CS 341 : Programming Languages
Spring 2022

Project 02 : MovieLens application

(doc v1.1)

Assignment: Python program to read/update MovieLens database in SQLite
Evaluation: Gradescope followed by manual execution & review
Policy: Individual work only
Complete By: Part 01 (data tier, 10 pts): Friday Feb 18th @ 11:59pm
Part 02 (object tier, 60 pts): Tuesday Feb 22nd @ 11:59pm
Part 03 (presentation tier, 30 pts): Friday Feb 25th @ 11:59pm

Early submission: there are no early submission bonuses available for this project
Late submissions: 10% penalty for any part that is submitted up to 24 hours late
20% penalty for any part that is submitted up to 48 hours late
48 hours after due date, a part cannot be submitted for credit

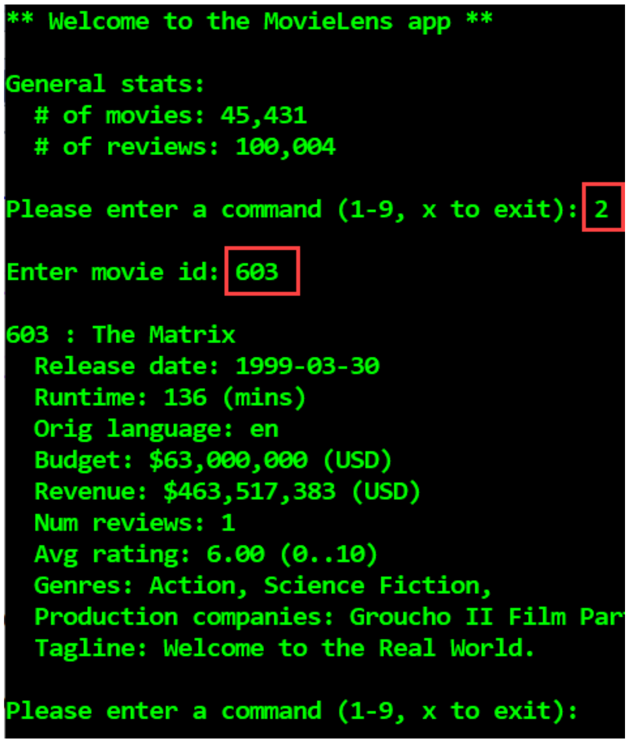
Pre-requisites: Lectures from Feb 02 and Feb 04; Homework #03

Overview

In project 02 we're going to build another database application, this time using an **N-tier design** as discussed in class and practiced in HW 03. We'll also switch databases, and work with a movies database called **MovieLens**.

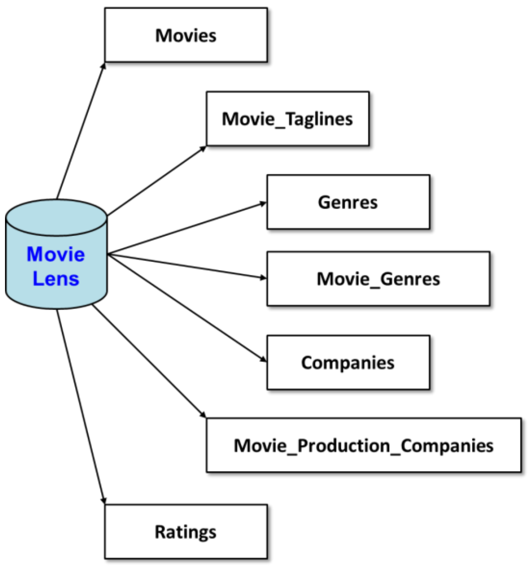
This project is in three parts, and you must complete all three parts before each deadline to earn full credit. Part 01 is to build and test the data access tier, and is worth 10/100 points. Part 02 is to build the object mapping tier, and is worth 60/100 points. Part 03 is to build the presentation tier, and is worth 30/100 points. Each part will be collected via Gradescope. There is no plotting in project 02.

You can work in replit.com, see the team project "**MovieLens + Python**". There is also a team project available for testing your SQL queries: "**MovieLens database**". If you prefer to work in a different environment, you can access the python files in this Dropbox folder, and the MovieLens database in this folder. For testing, note that movie 604 has no reviews, movie 595 has no tagline, movies 1777 and 2764 have no genres, and movies 92 and 206 have no production companies.


```
** Welcome to the MovieLens app **

General stats:
  # of movies: 45,431
  # of reviews: 100,004

Please enter a command (1-9, x to exit): 2

Enter movie id: 603

603 : The Matrix
  Release date: 1999-03-30
  Runtime: 136 (mins)
  Orig language: en
  Budget: $63,000,000 (USD)
  Revenue: $463,517,383 (USD)
  Num reviews: 1
  Avg rating: 6.00 (0..10)
  Genres: Action, Science Fiction,
  Production companies: Groucho II Film Par
  Tagline: Welcome to the Real World.

Please enter a command (1-9, x to exit):
```

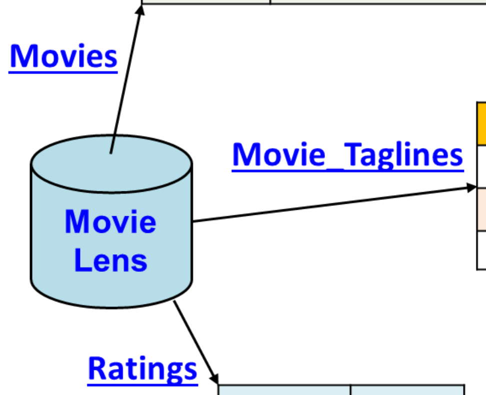
MovieLens database

The **MovieLens** database consists of data for over 45,000 movies, with approximately 100,000 reviews. Most popular movies are in the database, though the data only includes movies released before August 2017. [*The original database has over 26 million reviews, and is a good test of SQL efficiency; it's too large for replit.com, but you can download to try on other platforms.*]



The database consists of seven tables as shown here ------------> The database schema is available on replit.com: open the team project "MovieLens database", and run the **.schema** command. The schema will give you more information about the design of the database, the primary and foreign keys, and the column types.

