**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**

**ІМЕНІ ІГОРЯ СІКОРСЬКОГО”**

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**КУРСОВА РОБОТА**

з дисципліни “Бази даних”

спеціальність 121 – Програмна інженерія

на тему: “База даних фільмів та серіалів”

| **Студент групи** КП-01 | **Яблонський Олексій Васильович** | \_\_\_\_\_\_\_\_\_  (підпис) |
| --- | --- | --- |
| **Асистент кафедри СПіСКС** | **Радченко К.О.** | \_\_\_\_\_\_\_\_\_  (підпис) |

Захищено з оцінкою \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Київ – 2021

# **Анотація**

Курсова робота складається з бази даних відомостей про фільми та акторів (рецензії, акторський склад фільму, фільми у яких знімався певний актор тощо) та віконного додатку зі зручним інтерфейсом користувача для адміністративної взаємодії з цією базою даних, написаного мовою програмування C#, який дозволяє виконувати певні операції над нею (читання, оновлення, запис, видалення, псевдовипадкова генерація записів, аналіз даних тощо).

У результаті розробки даної бази даних та даного консольного додатку було набуто практичні навички розробкисучасного програмного забезпечення, що взаємодіє з реляційними базами даних, а також здобуто навички оформлення відповідного текстового, програмного та ілюстративного матеріалу у формі проєктної документації.

# **Зміст**

[Анотація](#_5kfm0grkguqg) **2**

[Зміст](#_8900uq8q3g4o) **3**

[Вступ](#_rbueiup52x39) **4**

[1. Аналіз інструментарію для виконання курсової роботи](#_1ogq8h34ns9v) **5**

[2. Структура бази даних](#_s7jber8whvhf) **6**

[3. Опис програмного забезпечення](#_bpvlhn62fcad) **7**

[3.1. Загальна структура програмного забезпечення](#_v3j0luielms5) 7

[3.2. Опис модулів програмного забезпечення](#_irnayr5v7pc) 7

[3.3. Опис основних алгоритмів роботи](#_iy8o9ns8uksl) 7

[4. Аналіз функціонування засобів резервування/відновлення бази даних](#_w06e4iluq5nz) **9**

5[. Опис результатів аналізу предметної галузі](#_44j3l8oqx8v4) **11**

[Висновки](#_2e5c06vg6llg) **12**

[Додатки](#_1t31go35u03g) **12**

[А. Графічні матеріали](#_8e53kjbnbkg1) 12

[Б. Фрагменти програмного коду](#_h9dx89qx8je5) 15

# **Вступ**

Галузь застосування розробки: рецензування кінофільмів: аутентифікація користувачів задля персоналізованих рецензій на фільми, серіали тощо, управління базою фільмів, акторів та рецензій.

Метою курсової роботи є набуття практичних навичок розробки сучасного програмного забезпечення, що взаємодіє з реляційними базами даних, а також здобуття навичок оформлення відповідного текстового, програмного та ілюстративного матеріалу у формі проєктної документації, на прикладі розробки системи, що допоможе в організації зручної взаємодії з великими базами фільмів та їх акторів для створення рецензій. База має такі сутності:

* Фільми
* Актори
* Рецензії
* Користувачі

База даних була спроєктована відповідно до 3-ої нормальної форми,

швидкість її роботи була оптимізована шляхом додавання індексів, також вона була убезпечена додаванням механізму резервного копіювання. Окрім того, розроблений віконний додаток може надавати результати певного аналізу даних у базі.

# **1. Аналіз інструментарію для виконання курсової роботи**

1. Для виконання даної роботи у якості системи керування базами даних було обрано SQLite. Такий вибір був зроблений у зв’язку з такими факторами:

* відкрите ПЗ відповідає стандарту SQL - SQLite - безкоштовне ПЗ з відкритим вихідним кодом. Ця СУБД є дуже потужною системою;
* SQLite є простою, як у розумінні так і у роботі, її функціонал достатній для розробки даного додатку;
* підтримка великої кількості типів даних, включно з власними;
* цілісність даних з усіма необхідними обмеженнями;
* надійність, безпека;
* працює з багатьма типами мереж;
* велика місткість;
* велика спільнота – просто знайти вирішення потенційних проблем при розробці
* це повністю опен-сорсний проєкт;
* розширення - існує можливість розширення функціоналу за рахунок своїх процедур.

1. Для взаємодії з базою даних було обрано бібліотеку Microsoft.Data.SQLite, оскільки:

* добре підходить для зручного використання у мові програмування C#;
* розроблена спеціально для SQLite;
* найпопулярніша для взаємодії з SQLite у мові програмування C#;
* має чітку, зрозумілу та вичерпну документацію з хорошими прикладами.

1. Для візуалізації результатів аналізу даних було обрано бібліотеку ScottPlot, оскільки:

* вона надає зручний інтерфейс для автоматичного будування графічних об’єктів;
* наявна можливість будувати надзвичайно різноманітні графічні об’єкти.

