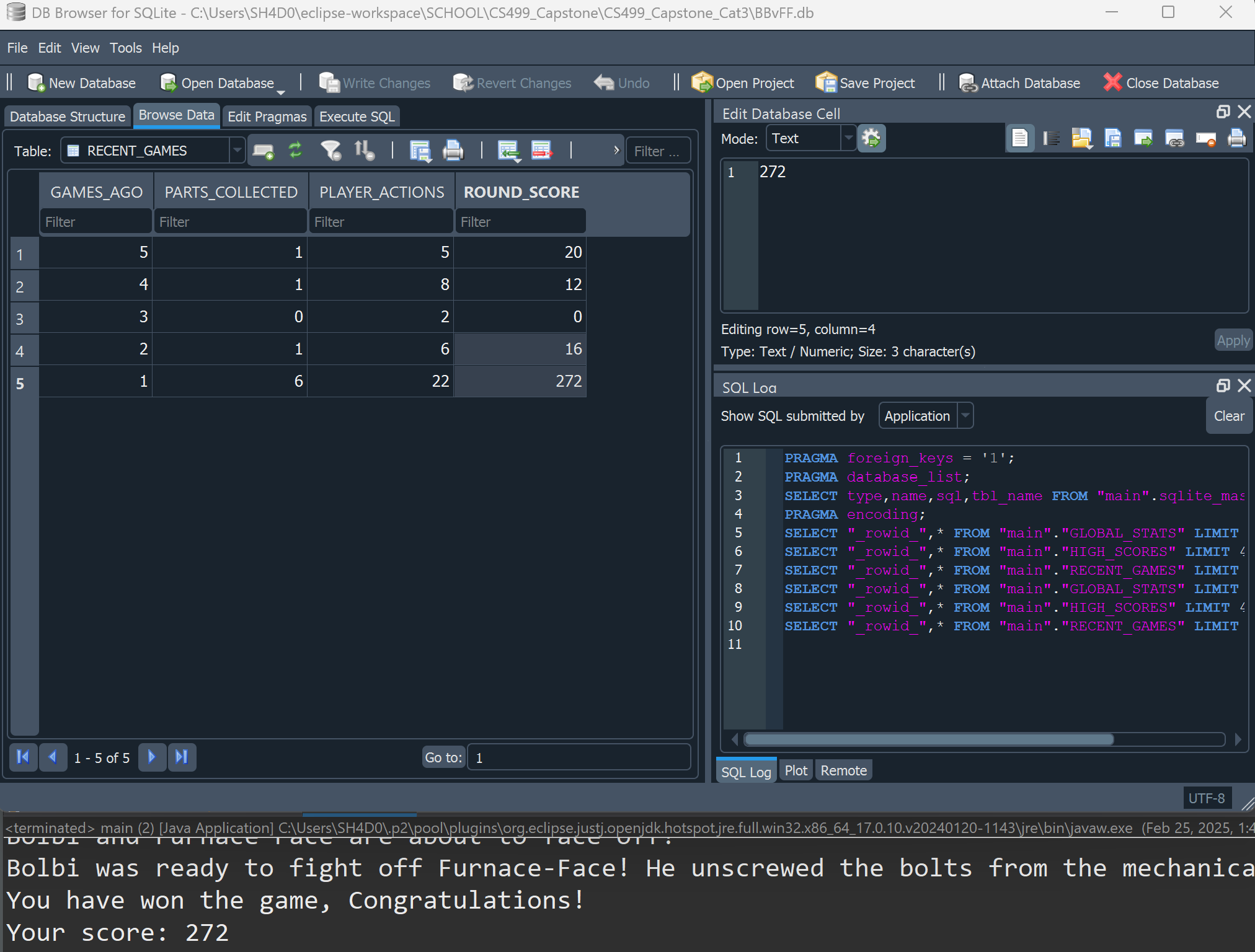
I read the documentation for how to use connections and statements and adapted the SQLite queries to the needs of my program. Some useful examples were found on <https://www.tutorialspoint.com/sqlite/sqlite_java.htm>

This video covered the installation process: <https://www.youtube.com/watch?v=0beocykXUag&ab_channel=LogicLambda>

**In eclipse, right clicking on the project>properties>Java Build Path>Libraries>Classpath>Add External JARs allows the sqlite jdbc driver to be integrated into the program.**

I also found a database browser to visually verify the results of operations after runtime:

<https://sqlitebrowser.org/about/>



During the process, I ran into a number of problems and errors, some of which was due to program code, and some of which was due to the SQL queries.