Three tables are shown below: **Movies**, **Movie_Taglines**, and **Ratings**. The Movies table contains all the movies, with the Movie_ID as a primary key. There are roughly 45,000 movies. A "tagline" is a summary of the movie, what you might see on a movie poster; most movies have taglines, but not all. All movie reviews are stored in the Ratings table; in this database a movie review is a rating from 0..10, inclusive. A movie can have 0 or more reviews.



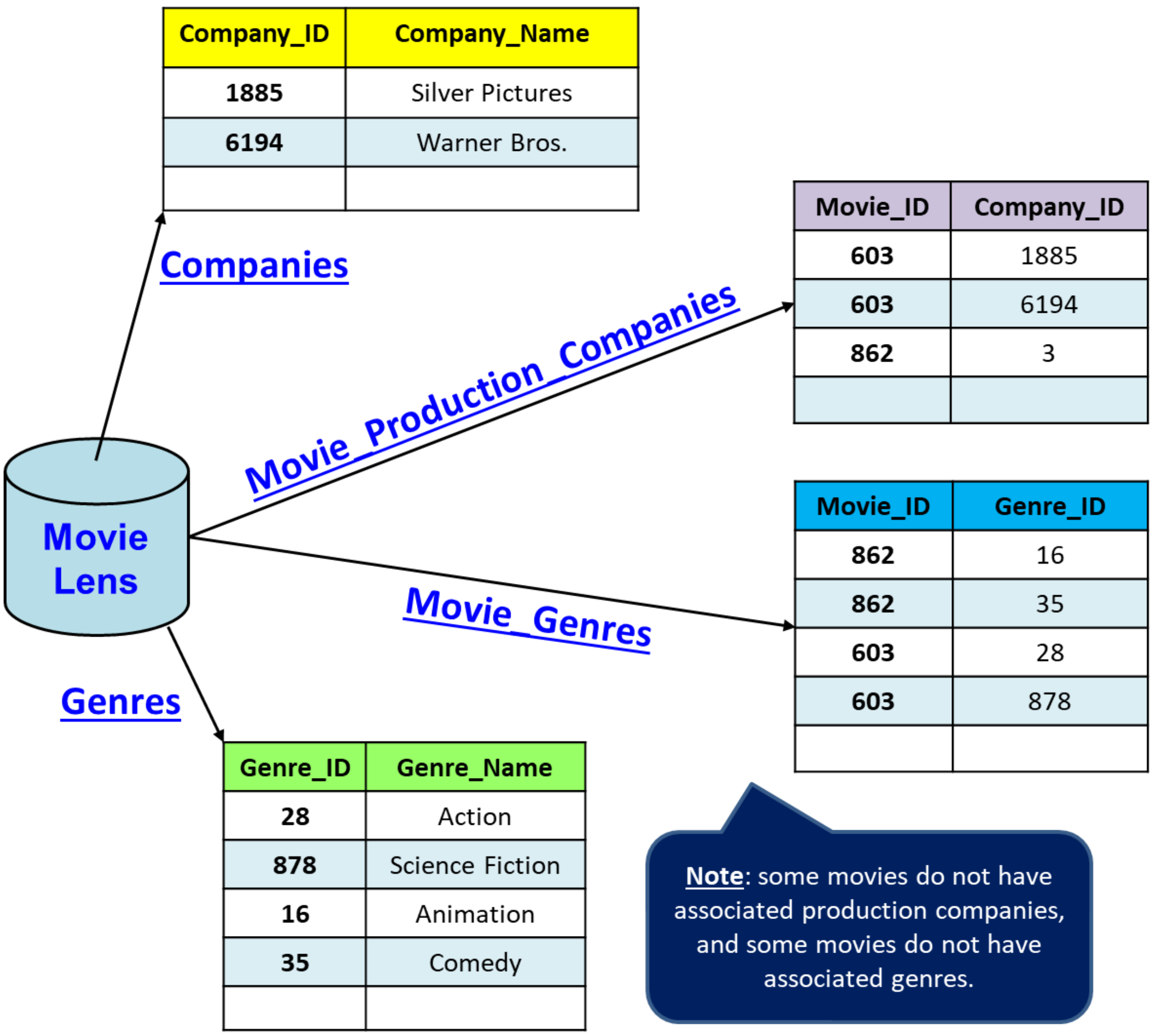
Movie_ID	Title	Release_Date	Runtime	Original_Language	Budget	Revenue
603	**The Matrix**	1999-03-30 00:00:00.000	136	en	63000000	463517383
862	**Toy Story**	1995-10-30 00:00:00.000	81	en	30000000	373554033

Movie_ID	Tagline
603	Welcome to the Real World.
605	Everything that has a beginning has an end.

Movie_ID	Rating
605	8
603	6
605	10
605	6

Note: some movies do not have a tagline, and some movies do not have reviews. Example: movie 862 has no tagline and no reviews.

The database contains four more tables: **Companies**, **Movie_Production_Companies**, **Genres**, and **Movie_Genres**. These tables associate a movie with its production companies (e.g. *Disney*), and its genres (e.g. *Comedy*). Since a movie can have 1 or more production companies (and fall into 1 or more genres), the table Movie_Production_Companies lists the companies associated with a movie, and Movie_Genres lists the genres that a movie falls into. Much like taglines and reviews, not all movies have associated production companies or genres.



Company_ID	Company_Name
1885	Silver Pictures
6194	Warner Bros.

Movie_ID	Company_ID
603	1885
603	6194
862	3

Movie_ID	Genre_ID
862	16
862	35
603	28
603	878

Genre_ID	Genre_Name
28	Action
878	Science Fiction
16	Animation
35	Comedy

Note: some movies do not have associated production companies, and some movies do not have associated genres.