# **2. Структура бази даних**

База даних має 5 таблиць:

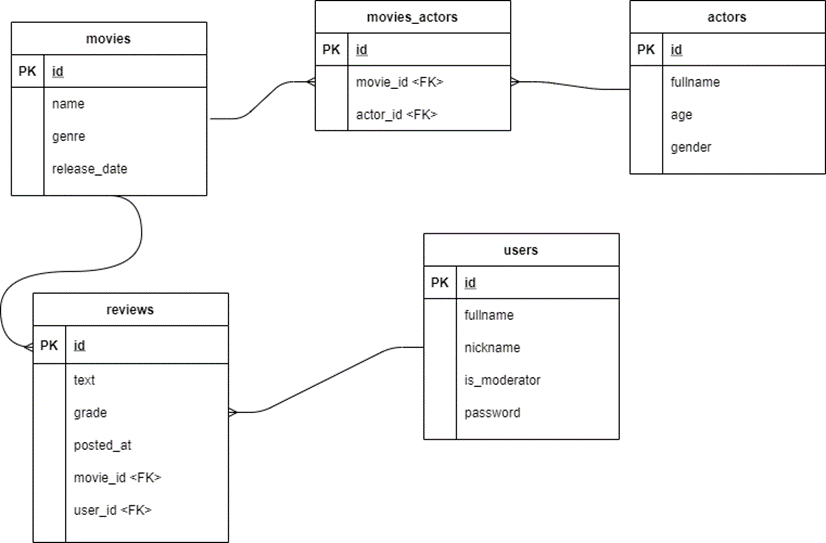
1. movies

2. actors

3. reviews

4. users

5. movies\_actors



movies:

Атрибути:

● id

● name

● genre

● release\_date

Дана таблиця має зв'язок one2many з таблицею reviews, тобто один фільм може мати багато рецензій, але рецензія може стосуватись тільки одного фільму. Також дана таблиця має зв’язок many2many з таблицею actors: у одному фільмі може зніматись багато акторів і один актор може зніматись у багатьох фільмах.

actors:

Атрибути:

● id

● fullname

● age

● gender

Дана таблиця має зв’язок many2many з таблицею movies, що описано вище.

movies\_actors:

Атрибути:

● id

● movie\_id

● actor\_id

Дана таблиця є допоміжною для реалізації зв'язку many2many між movies та actors.

Ця таблиця має зв'язки many2one з таблицями movies та actors.

reviews:

Атрибути:

● id

● text

● grade

● posted\_at

● movie\_id

● user\_id

Ця таблиця має зв’язки many2one з таблицями movies i users, які реалізуються за допомогою зовнішніх ключів movie\_id, user\_id.

users:

Атрибути:

* id
* fullname
* nickname
* is\_moderator
* password

Ця таблиця має one2many зв’язок з reviews оскільки один користувач може писати багато рецензій, але рецензія завжди належить одному користувачу.

# **3. Опис програмного забезпечення**

## **3.1. Загальна структура програмного забезпечення**

Розроблене програмне забезпечення містить такі компоненти:

1. База даних, що зберігає інформацію про фільми, акторів, рецензії та користувачів.
2. Засоби CRUD-функціоналу.
3. Засоби псевдовипадкової генерації даних.
4. Засоби пошуку, фільтрації та валідації.
5. Засоби резервного копіювання з можливістю вибору версії
6. Засоби аналізу даних

## **3.2. Опис модулів програмного забезпечення**

Розроблене програмне забезпечення було розбите на такі модулі:

1. DBLib

Даний модуль напряму взаємодіє з базою даних. В цьому модулі знаходяться запити до бази даних, їх обробка та пов’язані з цим операції.

1. MoviesAndActorsGUI

Даний модуль відповідає за зручну взаємодію користувача з базою даних, за допомогою віконного інтерфейсу Terminal.Gui.

1. DataGenerator

Цей модуль відповідає за генерацію псевдовипадкових даних.

## **3.3. Опис основних алгоритмів роботи**

При псевдовипадковій генерації даних для кожної із таблиць процес генерації був побудований так, аби генерувалися адекватні дані для обраної предметної галузі та для обраної структури бази даних. Зокрема, у запитах були застосовані власноруч розроблені SQLite-функції для забезпечення цілісності даних, унікальності тощо.

Для генерації даних шляхом імпорту відбувається обробка вхідних даних для забезпечення коректного їх занесення у таблиці, оскільки стиль датасетів, обраних для цієї мети, не повністю відповідає структурі бази даних.

Для аналізу різних даних було використано генерацію графіку та діаграм.

