Movie Review App - Database Solutions

Complete Solutions for All Challenges

Challenge 2: Get Movie by ID (Easy)

Task: Find and return ONE specific movie from the database.

Solution:

```
def get_movie_by_id(movie_id):
    session = get_session()
    try:
        movie = session.query(Movie).filter(Movie.id == movie_id).first()

        # Check if movie is None
        if movie is None:
            return None

        # Convert to dictionary and return
        return movie.to_dict()

    finally:
        session.close()
```

Key Points:

- Use filter(Movie.id == movie_id) to find the specific movie
- Use .first() to get one result (or None if not found)
- Always check if the result is None before calling .to_dict()
- Always close the session in the finally block

Challenge 3: Add a Review (Medium)

Task: Create a new Review object and add it to the database.

Solution:

```
def add_review(movie_id, review):
    session = get_session()
    try:
        new_review = Review(
            movie_id=movie_id,
            reviewer_name=review["name"],
            rating=review["rating"],
            comment=review["comment"]
        )
        session.add(new_review)
        session.commit()
    finally:
        session.close()
```

Key Points:

- Access dictionary values using review["name"], review["rating"], review["comment"]
- Use session.add() to add the new object
- Use session.commit() to save changes to the database
- The movie_id parameter links the review to the correct movie

Challenge 4: Get Average Rating (Medium)

Task: Calculate the average (mean) of all ratings for a movie.

Solution:

```
def get_average_rating(movie_id):
    movie = get_movie_by_id(movie_id)

    if movie is None:
        return 0

    reviews = movie.get("reviews", [])

    if len(reviews) == 0:
        return 0

    # Add up all the ratings
    total = 0
    for review in reviews:
        total = total + review["rating"]

    # Divide by number of reviews
    average = total / len(reviews)

    # Round to 1 decimal place and return
    return round(average, 1)
```

Key Points:

- First get the movie and check if it exists
- Get the reviews list from the movie dictionary
- Check if there are no reviews (return 0)
- Loop through reviews and add up all ratings
- Divide total by count to get average
- Use round(average, 1) to round to 1 decimal place

Challenge 5: Search Movies (Easy)

Task: Search for movies where the title contains the search query (case-insensitive).

Solution:

```
def search_movies(query):
    # Handle empty queries
    if query == "":
        return load_movies()

    # Get all movies
    movies = load_movies()

    # Convert query to lowercase
    query_lower = query.lower()
    results = []

    # Check if query is in the title
    for movie in movies:
        if query_lower in movie["title"].lower():
            results.append(movie)

    return results
```

Key Points:

- Return all movies if query is empty
- Convert both query and title to lowercase for case-insensitive search
- Use 'in' operator to check if query is part of the title
- Build a results list with matching movies

Challenge 6: Get Top Rated Movies (Medium)

Task: Find movies with reviews, sort by rating (highest first), return top 'limit' movies.

Solution:

```
def get_top_rated_movies(limit=5):
    movies = load_movies()
    rated_movies = []

    # Populate rated_movies with movies that have reviews
    for movie in movies:
        rating = get_average_rating(movie["id"])
        if rating > 0:
            movie["avg_rating"] = rating
            rated_movies.append(movie)

    # Sort by rating, highest first
    sorted_movies = sorted(rated_movies,
                           key=lambda m: m["avg_rating"],
                           reverse=True)

    # Return only the top 'limit' movies
    return sorted_movies[:limit]
```

Key Points:

- Loop through all movies and calculate their average rating
- Only include movies with rating > 0 (meaning they have reviews)
- Add the avg_rating to each movie dictionary
- Use sorted() with lambda to sort by avg_rating
- Use reverse=True for highest to lowest
- Use slicing [:limit] to get only the first 'limit' movies

Challenge 7: Get Movies by Genre (Bonus - Easy)

Task: Filter movies to show only those matching the given genre (case-insensitive).

Solution:

```
def get_movies_by_genre(genre):
    movies = load_movies()

    # Handle empty genre
    if genre == "":
        return movies

    # Convert genre to lowercase for comparison
    genre_lower = genre.lower()
    results = []

    # Check each film for matching genre
    for movie in movies:
        if movie["genre"].lower() == genre_lower:
            results.append(movie)

    return results
```

Key Points:

- Return all movies if genre is empty
- Convert both input genre and movie genre to lowercase
- Use == for exact match (not 'in')

Challenge 8: Count Reviews (Bonus - Easy)

Task: Return the number of reviews for a specific movie.

Solution:

```
def count_reviews(movie_id):
    movie = get_movie_by_id(movie_id)

    if movie is None:
        return 0

    return len(movie.get("reviews", []))
```

Key Points:

- Get the movie first
- Check if movie exists (return 0 if not)
- Use len() to count the reviews list
- Use .get("reviews", []) to safely get reviews (empty list if none)

Challenge 9: Get All Genres (Bonus - Medium)

Task: Return a list of all unique genres (no duplicates), sorted alphabetically.

Solution:

```
def get_all_genres():
    movies = load_movies()
    genres = []

    for movie in movies:
        genre = movie["genre"]
        if genre not in genres:
            genres.append(genre)

    # Sort alphabetically
    genres.sort()
    return genres
```

Key Points:

- Create an empty list to store unique genres
- Loop through movies and check if genre already exists
- Only add genre if it's not already in the list
- Sort the list alphabetically before returning

Challenge 10: Delete a Review (Bonus - Medium)

Task: Remove a review from the database using its ID.

Solution:

```
def delete_review(review_id):
    session = get_session()
    try:
        review = session.query(Review).filter(Review.id == review_id).first()
        if review:
            session.delete(review)
            session.commit()
    finally:
        session.close()
```

Key Points:

- Create a session
- Find the review using filter() and .first()
- Check if review exists before deleting
- Use session.delete() to remove the object
- Use session.commit() to save the deletion
- Always close session in finally block

Quick Reference Card

Challenge 2 (Get by ID): movie.to_dict() if movie else None

Challenge 3 (Add Review): review["name"], review["rating"], review["comment"]

Challenge 4 (Average): total / len(reviews), round(avg, 1)

Challenge 5 (Search): query.lower() in title.lower()

Challenge 6 (Top Rated): sorted(..., key=lambda m: m["avg_rating"], reverse=True)[:limit]

Challenge 7 (By Genre): movie["genre"].lower() == genre.lower()

Challenge 8 (Count): len(movie.get("reviews", []))

Challenge 9 (All Genres): if genre not in genres: genres.append(genre)

Challenge 10 (Delete): session.delete(review), session.commit()