Null values are not allowed in the database; every row and column contains a value, although some values may be the empty string "" (e.g. a tagline could be ""). For column types, see the database schema on replit.com.

Part 01: data access tier

In the Python file "datatier.py", complete the following 3 functions as defined by the header comments. This represents the data access tier as discussed in class, and used in HW #03; feel free to reuse the code from HW #03. Develop using replit.com and the team project "MovieLens and Python", or in a separate Python environment. Submit to Gradescope under the assignment "Project 02-01: Data Tier". You have unlimited submissions. If you are working outside replit.com, the initial "datatier.py" file can be found here.

You must implement the functions as defined below. Do not change function names, do not change the parameters / return values, and do not add functions. No global variables, no other imports.

```
#
# datatier.py
#
# Executes SQL queries against the given database.
#
# Original author:
#   Prof. Joe Hummel
#   U. of Illinois, Chicago
#   CS 341, Spring 2022
#   Project 02
#
import sqlite3


##################################################################
#
# select_one_row:
#
# Given a database connection and a SQL Select query,
# executes this query against the database and returns
# the first row retrieved by the query (or the empty
# tuple () if no data was retrieved). The query can
# be parameterized, in which case pass the values as
# a list via parameters; this parameter is optional.
#
# Returns: first row retrieved by the given query, or
#          () if no data was retrieved. If an error
#          occurs, a msg is output and None is returned.
#
# NOTE: error message format is
#   print("select_one_row failed:", err)
# where err is the Exception object.
#
def select_one_row(dbConn, sql, parameters = []):
   pass


##################################################################
#
# select_n_rows:
#
# Given a database connection and a SQL Select query,
# executes this query against the database and returns
# a list of rows retrieved by the query. If the query
# retrieves no data, the empty list [] is returned.
# The query can be parameterized, in which case pass
# the values as a list via parameters; this parameter
# is optional.
#
# Returns: a list of 0 or more rows retrieved by the
#          given query; if an error occurs a msg is
#          output and None is returned.
#
# NOTE: error message format is
#   print("select_n_rows failed:", err)
# where err is the Exception object.
#
def select_n_rows(dbConn, sql, parameters = []):
   pass


##################################################################
#
# perform_action:
#
# Given a database connection and a SQL action query,
# executes this query and returns the # of rows
# modified; a return value of 0 means no rows were
# updated. Action queries are typically "insert",
# "update", "delete". The query can be parameterized,
# in which case pass the values as a list via
# parameters; this parameter is optional.
#
# Returns: the # of rows modified by the query; if an
#          error occurs a msg is output and -1 is
#          returned. Note that a return value of 0 is
#          not considered an error --- it means the
#          query did not change the database (e.g.
#          because the where condition was false?).
#
# NOTE: error message format is
#   print("perform_action failed:", err)
# where err is the Exception object.
#
def perform_action(dbConn, sql, parameters = []):
   pass
```

Part 02: object mapping tier

In the Python file “objecttier.py”, implement the following classes and functions as defined by the header comments. This represents the object mapping tier as discussed in class, and implemented in HW #03. Develop using replit.com and the team project “MovieLens and Python”, or in a separate Python environment. Submit to Gradescope under the assignment “Project 02-02: Object Tier”. You have unlimited submissions. If you are working outside replit.com, the initial “objecttier.py” file can be found here. [*Hint: consider using inheritance to implement some of the classes.*]

You must implement the classes and functions as defined below, and you cannot use tools to generate the code for you (e.g. no use of SQLAlchemy). Do not change class / property / function names, do not change the parameters / return values, and do not add functions. No global variables, no other imports. All computations must be computed using SQL, including the average ratings; you cannot simply read in the data and do the searching/sorting/computing yourself. Keep in mind that all SQL must be executed by the functions provided in the data access tier, you cannot execute SQL here in the object tier. We will be testing your object tier against our implementation of the data access tier, as defined in part 01.

```
#
# objecttier
#
# Builds Movie-related objects from data retrieved through
# the data tier.
#
# Original author:
#   Prof. Joe Hummel
#   U. of Illinois, Chicago
#   CS 341, Spring 2022
#   Project #02
#
import datatier


##################################################################
#
# Movie:
#
# Constructor(...)
# Properties:
#   Movie_ID: int
#   Title: string
#   Release_Year: string
#
class Movie:
   pass


##################################################################
#
# MovieRating:
#
# Constructor(...)
# Properties:
#   Movie_ID: int
#   Title: string
#   Release_Year: string
#   Num_Reviews: int
#   Avg_Rating: float
#
class MovieRating:
   pass


##################################################################
#
# MovieDetails:
#
# Constructor(...)
# Properties:
#   Movie_ID: int
#   Title: string
#   Release_Date: string, date only (no time)
#   Runtime: int (minutes)
#   Original_Language: string
#   Budget: int (USD)
#   Revenue: int (USD)
#   Num_Reviews: int
#   Avg_Rating: float
#   Tagline: string
#   Genres: list of string
#   Production_Companies: list of string
#
class MovieDetails:
   pass

##################################################################
#
# num_movies:
#
# Returns: # of movies in the database; if an error returns -1
#
def num_movies(dbConn):
   pass

##################################################################
#
# num_reviews:
#
# Returns: # of reviews in the database; if an error returns -1
#
def num_reviews(dbConn):
   pass

##################################################################
#
# get_movies:
#
# gets and returns all movies whose name are "like"
# the pattern. Patterns are based on SQL, which allow
# the _ and % wildcards. Pass "%" to get all stations.
#
# Returns: list of movies in ascending order by name;
#          an empty list means the query did not retrieve
#          any data (or an internal error occurred, in
#          which case an error msg is already output).
#
def get_movies(dbConn, pattern):
   pass

##################################################################
#
# get_movie_details:
#
# gets and returns details about the given movie; you pass
# the movie id, function returns a MovieDetails object. Returns
# None if no movie was found with this id.
#
# Returns: if the search was successful, a MovieDetails obj
#          is returned. If the search did not find a matching
#          movie, None is returned; note that None is also
#          returned if an internal error occurred (in which
#          case an error msg is already output).
#
def get_movie_details(dbConn, movie_id):
   pass

##################################################################
#
# get_top_N_movies:
#
# gets and returns the top N movies based on their average
# rating, where each movie has at least the specified # of
# reviews. Example: pass (10, 100) to get the top 10 movies
# with at least 100 reviews.
#
# Returns: returns a list of 0 or more MovieRating objects;
#          the list could be empty if the min # of reviews
#          is too high. An empty list is also returned if
#          an internal error occurs (in which case an error
#          msg is already output).
#
def get_top_N_movies(dbConn, N, min_num_reviews):
   pass

##################################################################
#
# add_review:
#
# Inserts the given review --- a rating value 0..10 --- into
# the database for the given movie. It is considered an error
# if the movie does not exist (see below), and the review is
# not inserted.
#
# Returns: 1 if the review was successfully added, returns
#          0 if not (e.g. if the movie does not exist, or if
#          an internal error occurred).
#
def add_review(dbConn, movie_id, rating):
   pass

##################################################################
#
# set_tagline:
#
# Sets the tagline --- summary --- for the given movie. If
# the movie already has a tagline, it will be replaced by
# this new value. Passing a tagline of "" effectively
# deletes the existing tagline. It is considered an error
# if the movie does not exist (see below), and the tagline
# is not set.
#
# Returns: 1 if the tagline was successfully set, returns
#          0 if not (e.g. if the movie does not exist, or if
#          an internal error occurred).
#
def set_tagline(dbConn, movie_id, tagline):
   pass
```