# **4. Аналіз функціонування засобів резервування/відновлення бази даних**

Резервне копіювання необхідне для забезпечення безпечного та швидкого відновлення даних у разі втрати їх із бази даних. Тип резервного копіювання, який було реалізовано - повне. Це такий вид резервного копіювання, у якому щоразу копіюються повністю всі дані. Перевага такого різновиду резервного копіювання полягає у тому, що не потрібно об’єднувати різні файли для відновлення, натомість відновлюється все з одного файлу, за рахунок чого відновлення є помітно швидшим порівняно з іншими видами резервного копіювання. Також було реалізовано розширену функціональність - експорт та імпорт у XML форматі.

Розроблений додаток надає можливість користувачеві керувати резервним копіюванням та відновленням з консолі за допомогою аргументів командного рядка. Користувач сам може ввести шлях і назву файлу. При виконанні відновлення додаток необхідно назву і шлях до файлу бекапу і потім база даних відновлюється відповідно до бекапу з вказаної папки.

# **5. Опис результатів аналізу предметної галузі**

У розробленому консольному додатку наявний такий аналіз даних, що містяться у базі:

* знаходження середнього значення, медіани та моди (і їх порівняння на діаграмі) вибірки віку акторів.
* знаходження середнього значення, медіани та моди (і їх порівняння на діаграмі) вибірки року випуску фільмів.
* аналіз частоти у вибірці віку акторів.
* аналіз частоти у вибірці року випуску фільмів (у які роки було випущено скільки фільмів)
* підрахунок середньої оцінки фільму (показується у інтерфейсі користувача перед назвою фільму)

Результати кожного статистичного аналізу подаються у вигляді графіків та діаграм. Приклади графіків та діаграм наведені у додатках.

# **Висновки**

Під час виконання даної курсової роботи виконано таку роботу та отримано такі результати:

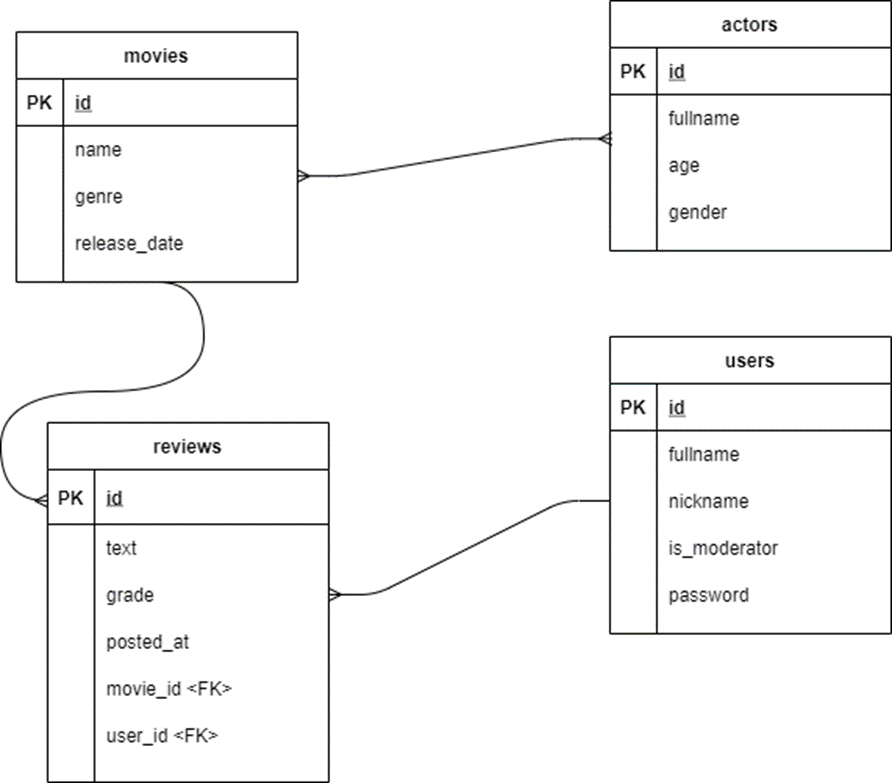
* Було розроблено базу даних, яка відповідає 3-ій нормальній формі та організована максимально зручно та просто
* Резервне копіювання було реалізовано повне, що дає можливість швидкого відновлення
* Була розроблена псевдовипадкова генерація для всіх таблиць, яка генерує реалістичні значення
* Були розроблені засоби для аналізу даних із бази, які також надають можливість виводити графічне представлення його результату для наочності висновків
* Був розроблений зручний віконний інтерфейс який також обробляє всі помилки, валідує дані та надає можливість виконання CRUD-операцій та фільтрації командами, а не SQL-запитами
* Був розроблений розширений функціонал: реєстрація та аутентифікація, ролі (модерування), пагінація (програма розбиває базу даних на сторінки) зі зручною навігацією, експорт/імпорт у XML форматі тощо.

