**Software Documentation**

Data sources

We used two sources for our data:

* We downloaded csv data entries from https://www.kaggle.com. The database is using data from the movies website IMBD.
* We used tmbdv3api as our API source for the movies data.

Database

Our database design is as follows:

* Actors(actor\_id, actor\_name, bio)
* Movies(movie\_id, title, movie\_year)
* Movies\_budget(movie\_id, budget)
* Movies\_countries(movie\_id, country, language)
* Movies\_genres(movie\_id, genres)
* Movies\_productions(movie\_id, director, writer, productions, actors, description)
* Movies\_reviews(movie\_id, reviews\_from\_users, reviews\_from\_critics)
* Movies\_votes(movie\_id, avg\_votes, votes, critics\_votes)
* Ratings(movie\_id, male\_avg, female\_avg)

We wanted to create small, dynamic, and readable tables which we could easily read and analyze. Each table catches one aspect of the movie production we wanted to capture, and our analysis will use several joins to answer the questions we want to ask.

A total of several tens of thousands of data entries were entered into the project.

Queries

We designed 8 different queries:

1. Find the highest rank movies with at least X ratings.
   * We get the data from the connection between the movies table and movies\_votes table.
2. From the table mentioned above, get the movies with a least X reviews.
   * We get the data from the connection between the movies, movies\_votes and movies reviews table.
3. Get the most popular genres movie based on their ratings.
   * We get the data from the connection between the movies, movies\_genres and movies\_votes tables.
4. Get the most popular films by the name of the actor that played.
   * We get the data from the connection between the movies, actors and movies\_productions tables.
5. Get the highest country film producer and the movie's ranking.
   * We get the data from the connection between the movies\_votes and movies\_countries tables.
6. Get the best movies based on their topics.
   * We conducting several union calls and doing full text search from the movie\_productions table.
7. Get the countries with the most budget for their movies and their average rankings.
   * We get the data from the connection between the movies\_countries, movies\_budget and movies\_votes tables.
8. Get the most popular movies based on viewer gender.
   * We conduct 2 unions from the data we connect between ratings and movies tables.

Optimization

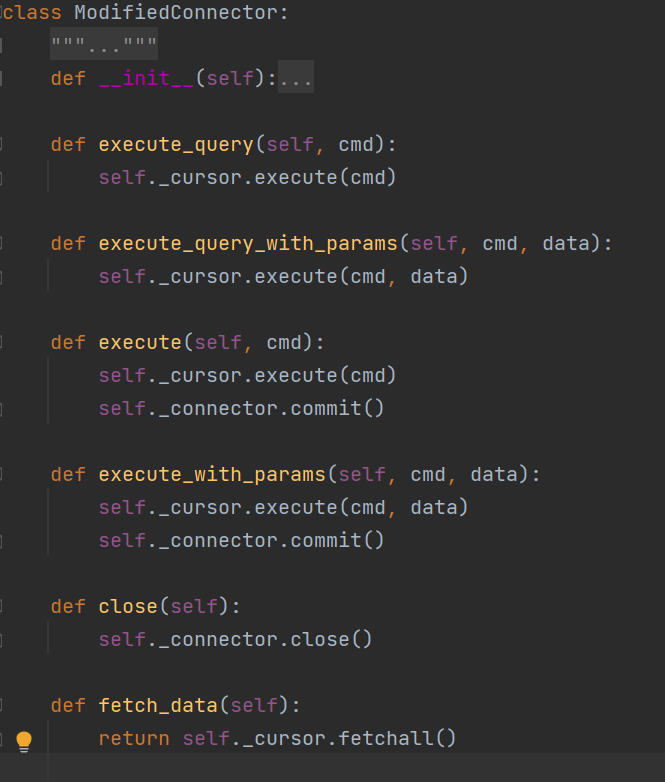
We created several indexing for the full text search we are conducting to optimize our queries.