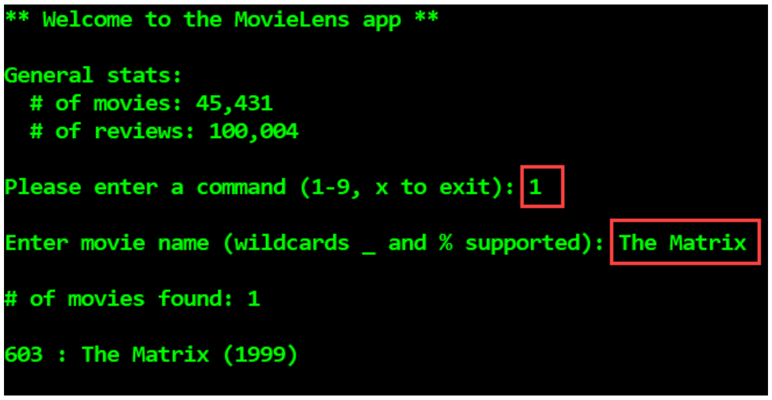
Part 03: presentation tier

In the Python file "main.py", write a Python program to implement 5 commands: (1) lookup movies by name/pattern, (2) lookup details about a specific movie, (3) top N movies by average rating, (4) insert a review, and (5) set a movie's tagline. Each of these commands should be a separate function in your Python program. This represents the presentation tier as discussed in class, and in HW #03. Develop using replit.com and the team project "MovieLens and Python", or in a separate Python environment. Submit to Gradescope under the assignment "Project 02-03: Presentation Tier". You have unlimited submissions.

The presentation tier cannot execute SQL directly, all data must be retrieved by calling the object mapping tier as defined in part 02. No global variables, no imports other than **sqlite3** and **objecttier**. We will be testing your file against our implementation of the object mapping tier, so do not extend / change the object tier as you develop the presentation tier. [***Note***: *screenshots say enter command 1-9, it should be 1-5.*]

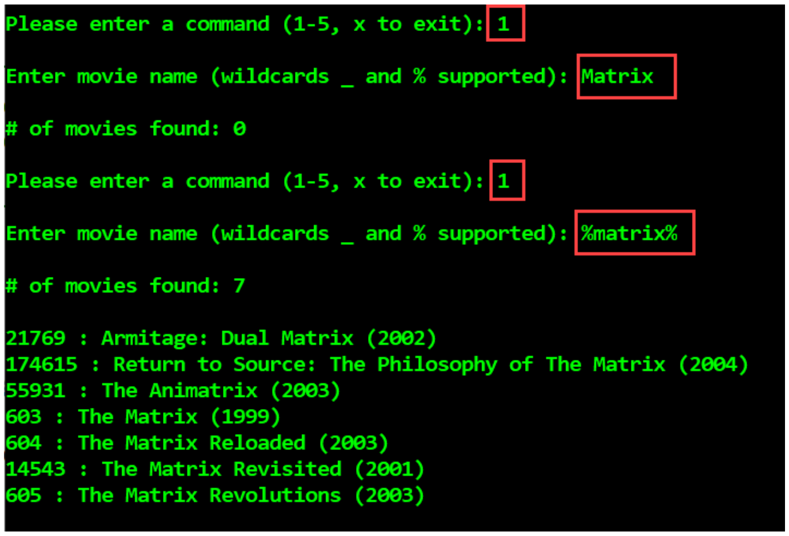
Command "1"

Inputs a movie name and outputs the movie's ID, title, and year of release. Here's an example of the program starting up, and searching for "The Matrix":



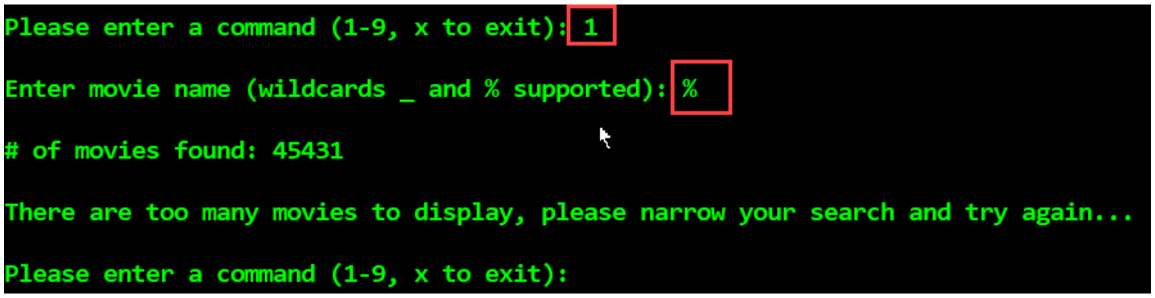
```
** Welcome to the MovieLens app **

General stats:
  # of movies: 45,431
  # of reviews: 100,004

Please enter a command (1-9, x to exit): 1

Enter movie name (wildcards _ and % supported): The Matrix

# of movies found: 1

603 : The Matrix (1999)
```