У результаті виконання даної курсової роботи було набуто практичні навички розробкисучасного програмного забезпечення, що взаємодіє з реляційними базами даних, а також здобуто навички оформлення відповідного текстового, програмного та ілюстративного матеріалу у формі проєктної документації.

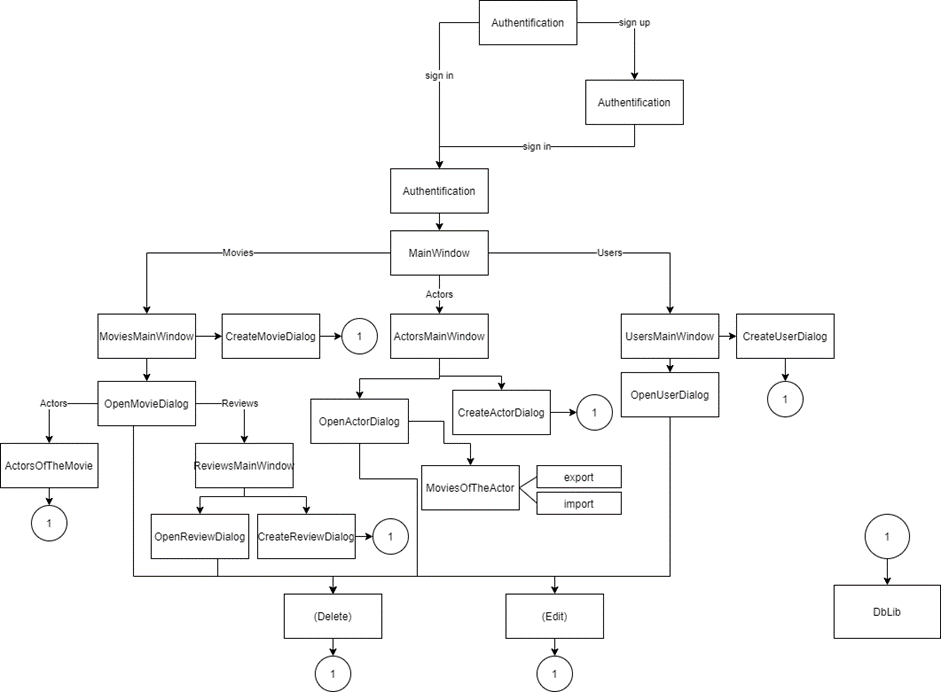
Завдяки виконанню даної роботи було здобуто вміння розробляти програмне забезпечення для реляційних баз даних, відбулося оволодіння основами використання СУБД, а також інструментальними засобами підтримки розробки додатків для подібних баз даних.

# **Додатки**

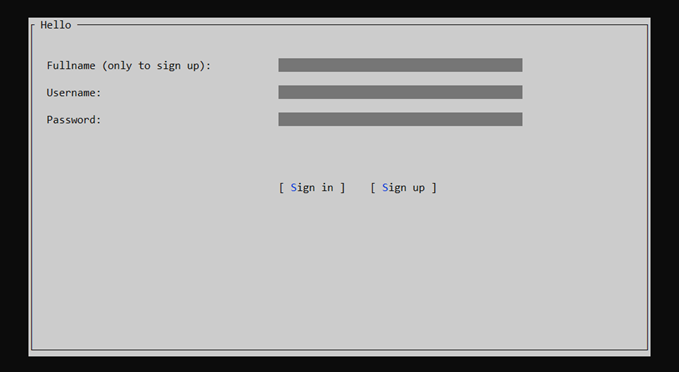
## **А. Графічні матеріали**



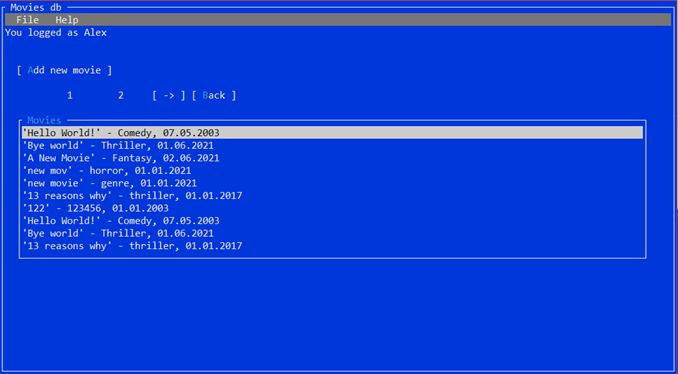
Структура бази даних



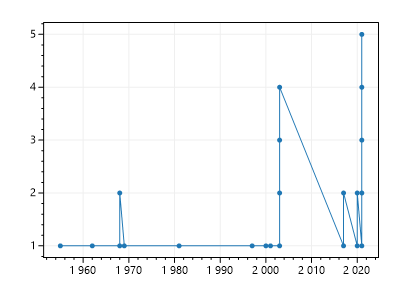
Діаграма залежностей модулів і навігації по програмі