* In movies\_productions table we index the actors and descriptions columns because we will try to match the strings.
* In actors table we index the names and bio because we will be doing string matching with those attributes as well.
* In addition we indexed the votes in movies\_votes table. We are using the votes in several queries and it will improve our running time.

Those index selections making all our queries run pretty fast and getting the execution results we aimed for.

We will go over our code to review its functionality.

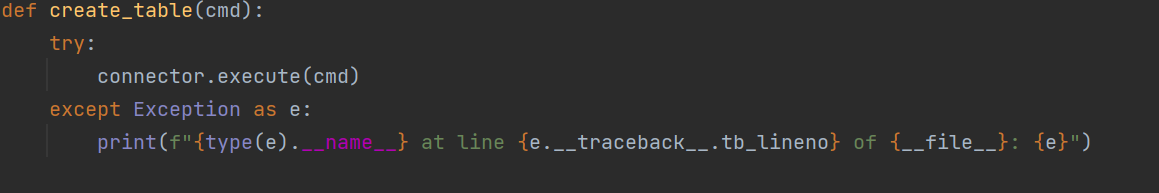
Modified Connector



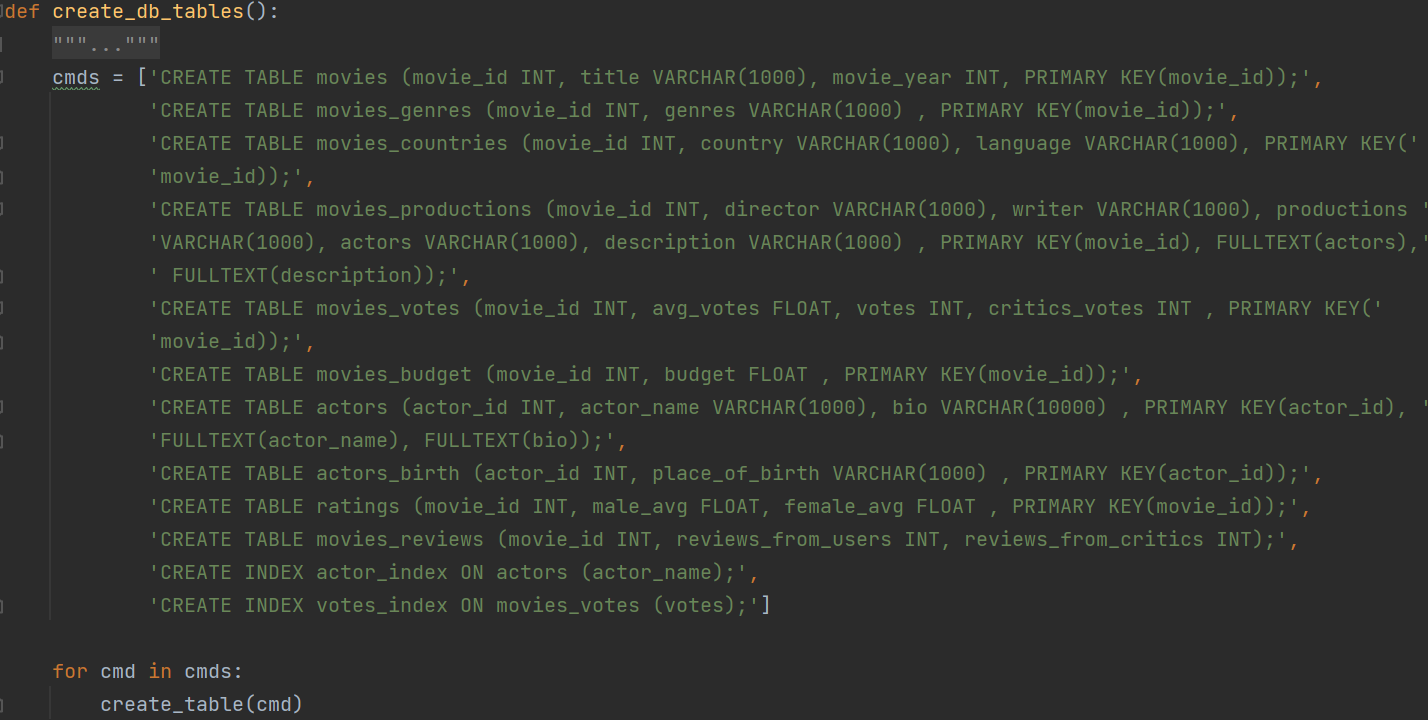
We use this class to envelope the MySql connector class.

