**Main.py:**

**register():**

* Registers a new user by prompting for user details (nickname, first name, last name, password).
* Utilizes the Users class methods to check if the nickname is already in use and then registers the user using the register method.
* Maintains a global user\_id variable to track the currently logged-in user.

**login():**

* Allows an existing user to log in by taking input for nickname and password.
* Utilizes the Users class methods to verify the user's existence and login credentials.
* Updates the global user\_id variable upon successful login.

**movlst():**

* Invokes the movlst method of the Movies class, displaying a list of movies by interacting with the database.

**movdt():**

* Prompts the user to input a movie title and invokes the movdt method of the Movies class to display detailed information about the specified movie.

**movsrch():**

* Takes input for the type of search (title, director, or genre) and performs a movie search by calling the respective methods (movsrch\_title, movsrch\_director, movsrch\_genre) of the Movies class.

**movlike():**

* Loops to allow the user to continuously input liked movies or request their liked movies list.
* Invokes the movlike method of the Movies class to handle liked movies and updates the database accordingly.

**movfv():**

* Similar to movlike(), allows the user to add favorite movies or view their favorite movies list.
* Invokes the movfv method of the Movies class and updates the database.

**movadd():**

Adds a new movie to the database by invoking the movadd method of the Movies class and taking user input for movie details.

**movcat():**

* Takes input for the type of category and invokes corresponding methods (most\_liked, genres, newest) of the Movies class to display movies based on the chosen category.

**Main Loop:**

* Initially prompts the user to input either 'login' or 'register'.
* Continues looping until a valid input is received.
* Executes the corresponding commands based on the user's input, interacting with the defined functions and class methods.
* The loop continues until the user inputs 'end', which results in program termination.

**Users.py:**

Users Class Overview:

* The Users class is designed to manage user-related operations.
* It provides functions for user registration, login, and user information retrieval.

**\_\_init\_\_(self, nickname="", first\_name="", last\_name="", password="")**

* Constructor method for the Users class.
* Initializes a Users object with optional parameters for user details.

**create\_table(self, conn, cursor)**

* Creates the 'users' table in the database if it does not exist.
* The table structure includes columns for user ID (auto-incremented primary key), first name, last name, nickname, and the hashed password.

**get\_user\_id(self, cursor, nickname)**

* Retrieves the user ID based on the provided nickname from the 'users' table.
* Returns the user ID if the user is found; otherwise, returns None.

**register(self, conn, cursor, nickname, first\_name, last\_name, password)**

* Registers a new user in the 'users' table by inserting their details, including first name, last name, nickname, and hashed password.

**login(self, cursor, nickname, password)**

* Validates user login credentials by checking if the provided nickname exists in the 'users' table and verifying the hashed password.
* Prints "Login successful!" if the credentials are valid; otherwise, prints "Incorrect password. Try again." or "User not found. You can register with the command 'register'."

**check\_password(self, password, stored\_hash\_password)**

* Compares the hashed password stored in the 'users' table with the hashed password provided during login.
* Returns True if the passwords match; otherwise, returns False.

**check\_if\_in\_userlist(self, cursor, nickname)**

* Checks if a user with the specified nickname already exists in the 'users' table.
* Returns True if the user exists; otherwise, returns False.

**user\_list(self, cursor)**

* Retrieves and prints the details of all users in the 'users' table.

**Movies.py:**

Movies class Overview:

* This class is designed to handle movie-related operations. It includes functions for adding movies, retrieving movie details, searching for movies, and fetching movies based on different criteria.

**\_\_init\_\_(self, title="", desc="", date="", director="", genre="")**

- Constructor for the Movies class.

- Initializes the movie attributes such as title, description, release date, director, and genre.

**add\_movie(self, conn, cursor)**

- Adds a movie to the database using the provided conn (connection) and cursor.

- Executes an SQL INSERT query with the movie details.

**get\_movie\_id(self, cursor, title)**

- Retrieves the movie ID from the database based on the provided title.

- Executes an SQL SELECT query to fetch the movie ID.

**check\_if\_in\_movlist(self, cursor, title)**

- Checks if a movie with the given title exists in the database.

- Executes an SQL SELECT query and returns a boolean indicating whether the movie is in the database.

**movlst(self, cursor)**

- Retrieves and prints all movies from the database.

- Executes an SQL SELECT query to fetch all movies and prints the results.