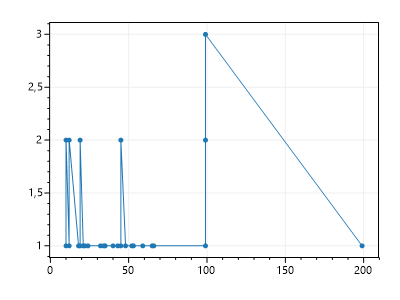
Віконний інтерфейс користувача - аутентифікація



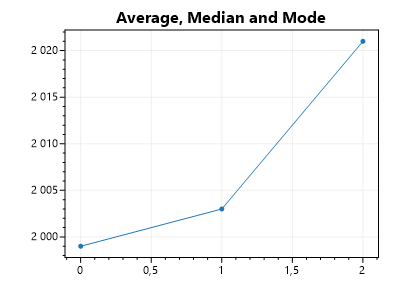
Вікно перегляду сутностей



Приклад графіка року випуску фільмів



Приклад графіка віку акторів



Приклад діаграми порівняння середнього значення, медіани та моди

## **Б. Фрагменти програмного коду**

| Приклади CRUD |
| --- |
| public class MovieRepository  {  private SqliteConnection connection;  public MovieRepository(SqliteConnection connection)  {  this.connection = connection;  }  public Movie GetById(long id)  {  connection.Open();  SqliteCommand command = connection.CreateCommand();  command.CommandText = @"SELECT \* FROM movies WHERE id = $id";  command.Parameters.AddWithValue("$id", id);  SqliteDataReader reader = command.ExecuteReader();  Movie movie = new Movie();  if (reader.Read())  {  movie.id = reader.GetInt64(0);  movie.name = reader.GetString(1);  movie.genre = reader.GetString(2);  movie.releaseDate = DateTime.Parse(reader.GetString(3));  }  else  {  movie = null;  }  reader.Close();  connection.Close();  return movie;  }  public bool DeleteById(long id)  {  connection.Open();  SqliteCommand command = connection.CreateCommand();  command.CommandText = @"DELETE FROM movies WHERE id = $id";  command.Parameters.AddWithValue("$id", id);  long nChanged = command.ExecuteNonQuery();  connection.Close();  return nChanged != 0;  }  public long Insert(Movie movie)  {  connection.Open();  SqliteCommand command = connection.CreateCommand();  command.CommandText =  @"  INSERT INTO movies (name, genre, release\_date)  VALUES ($name, $genre, $release\_date);    SELECT last\_insert\_rowid();  ";  command.Parameters.AddWithValue("$name", movie.name);  command.Parameters.AddWithValue("$genre", movie.genre);  command.Parameters.AddWithValue("$release\_date", movie.releaseDate.ToShortDateString());  long newId = (long)command.ExecuteScalar();  connection.Close();  return newId;  }  private long GetCount()  {  connection.Open();  SqliteCommand command = connection.CreateCommand();  command.CommandText = @"SELECT COUNT(\*) FROM movies";  long count = (long)command.ExecuteScalar();  connection.Close();  return count;  }  public List<Movie> GetAll()  {  connection.Open();  SqliteCommand command = connection.CreateCommand();  command.CommandText = @"SELECT \* FROM movies";  SqliteDataReader reader = command.ExecuteReader();  List<Movie> movies = new List<Movie>();  while (reader.Read())  {  Movie movie = new Movie(reader.GetInt64(0), reader.GetString(1), reader.GetString(2), DateTime.Parse(reader.GetString(3)));  movies.Add(movie);  }  reader.Close();  connection.Close();  return movies;  }  public long GetTotalPages()  {  connection.Open();  const long pageSize = 10;  long total = (long)Math.Ceiling(this.GetCount() / (double)pageSize);  connection.Close();  return total;  }  public List<Movie> GetPage(long pageNumber)  {  connection.Open();  pageNumber -= 1;  SqliteCommand command = connection.CreateCommand();  command.CommandText = @"SELECT \* FROM movies LIMIT $limit OFFSET $offset";  command.Parameters.AddWithValue("$limit", 10);  command.Parameters.AddWithValue("$offset", pageNumber \* 10);  SqliteDataReader reader = command.ExecuteReader();  List<Movie> movies = new List<Movie>();  while (reader.Read())  {  Movie movie = new Movie(reader.GetInt64(0), reader.GetString(1), reader.GetString(2), DateTime.Parse(reader.GetString(3)));  movies.Add(movie);  }  reader.Close();  connection.Close();  return movies;  }  public bool Update(long taskId, Movie movie)  {  connection.Open();  SqliteCommand command = connection.CreateCommand();  command.CommandText = @"UPDATE movies SET name = $name, genre = $genre, release\_date = $release\_date WHERE id = $id";  command.Parameters.AddWithValue("$id", taskId);  command.Parameters.AddWithValue("$name", movie.name);  command.Parameters.AddWithValue("$genre", movie.genre);  command.Parameters.AddWithValue("$release\_date", movie.releaseDate.ToShortDateString());  long nChanged = command.ExecuteNonQuery();  connection.Close();  return nChanged != 0;  }  public List<Review> GetAllReviews(Movie movie)  {  connection.Open();  SqliteCommand command = connection.CreateCommand();  command.CommandText = @"SELECT \* FROM reviews WHERE movie\_id = $movie\_id";  command.Parameters.AddWithValue("$movie\_id", movie.id);  SqliteDataReader reader = command.ExecuteReader();  List<Review> reviews = new List<Review>();  while (reader.Read())  {  Review review = new Review(reader.GetInt64(0), reader.GetString(1), reader.GetInt32(2), DateTime.Parse(reader.GetString(3)));  reviews.Add(review);  }  reader.Close();  movie.reviews = reviews;  connection.Close();  return reviews;  }  public List<Actor> MovieActors(long movieId)  {  connection.Open();  SqliteCommand command = connection.CreateCommand();  command.CommandText = @"SELECT movies\_actors.movie\_id, actors.id, actors.fullname, actors.age, actors.gender  FROM movies\_actors, actors WHERE movies\_actors.movie\_id = $movie\_id  AND actors.id = movies\_actors.actor\_id";  command.Parameters.AddWithValue("$movie\_id", movieId);  SqliteDataReader reader = command.ExecuteReader();  List<Actor> actors = new List<Actor>();  while (reader.Read())  {  string fullName = reader.GetString(2);  int age = reader.GetInt32(3);  string gender = reader.GetString(4);  Actor newActor = new Actor(fullName, age, gender);  newActor.id = int.Parse(reader.GetString(1));  actors.Add(newActor);  }  reader.Close();  connection.Close();  Movie movie = GetById(movieId);  movie.actors = actors;  return actors;  }  public SqliteConnection GetConnection()  {  return connection;  }  } |