DB creation

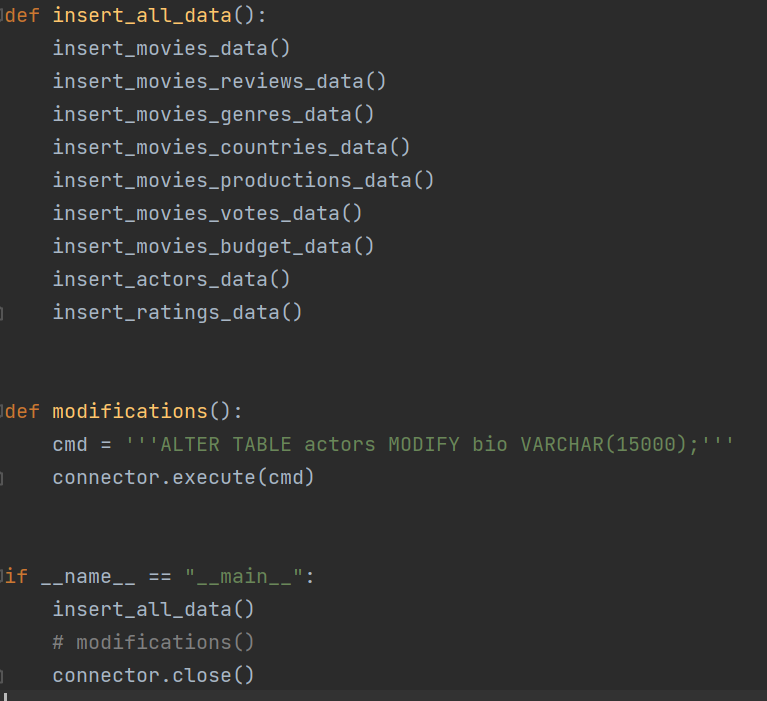
The create table function is



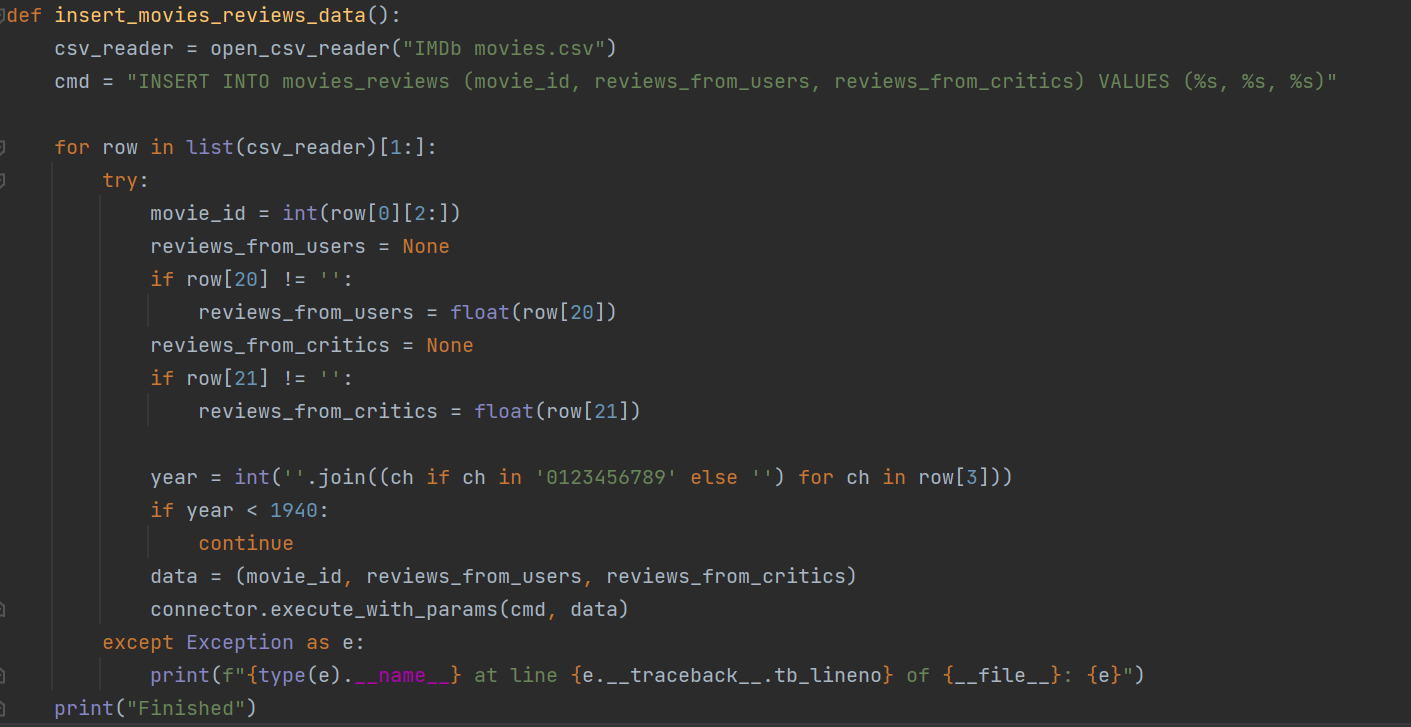
To create all the tables we use:

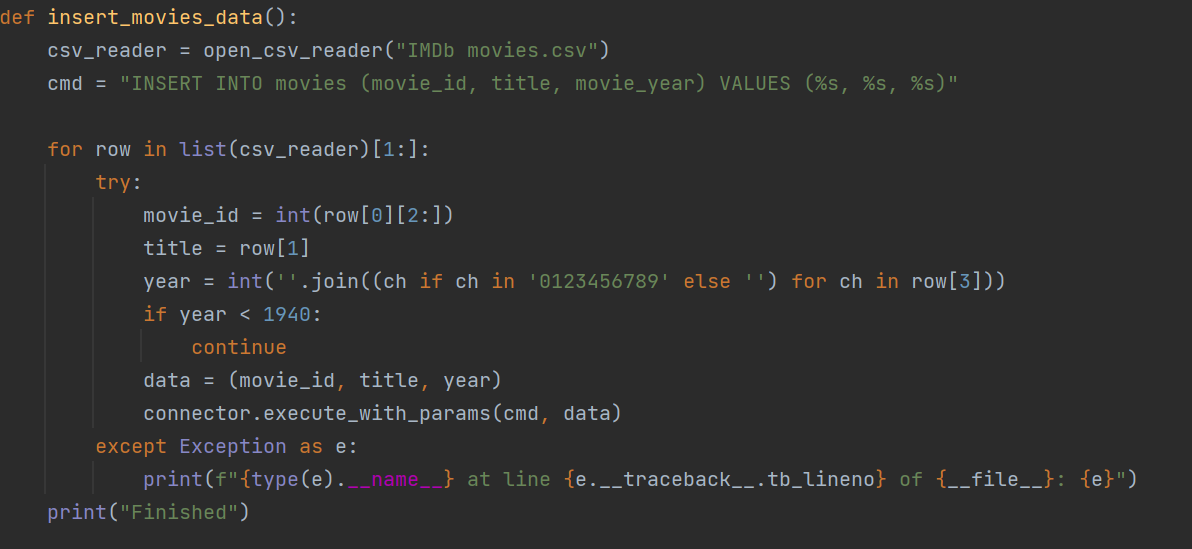


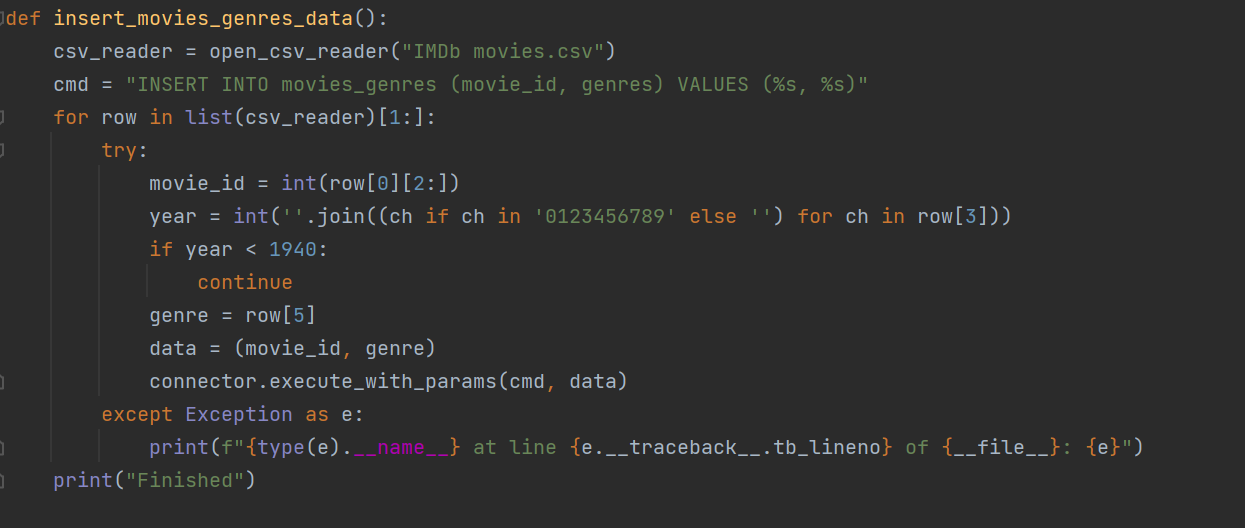
Data insertion