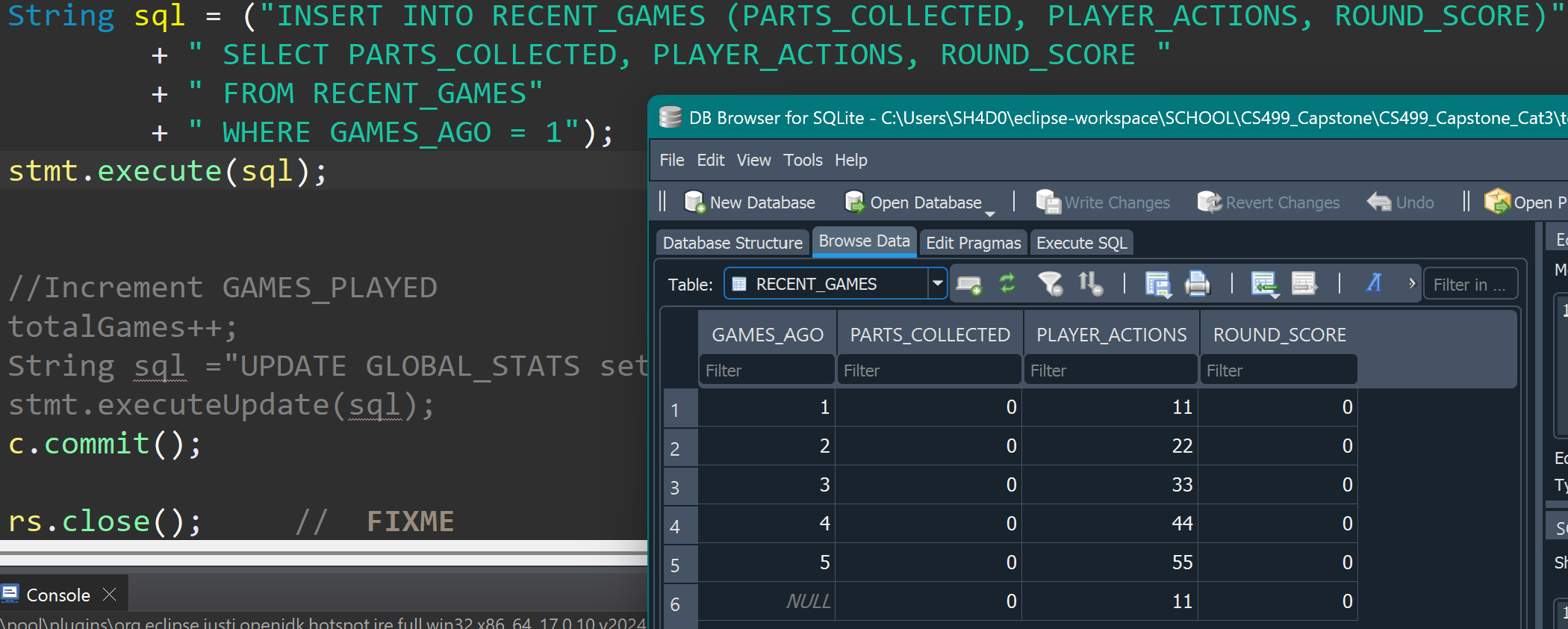
In one instance, I forgot to put commas between column names, ended up with one super-column and got “incorrect syntax near 1” error.

While building out the database CRUD methods, I got errors saying the database was locked, or closed. In these cases, I forgot to close connections or database objects, or I accidentally closed them before they could be used.