The user is allowed to enter patterns using the SQL wildcards _ and %. For example:



```
Please enter a command (1-5, x to exit): 1

Enter movie name (wildcards _ and % supported): Matrix

# of movies found: 0

Please enter a command (1-5, x to exit): 1

Enter movie name (wildcards _ and % supported): %matrix%

# of movies found: 7

21769 : Armitage: Dual Matrix (2002)
174615 : Return to Source: The Philosophy of The Matrix (2004)
55931 : The Animatrix (2003)
603 : The Matrix (1999)
604 : The Matrix Reloaded (2003)
14543 : The Matrix Revisited (2001)
605 : The Matrix Revolutions (2003)
```

If the user enters a pattern and more than 100 movies are found, do not display the movies. Instead, inform the user and ask them to narrow their search:

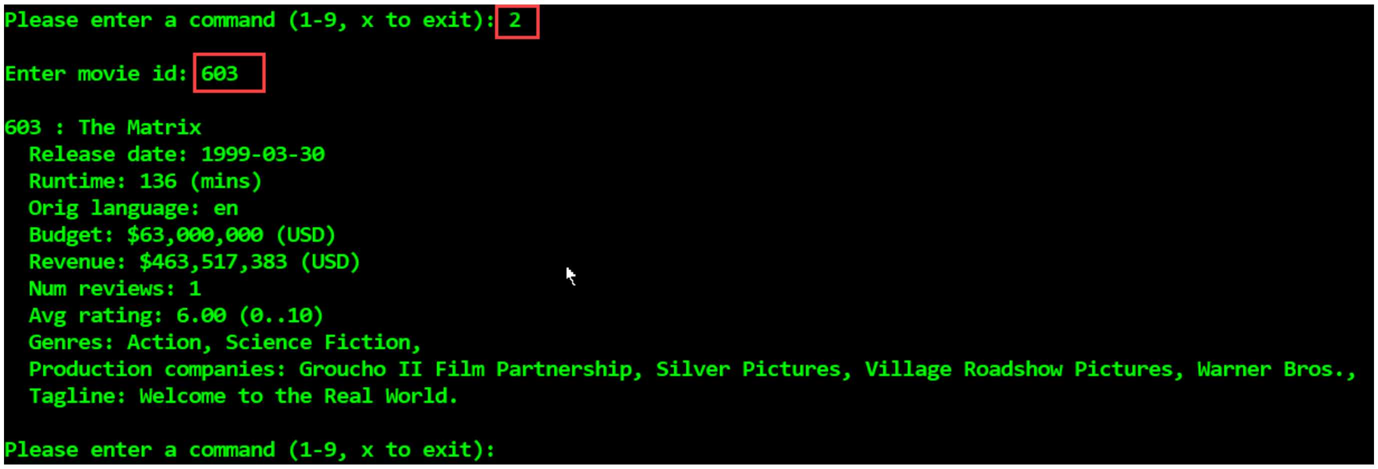


```
Please enter a command (1-9, x to exit): 1

Enter movie name (wildcards _ and % supported): %

# of movies found: 45431

There are too many movies to display, please narrow your search and try again...

Please enter a command (1-9, x to exit):
```

The presentation tier should be calling the object mapping tier function **get_movies( )** to retrieve a list of **Movie** objects, which are then output to the screen. If the object mapping tier returns None (e.g. denoting an error), the presentation tier should behave as if no movies were found.

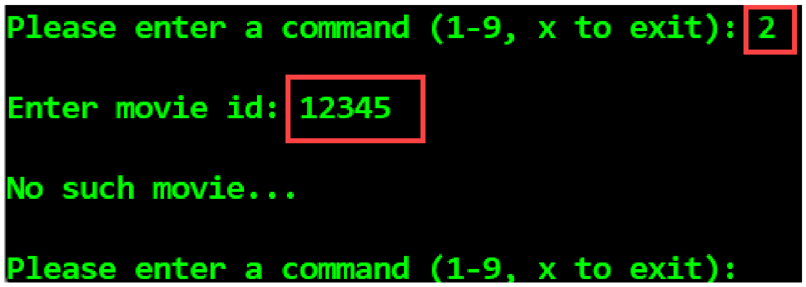
Command “2”

Inputs a movie id and outputs detailed movie information about this movie --- tagline, budget, revenue, genres, etc. Example:



```
Please enter a command (1-9, x to exit): 2

Enter movie id: 603

603 : The Matrix
  Release date: 1999-03-30
  Runtime: 136 (mins)
  Orig language: en
  Budget: $63,000,000 (USD)
  Revenue: $463,517,383 (USD)
  Num reviews: 1
  Avg rating: 6.00 (0..10)
  Genres: Action, Science Fiction,
  Production companies: Groucho II Film Partnership, Silver Pictures, Village Roadshow Pictures, Warner Bros.,
  Tagline: Welcome to the Real World.

Please enter a command (1-9, x to exit):
```