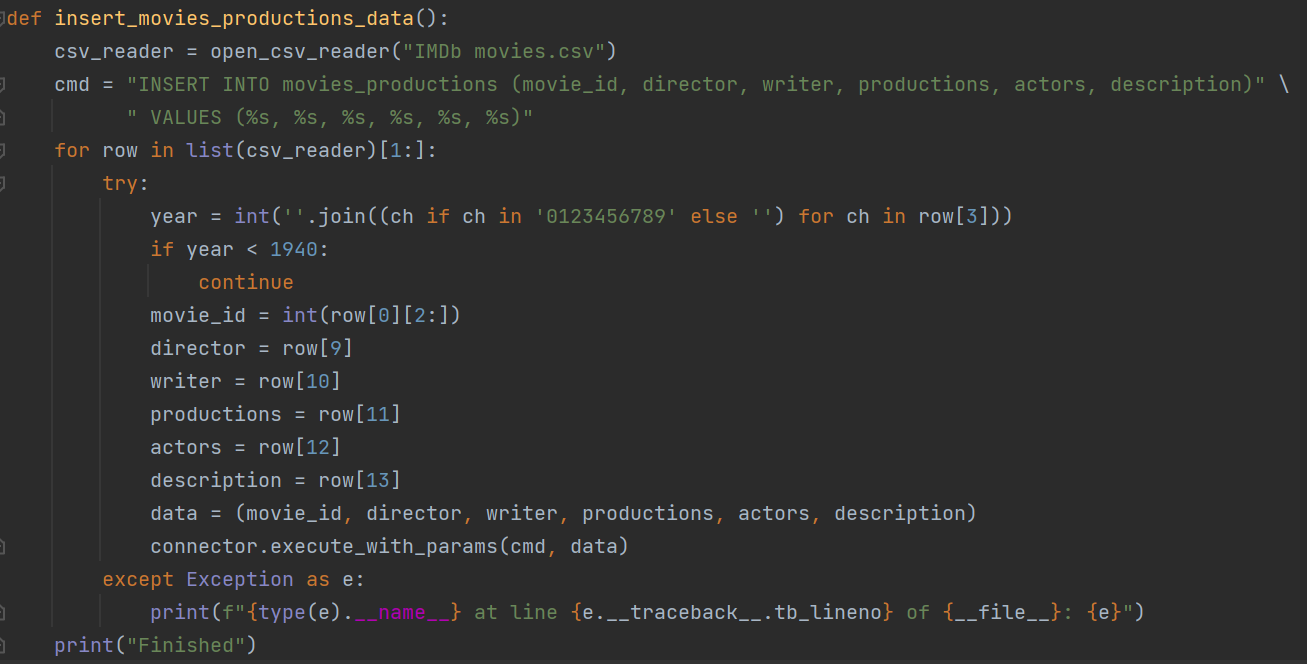
We call this function to fill all the tables’ data.