| Псевдовипадкова генерація |
| --- |
| public class DataGenerator  {  private SqliteConnection connection;  public void Run(SqliteConnection connection)  {  this.connection = connection;  Console.WriteLine("This generator generates random entities with random relations! Enjoy:)");  while (true)  {  Console.WriteLine("command format '{entity to generate} {amount of generating entities}'\r\nEntities: movie, actor, review\r\nType 'quit' to stop the program.");  Console.Write("Type your command: ");  string command = Console.ReadLine();  string[] subcommands = command.Split(' ');  if (command == "quit" || command == "exit")  {  Console.WriteLine("The generator is stopped, goodbye:)");  break;  }  if (subcommands.Length != 2)  {  Console.WriteLine($"Command should consist of 2 parts, you wrote {subcommands.Length}");  continue;  }  int quantity = 0;  if (!int.TryParse(subcommands[1], out quantity))  {  Console.WriteLine($"The second part of command is not integer, you wrote {subcommands[1]}");  continue;  }  if (subcommands[0] == "movie")  {  GenerateFilms(quantity);  Console.WriteLine($"Generated {quantity} movies.");  }  else if (subcommands[0] == "actor")  {  GenerateActors(quantity);  Console.WriteLine($"Generated {quantity} actors.");  }  else if (subcommands[0] == "review")  {  GenerateReviews(quantity);  Console.WriteLine($"Generated {quantity} reviews.");  }  else  {  Console.WriteLine($"Unknown command '{command}'");  continue;  }  }  }  public void GenerateFilms(int quantity)  {  string[] namesBeginnings = new string[]  {  "Hello ", "Meet ", "Let me introduce you to ", "Midnight ", "Just ", "Dear ", "Dangerous ", "Scary ",  "Angry ", "Nice ", "Classy ", "Kind ", "Style of ", "Forbidden ", "Some interesting facts about ",  "Your ", "Forget about ", "Forgive ", "Classical ", "Sweet ", "It is ", "Cry with ", "It's just ",  "It's all about ", "All about ", "Stop thinking about ", "Lovely ", "Sincerely your`s, ", "Literally ",  "Friendly ",  };  string[] namesEndings = new string[]  {  "love", "Paris", "Kyiv", "New York", "Hollywood", "Berlin", "Amsterdam", "London", "friends",  "Emily", "Sara", "Betty", "Hanna", "Carla", "Tanya", "Sabrina", "Rose", "Veronica", "Diana", "Kate",  "Andrew", "Matt", "Alex", "Harry", "Thomas", "Lenny", "Michael", "Bryce", "Simon", "Victor", "James",  "John", "Jeremy",  };  string[] genres = new string[]  {  "Action", "Comedy", "Drama", "Fantasy", "Horror", "Mystery", "Romance", "Thriller", "Western",  };  Random random = new Random();  MovieRepository movieRepository = new MovieRepository(connection);  ActorRepository actorRepository = new ActorRepository(connection);  MovieActorRepository movieActorRepository = new MovieActorRepository(connection);    int actorsAmount = actorRepository.GetAll().Count;  for (int i = 0; i < quantity; i++)  {  string name = namesBeginnings[random.Next(0, namesBeginnings.Length)] + namesEndings[random.Next(0, namesEndings.Length)];  string genre = genres[random.Next(0, genres.Length)];  DateTime releaseDate = DateTime.Parse($"{random.Next(1, 29)}.{random.Next(1, 13)}.