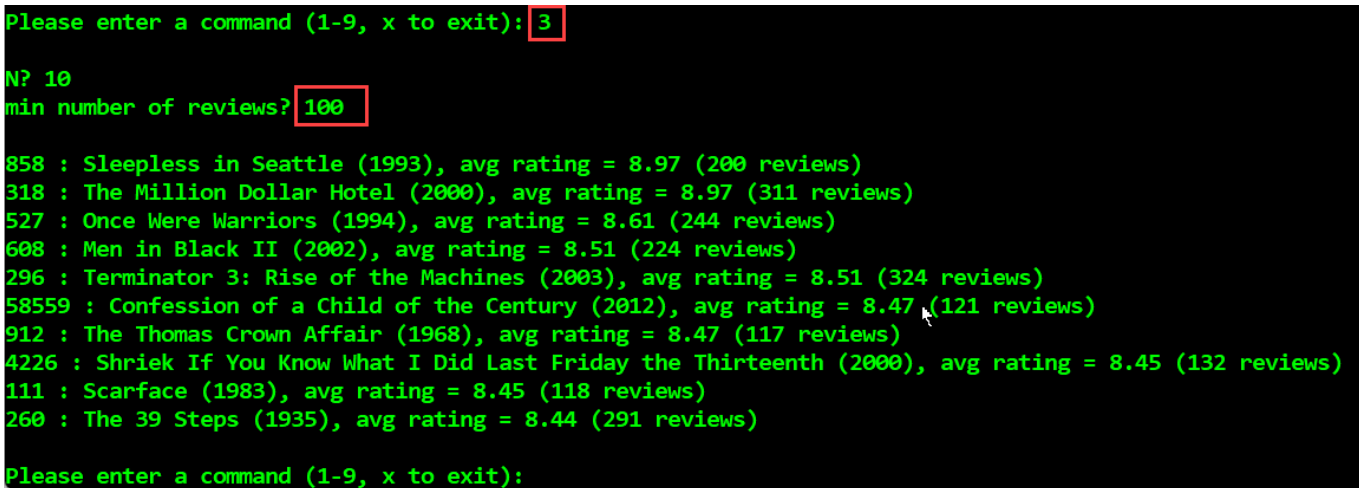
The presentation tier should call the **get_movie_details()** function in the object mapping tier, which returns a **MovieDetails** object. If the movie is not found (or None is returned), say so:



```
Please enter a command (1-9, x to exit): 2

Enter movie id: 12345

No such movie...

Please enter a command (1-9, x to exit):
```

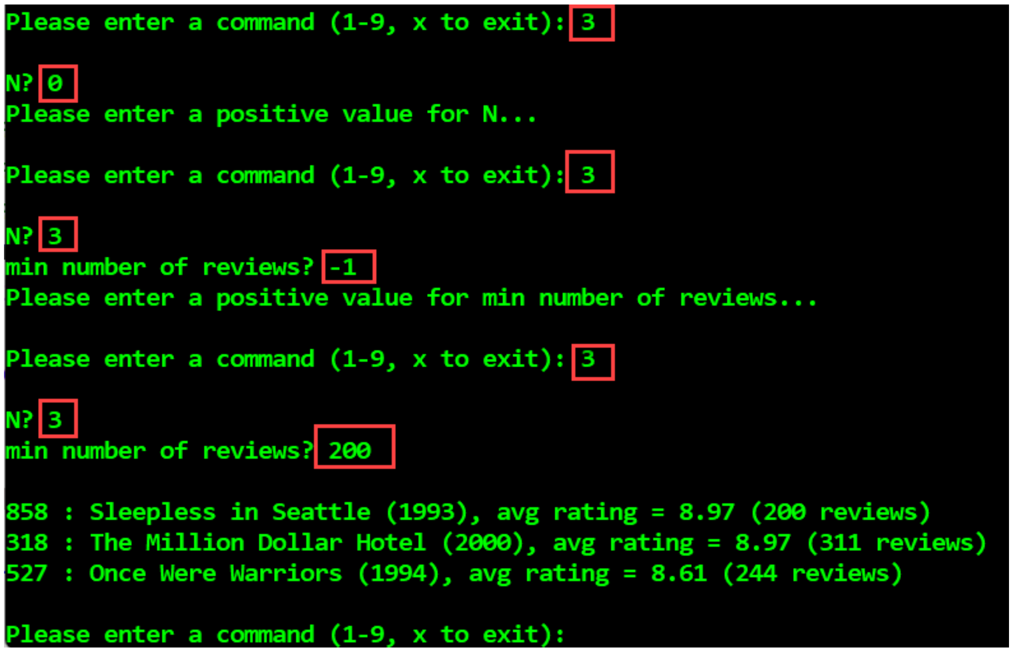
Command "3"

Output the top N movies based on their average rating *and* with a minimum number of reviews. For example, here are the top 10 movies with at least 100 reviews:



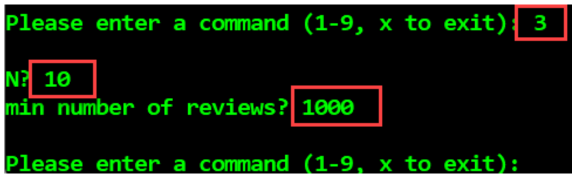
```
Please enter a command (1-9, x to exit): 3

N? 10
min number of reviews? 100

858 : Sleepless in Seattle (1993), avg rating = 8.97 (200 reviews)
318 : The Million Dollar Hotel (2000), avg rating = 8.97 (311 reviews)
527 : Once Were Warriors (1994), avg rating = 8.61 (244 reviews)
608 : Men in Black II (2002), avg rating = 8.51 (224 reviews)
296 : Terminator 3: Rise of the Machines (2003), avg rating = 8.51 (324 reviews)
58559 : Confession of a Child of the Century (2012), avg rating = 8.47 (121 reviews)
912 : The Thomas Crown Affair (1968), avg rating = 8.47 (117 reviews)
4226 : Shriek If You Know What I Did Last Friday the Thirteenth (2000), avg rating = 8.45 (132 reviews)
111 : Scarface (1983), avg rating = 8.45 (118 reviews)
260 : The 39 Steps (1935), avg rating = 8.44 (291 reviews)

Please enter a command (1-9, x to exit):
```