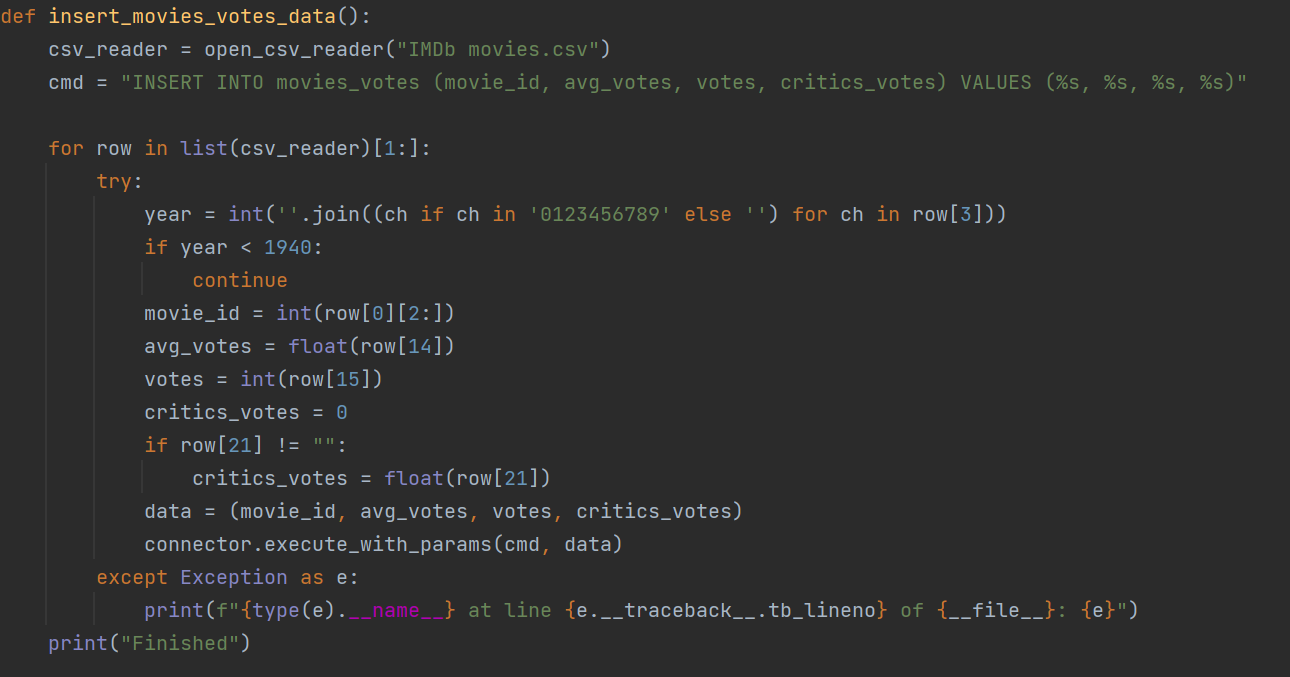


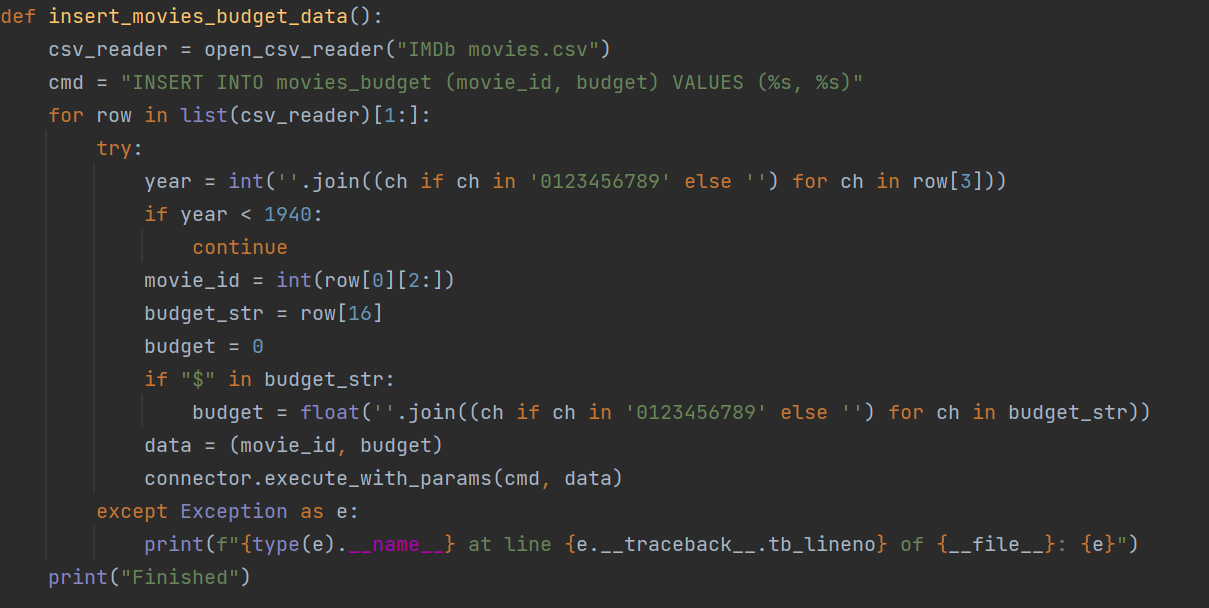


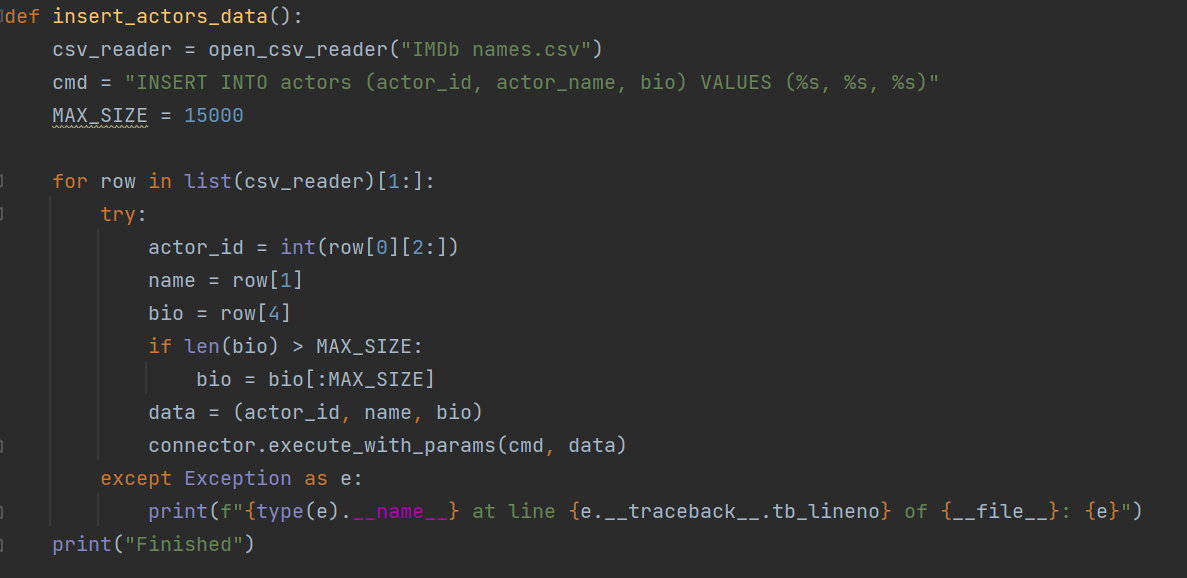