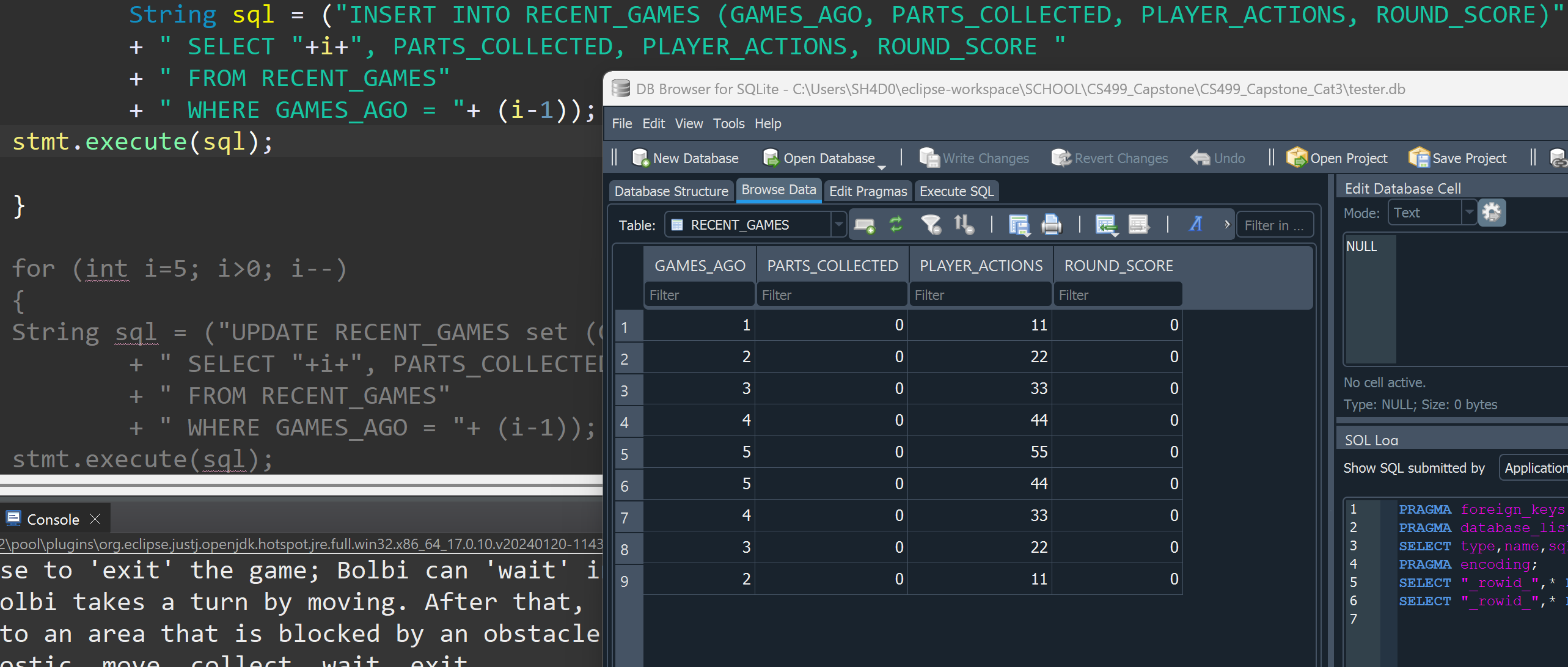
I decided on a scoring system based on ItemsCollected divided by PlayerActions multiplied by 1000 if all parts were collected, or by 100 for not finding all parts. However, I found that calculation was resulting in 0 score consistently. This is because in division, I needed to typecast integers as floats or some decimal enabled variable, as otherwise every resulting fraction would be 0 multiplied by something which is always zero.