The presentation tier should call the **get_top_N_movies()** function in the object mapping tier, which returns 0 or more **MovieRating** objects in a list. Be sure to validate the input values, as shown below:



```
Please enter a command (1-9, x to exit): 3

N? 0
Please enter a positive value for N...

Please enter a command (1-9, x to exit): 3

N? 3
min number of reviews? -1
Please enter a positive value for min number of reviews...

Please enter a command (1-9, x to exit): 3

N? 3
min number of reviews? 200

858 : Sleepless in Seattle (1993), avg rating = 8.97 (200 reviews)
318 : The Million Dollar Hotel (2000), avg rating = 8.97 (311 reviews)
527 : Once Were Warriors (1994), avg rating = 8.61 (244 reviews)

Please enter a command (1-9, x to exit):
```

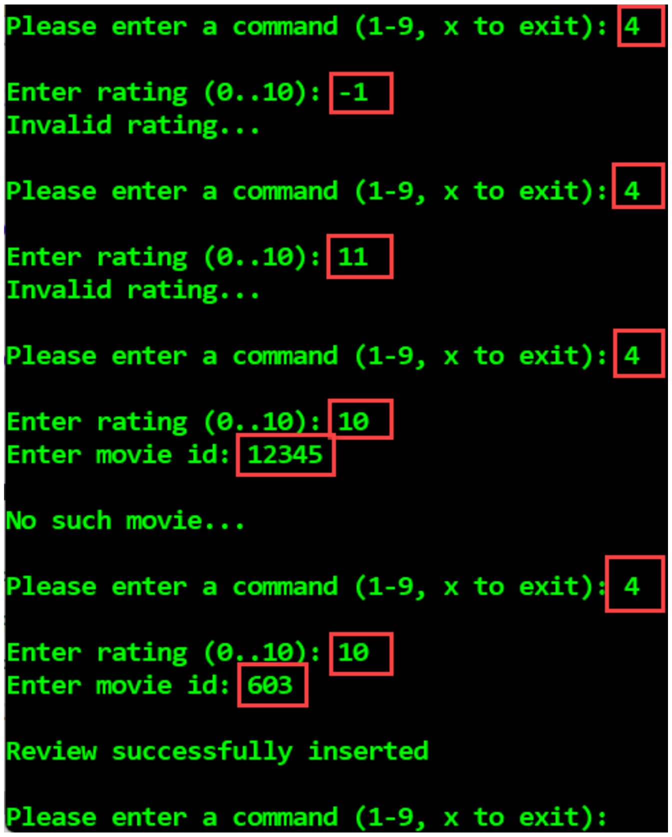
If the object mapping tier returns None or an empty list, there is no output.



```
Please enter a command (1-9, x to exit): 3

N? 10
min number of reviews? 1000

Please enter a command (1-9, x to exit):
```

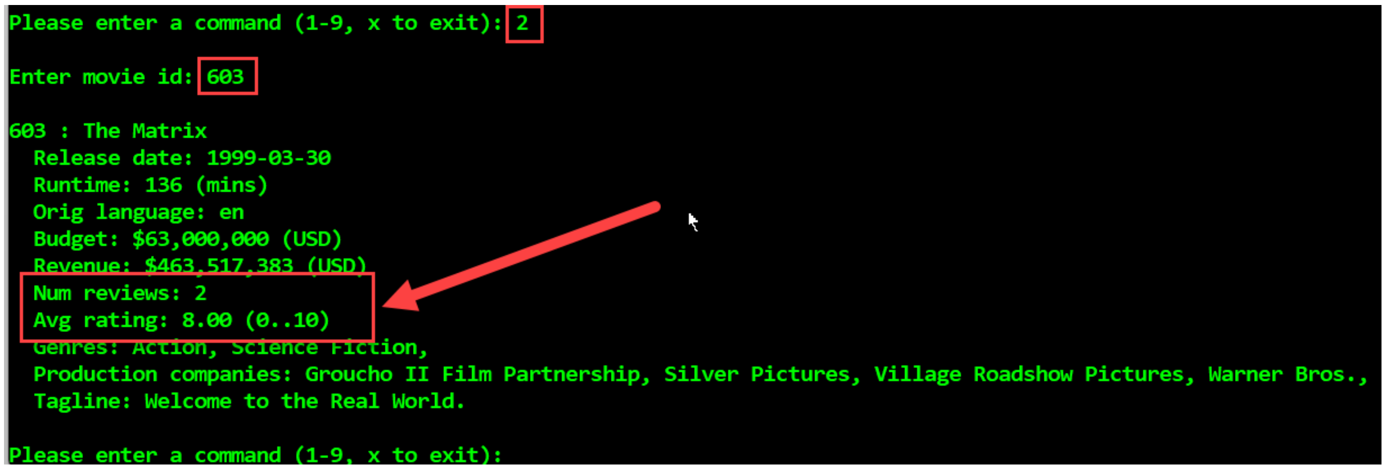
Command “4”

Inserts a new review into the database. The program inputs a movie id and a rating (0..10), and then inserts the review after validating the input:



```
Please enter a command (1-9, x to exit): 4

Enter rating (0..10): -1
Invalid rating...

Please enter a command (1-9, x to exit): 4

Enter rating (0..10): 11
Invalid rating...

Please enter a command (1-9, x to exit): 4

Enter rating (0..10): 10
Enter movie id: 12345

No such movie...

Please enter a command (1-9, x to exit): 4

Enter rating (0..10): 10
Enter movie id: 603

Review successfully inserted

Please enter a command (1-9, x to exit):
```