**movdt(self, cursor, given\_movie)**

- Retrieves and prints details for a specific movie (given\_movie) from the database.

- Executes an SQL SELECT query and checks if the movie exists using check\_if\_in\_movlist.

**movsrch\_title(self, cursor, search\_input)**

- Searches for movies in the database based on a title-like search.

- Executes an SQL SELECT query and applies Levenshtein distance and partial ratio matching to filter search results.

**movsrch\_director(self, cursor, search\_input)**

- Searches for movies in the database based on a director-like search.

- Executes an SQL SELECT query and applies Levenshtein distance and partial ratio matching to filter search results.

**movsrch\_genre(self, cursor, search\_input)**

- Searches for movies in the database based on a genre-like search.

- Executes an SQL SELECT query and applies Levenshtein distance and partial ratio matching to filter search results.

**movfv(self, cursor, user\_id)**

- Fetches and prints movies marked as favorites by a specific user.

- Executes a JOIN query on the movies and user\_favorites tables based on the provided user\_id.

**movlike(self, cursor, user\_id)**

- Fetches and prints movies liked by a specific user.

- Executes a JOIN query on the movies and user\_likes tables based on the provided user\_id.

**movadd(self, conn, cursor, title, desc, date, director, genre)**

- Adds a new movie to the database with the provided details.

- Executes an SQL INSERT query with the new movie's information.

**genres(self, cursor, given\_genre)**

- Fetches and prints top-rated movies of a given genre.

- Executes an SQL SELECT query based on the provided genre, ordering by like\_count.

**most\_liked(self, cursor)**

- Fetches and prints the top 5 most liked movies from the database.

- Executes an SQL SELECT query, ordering by like\_count.

**newest(self, cursor)**

- Fetches and prints the top 5 newest movies from the database.

- Executes an SQL SELECT query, ordering by release\_date.

**Database:**

**movies:**

The Movies database table stores information about various movies, including details such as title, description, release date, director, genre, like count, and favorite count. Below is a breakdown of the table structure:

**MovieID** (Primary Key):

**Type:** Integer

**Description:** A unique identifier for each movie in the database.

**Title:**

**Type:** String

**Description:** The title of the movie, providing a distinct and recognizable name.

**Description:**

**Type:** String

**Description:** A brief overview or synopsis of the movie, highlighting its key themes or storyline.

**Release Date:**

**Type:** Date

**Description:** The date when the movie was officially released to the public.

**Director:**

**Type:** String

**Description:** The name of the director responsible for bringing the movie to life.

**Genre:**

**Type:** String

**Description:** The genre classification of the movie, categorizing it into broader thematic areas such as Drama, Action, Comedy, History, Fantasy, Crime, Animation, or Documentary.

**Like Count:**

**Type:** Integer

Description: The number of users who have liked the movie, indicating its popularity among the audience.

**Favorite Count:**

**Type:** Integer

**Description:** The count of users who have marked the movie as a favorite, showcasing a personal preference for the film.

The table captures essential information about each movie, enabling users to search, retrieve, and analyze data based on different criteria. It serves as the foundation for the Movies class methods in the program, allowing seamless interactions with the movie-related functionalities offered by the application.

**users:**

The Users database table contains information about registered users, encompassing details such as user ID, first name, last name, nickname, and hashed password. Below is a breakdown of the table structure:

**UserID** (Primary Key):

**Type:** Integer

**Description:** A unique identifier for each user in the database.

**FirstName:**

**Type:** String

**Description:** The first name of the user, representing their given name.

**LastName:**

**Type:** String

**Description:** The last name of the user, indicating their family name or surname.

**Nickname:**

**Type:** String

**Description:** The chosen nickname of the user, serving as a unique identifier for login purposes.

**Password** (Hashed):

**Type:** String

**Description:** The securely hashed representation of the user's password, providing data protection and privacy.

The table captures fundamental information about each registered user, ensuring a secure and organized storage of user data. The hashed password adds an extra layer of security, protecting user credentials from unauthorized access. This table serves as the foundation for the Users class methods in the program, enabling user registration, login, and retrieval of user information functionalities.

**user\_likes:**

* Storing data after every ‘like’ interaction.
* Links the user\_id with the movie\_id of the liked movie.

**user\_favorites:**

* Storing data after every ‘favorite’ interaction.
* Links the user\_id with the movie\_id of the favorited movie.