{random.Next(1950, 2022)}");  Movie movie = new Movie(name, genre, releaseDate);  long movieId = movieRepository.Insert(movie);  movieActorRepository.Insert(new MovieActor(movieId, random.Next(1, actorsAmount)));  }  }  public void GenerateActors(int quantity)  {  string[] firstNames = new string[]  {  "Emily ", "Sara ", "Betty ", "Hanna ", "Carla ", "Tanya ", "Sabrina ", "Rose ", "Veronica ", "Diana ", "Kate ",  "Andrew ", "Matt ", "Alex ", "Harry ", "Thomas ", "Lenny ", "Michael ", "Bryce ", "Simon ", "Victor ", "James ",  "John ", "Jeremy ",  };  string[] lastNames = new string[]  {  "Smith","Johnson","Williams","Brown","Jones","Garcia","Miller","Davis","Rodriguez","Martinez","Hernandez",  "Lopez","Gonzalez","Wilson","Anderson","Thomas","Taylor","Moore","Jackson","Martin","Lee","Perez",  "Thompson","White","Harris","Sanchez","Clark","Ramirez","Lewis","Robinson","Walker","Young","Allen",  "King","Wright","Scott","Torres","Nguyen","Hill","Flores","Green","Adams","Nelson","Baker","Hall",  "Rivera","Campbell","Mitchell","Carter","Roberts",  };  string[] genders = new string[] { "Male", "Female", "Other" };  Random random = new Random();  MovieRepository movieRepository = new MovieRepository(connection);  ActorRepository actorRepository = new ActorRepository(connection);  int movieAmount = movieRepository.GetAll().Count;  MovieActorRepository movieActorRepository = new MovieActorRepository(connection);  for (int i = 0; i < quantity; i++)  {  string name = firstNames[random.Next(0, firstNames.Length)] + lastNames[random.Next(0, lastNames.Length)];  int age = random.Next(10, 70);  string gender = genders[random.Next(0, genders.Length)];  Actor newActor = new Actor(name, age, gender);  long actorId = actorRepository.Insert(newActor);  movieActorRepository.Insert(new MovieActor(random.Next(1, movieAmount), actorId));  }  }  public void GenerateReviews(int quantity)  {  string[] part1 = new string[]  {  "This ", "The ",  };  string[] part2 = new string[]  {  "film ", "movie ", "pic ", "tape ",  };  string[] part3 = new string[]  {  "is ", "is really ", "is literally ", "is actually ", "turned out to be ",  };  string[] part4 = new string[]  {  "very ", "so much ", "",  };  string[] part5 = new string[]  {  "interesting.", "exciting.", "funny.", "ridiculous.", "unusual", "usual.", "boring.",  "good.", "bad.", "genial.", "ingenious.", "scary.", "hillarious.", "sad.", "facsinating."  };  UserRepository userRepository = new UserRepository(connection);  MovieRepository movieRepository = new MovieRepository(connection);  ReviewRepository reviewRepository = new ReviewRepository(connection);  int userAmount = userRepository.GetAll().Count;  int moviesAmount = movieRepository.GetAll().Count;  Random random = new Random();  for (int i = 0; i < quantity; i++)  {  string text = $"{part1[random.Next(0, part1.Length)]}{part2[random.Next(0, part2.Length)]}{part3[random.Next(0, part3.Length)]}{part4[random.Next(0, part4.Length)]}{part5[random.Next(0, part5.Length)]}";  int grade = random.Next(1, 11);  DateTime postedAt = DateTime.Parse($"{random.Next(1, 29)}.{random.Next(1, 13)}.{random.Next(2015, 2022)}");  Review newReview = new Review()  {  text = text,  grade = grade,  postedAt = postedAt,  author = userRepository.GetById(random.Next(1, userAmount)),  movie = movieRepository.GetById(random.Next(1, moviesAmount)),  };  reviewRepository.Insert(newReview);  }  }  } |