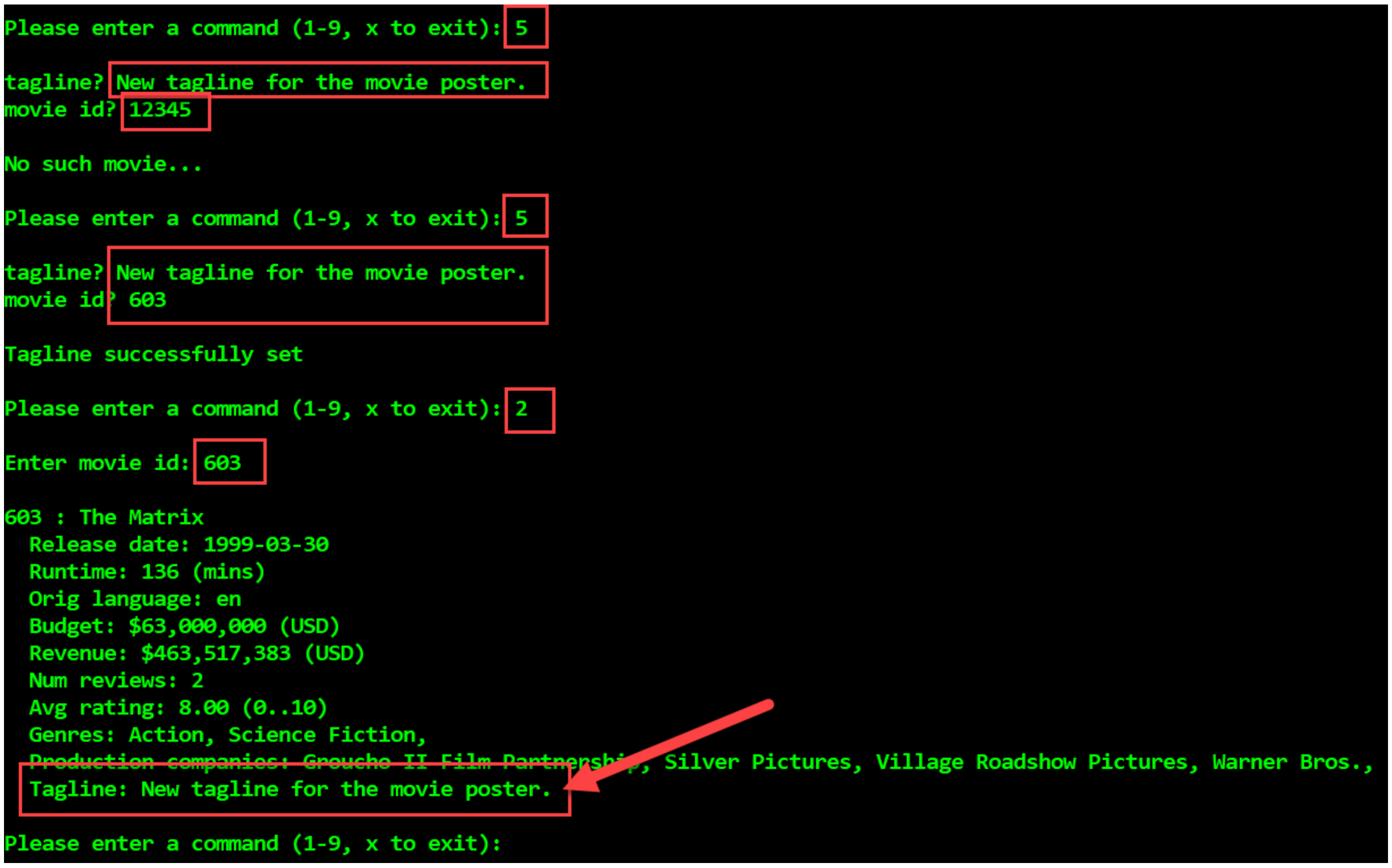
To confirm the review was added, get details about the movie (and compare to the earlier screenshot):



```
Please enter a command (1-9, x to exit): 2

Enter movie id: 603

603 : The Matrix
  Release date: 1999-03-30
  Runtime: 136 (mins)
  Orig language: en
  Budget: $63,000,000 (USD)
  Revenue: $463,517,383 (USD)
  Num reviews: 2
  Avg rating: 8.00 (0..10)
  Genres: Action, Science Fiction,
  Production companies: Groucho II Film Partnership, Silver Pictures, Village Roadshow Pictures, Warner Bros.,
  Tagline: Welcome to the Real World.

Please enter a command (1-9, x to exit):
```

Command “5”

Sets the tagline for a given movie, either by inserting (if not already there) or updating (if already there).



```
Please enter a command (1-9, x to exit): 5

tagline? New tagline for the movie poster.
movie id? 12345

No such movie...

Please enter a command (1-9, x to exit): 5

tagline? New tagline for the movie poster.
movie id? 603

Tagline successfully set

Please enter a command (1-9, x to exit): 2

Enter movie id: 603

603 : The Matrix
  Release date: 1999-03-30
  Runtime: 136 (mins)
  Orig language: en
  Budget: $63,000,000 (USD)
  Revenue: $463,517,383 (USD)
  Num reviews: 2
  Avg rating: 8.00 (0..10)
  Genres: Action, Science Fiction,
  Production companies: Groucho II Film Partnership, Silver Pictures, Village Roadshow Pictures, Warner Bros.,
  Tagline: New tagline for the movie poster.

Please enter a command (1-9, x to exit):
```

Academic Honesty

In this assignment, all work submitted for grading *must* be done individually. While we encourage you to talk to your peers and learn from them, this interaction must be superficial with regards to all work submitted for grading. This means you *cannot* work in teams, you cannot work side-by-side, you cannot submit someone else’s work (partial or complete) as your own. The University’s policy is available here:

https://dos.uic.edu/conductforstudents.shtml

In particular, note that you are guilty of academic dishonesty if you **extend or receive any kind of unauthorized assistance**. Absolutely no transfer of program code between students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums. Other examples of academic dishonesty include emailing your program to another student, screen sharing, copying-pasting code from the internet, working in a group on a homework assignment, and allowing a tutor, TA, or another individual to write an answer for you. Academic dishonesty is unacceptable, and penalties range from a letter grade drop to expulsion from the university; cases are handled via the official student conduct process described at https://dos.uic.edu/conductforstudents.shtml .