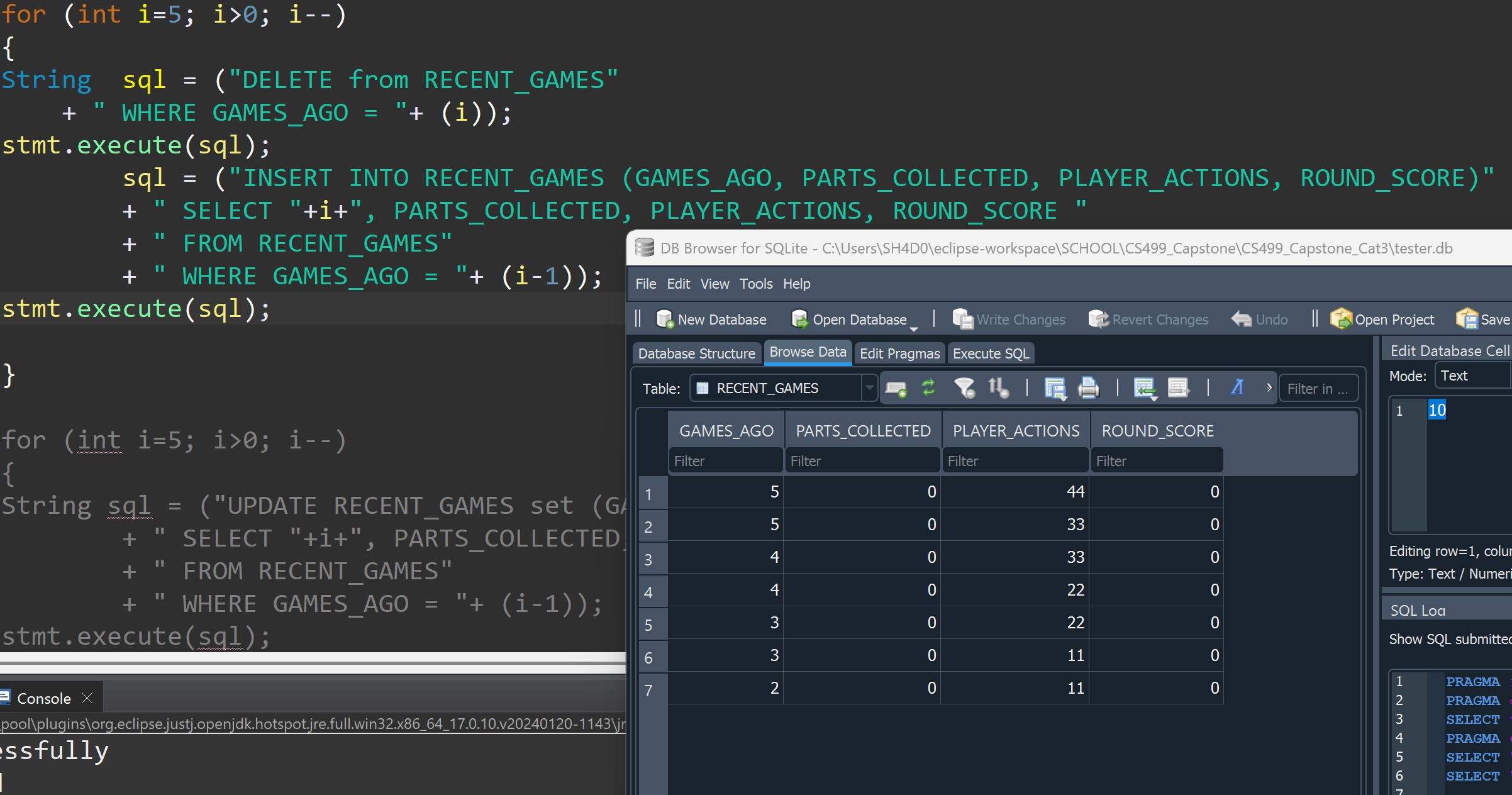
I tried multiple approaches to initializing and updating the database. At first, I tried copying the data from one entry and applying it to the next in a loop, but these were inefficient and left unwanted duplicates or dangling data beyond the desired bounds.

Trying to shift rows down by one, to make room for the newest:



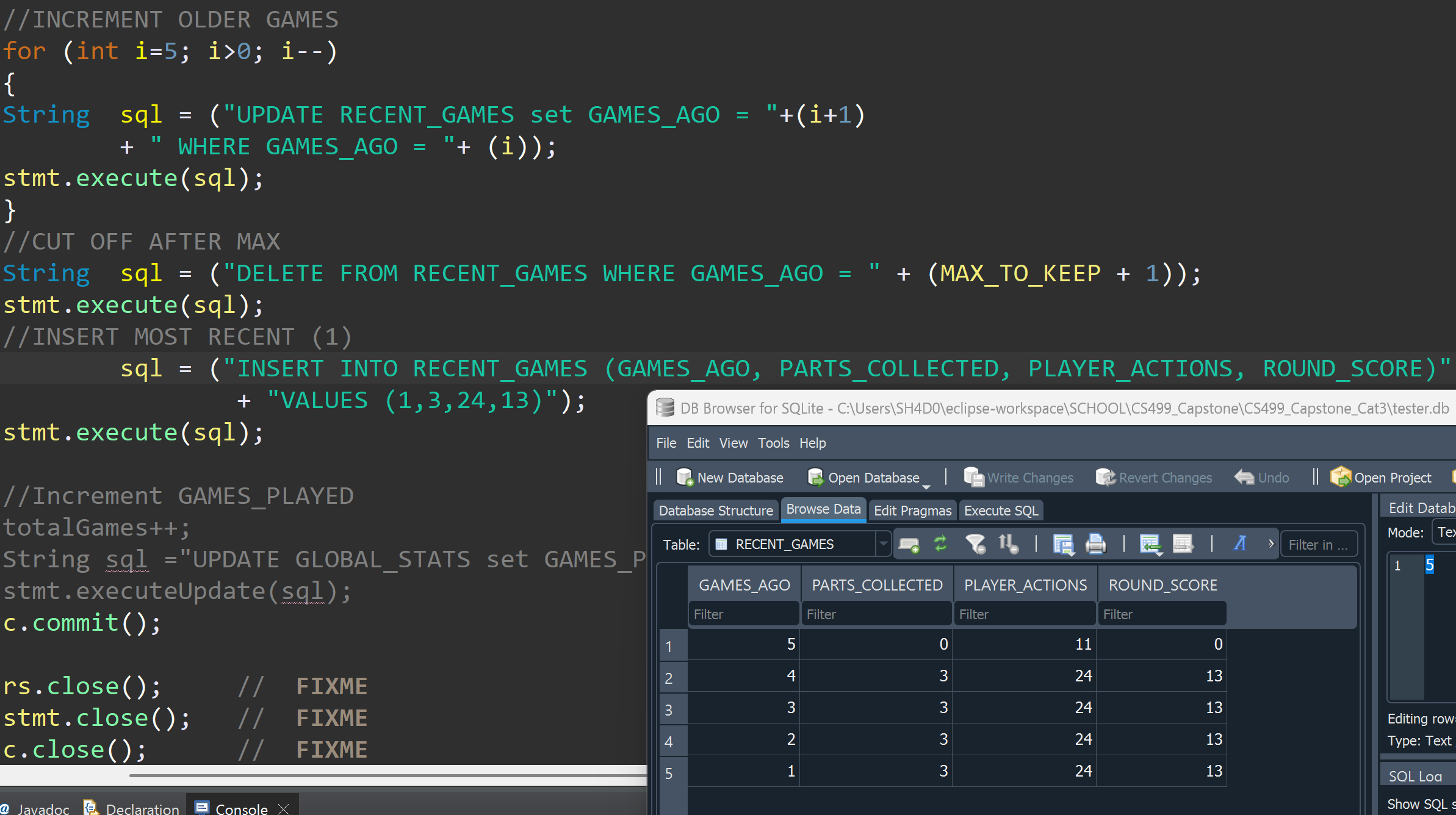
Tried looping and shifting, ended up with duplicates of all but 1:

Tried deleting the one after?