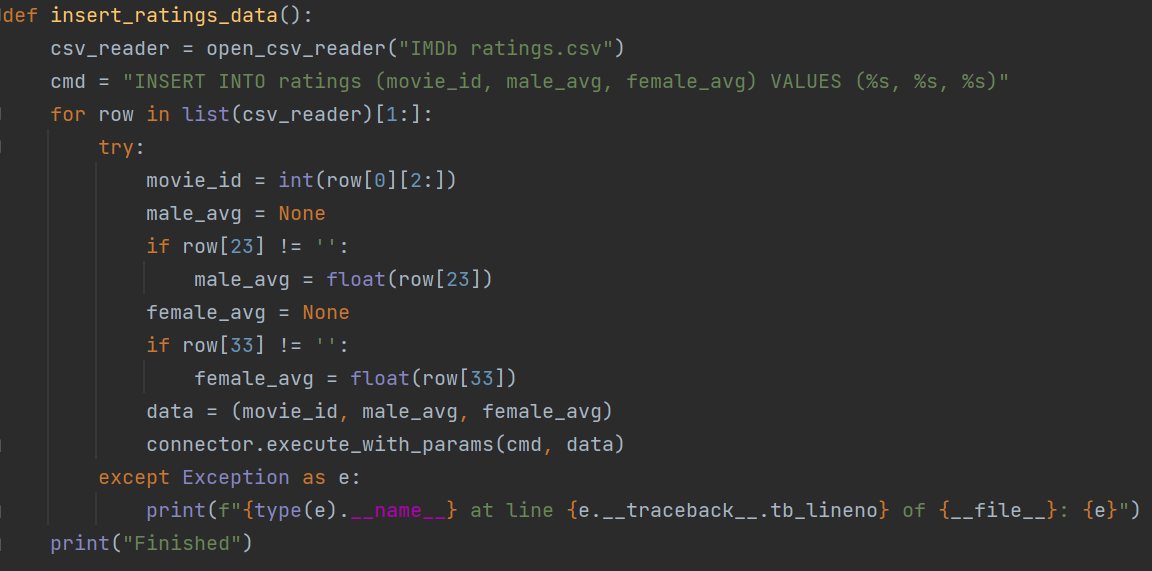












“Server”

We didn’t implement a real server in this project, but if we had our query would run with those commands from the back end application and the input we would get from the user (the inputs are explained in the second documentation file).