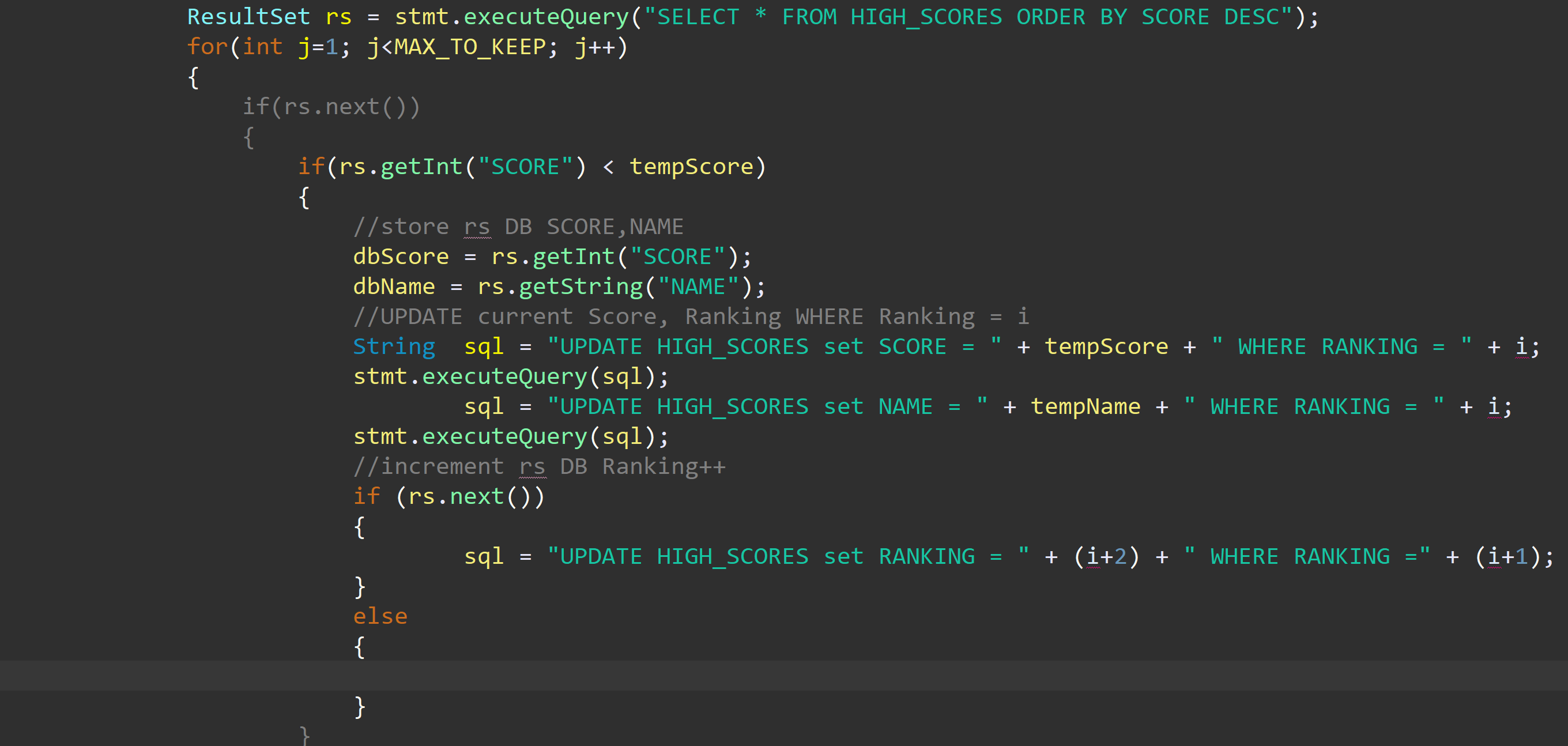


I was overthinking, and determined it was not efficient to sort within the database.

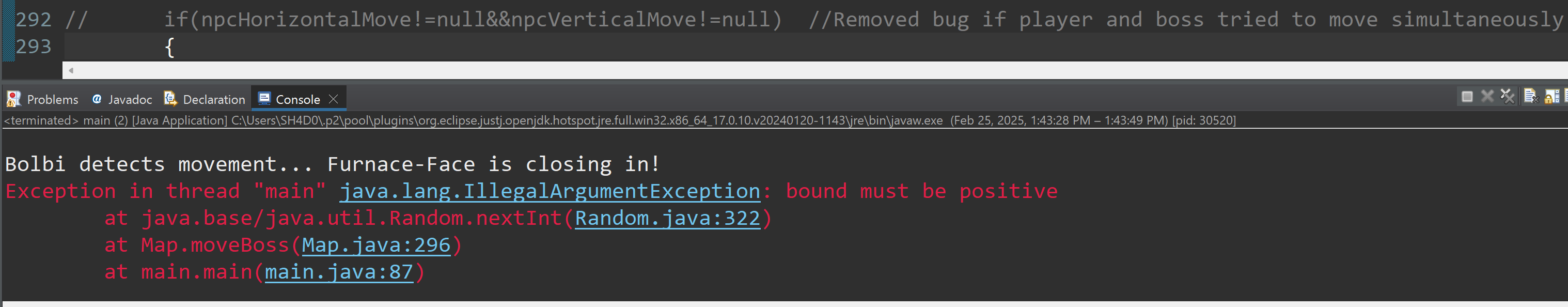
I then started simply incrementing the ordering column variable and inserting at the beginning, after cutting off the end. This created a First in-Last out Queue structure.



I overcomplicated it again before realizing I could also apply the same principle to something that needed sorting like the High Scores. Select it sorted the way I want, compare until I find the insertion spot, and repeat the previous Queue pattern after that index. This worked perfectly.

 Finally I fixed the simultaneous boss player move fatal error by introducing a check for null or invalid movement options being passed on through the NPC decision logic.

if(npcHorizontalMove!=null&&npcVerticalMove!=null)



Correction: if(npcHorizontalMove!=null||npcVerticalMove!=null)

Checking both made it so the boss would never catch up to the player, but has been fixed.

The only thing I did not finish was the implementation of user-defined names, though I left room for names in the high scores table of the database.