| Приклад фільтрації |
| --- |
| public List<Actor> MovieActors(long movieId)  {  connection.Open();  SqliteCommand command = connection.CreateCommand();  command.CommandText = @"SELECT movies\_actors.movie\_id, actors.id, actors.fullname, actors.age, actors.gender  FROM movies\_actors, actors WHERE movies\_actors.movie\_id = $movie\_id  AND actors.id = movies\_actors.actor\_id";  command.Parameters.AddWithValue("$movie\_id", movieId);  SqliteDataReader reader = command.ExecuteReader();  List<Actor> actors = new List<Actor>();  while (reader.Read())  {  string fullName = reader.GetString(2);  int age = reader.GetInt32(3);  string gender = reader.GetString(4);  Actor newActor = new Actor(fullName, age, gender);  newActor.id = int.Parse(reader.GetString(1));  actors.Add(newActor);  }  reader.Close();  connection.Close();  Movie movie = GetById(movieId);  movie.actors = actors;  return actors;  } |

| Резервне копіювання та відновлення |
| --- |
| else if (args[0] == "backup")  {  string backupPath = args[1];  File.Copy(databaseFilePath, backupPath);  Console.WriteLine($"Backup saved in {backupPath}. To load it, type 'dotnet run load {backupPath}'");  }  else if (args[0] == "load")  {  SqliteConnection newConnection = new SqliteConnection($"Data Source={args[1]}");  RunGui(newConnection);  }  —--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------  public static class ExportAndImport  {  public static void Export(string filepath, List<Movie> list)  {  XmlSerializer ser = new XmlSerializer(typeof(List<Movie>));  StreamWriter writer = new StreamWriter(filepath);  ser.Serialize(writer, list);  writer.Close();  }  public static List<Movie> Import(string filepath)  {  XmlSerializer ser = new XmlSerializer(typeof(List<Movie>));  StreamReader reader = new StreamReader(filepath);  List<Movie> value = (List<Movie>)ser.Deserialize(reader);  reader.Close();  return value;  }    } |

| Створення статистики |
| --- |
| public static class StatisticAnalyzer  {  public static void BasicPlot(List<int> numbers)  {  double[] dataY = new double[numbers.Count];  numbers.Sort();  double[] dataX = new double[numbers.Count];  for (int i = 0; i < dataY.Length; i++)  {  dataX[i] = numbers[i];  int amount = 0;  for (int j = 0; j < dataY.Length; j++)  {  if (dataX[i] == dataX[j])  {  amount++;  }  }  dataY[i] = amount;  }  Random r = new Random();  ScottPlot.Plot plt = new ScottPlot.Plot(400, 300);  plt.AddScatter(dataX, dataY);  plt.SaveFig($"{r.Next(1, 1000000)}.png");  }  public static void StatisticDataGetter(List<int> numbers)  {  Random r = new Random();  ScottPlot.Plot plt = new ScottPlot.Plot(400, 300);  plt.Title("Average, Median and Mode");  plt.AddSignal(new double[] {GetAverage(numbers), Median(numbers), Mode(numbers)});  plt.SaveFig($"{r.Next(1, 1000000)}.png");  }  private static double Median(List<int> numbers)  {  numbers.Sort();  return numbers[numbers.Count/2];  }  static double Mode(List<int> numbers)  {  Dictionary<double, int> dict = new Dictionary<double, int>();  foreach(double elem in numbers)  {  if (dict.ContainsKey(elem))  dict[elem]++;  else  dict[elem] = 1;  }    int maxCount = 0;  double mode = Double.NaN;  foreach(double elem in dict.Keys)  {  if (dict[elem] > maxCount)  {  maxCount = dict[elem];  mode = elem;  }  }    return mode;  }  private static int GetSum(List<int> numbers)  {  int sum = 0;  foreach (int num in numbers)  {  sum += num;  }  return sum;  }  private static int GetAverage(List<int> numbers)  {  int sum = GetSum(numbers);  int average = sum / numbers.Count;  return average;  }  private static int GetMin(List<int> numbers)  {  int min = int.MaxValue;  for (int i = 0; i < numbers.Count; i++)  {  if (min > numbers[i])  {  min = numbers[i];  }  }  return min;  }  private static int GetMax(List<int> numbers)  {  int max = int.MinValue;  for (int i = 0; i < numbers.Count; i++)  {  if (max < numbers[i])  {  max = numbers[i];  }  }  return max;  }  } |

| Приклади коду MoviesAndActorsGUI |
| --- |
| class OpenMovieDialog : Dialog  {  protected Movie movie;  public bool deleted;  public bool updated;  public SqliteConnection connection;  protected TextField nameInput;  protected TextField genreInput;  protected DateField dateInput;  protected User currentUser;  public OpenMovieDialog(User currentUser)  {  this.currentUser = currentUser;  this.Title = "Open movie";  Button backBtn = new Button("Back");  backBtn.Clicked += OnOpenDialogBack;  this.AddButton(backBtn);  int rightColumnX = 20;  Label nameLbl = new Label(2, 2, "Name:");  nameInput = new TextField("")  {  X = rightColumnX,  Y = Pos.Top(nameLbl),  Width = 40,  ReadOnly = true,  };  this.Add(nameLbl, nameInput);  Label genreLbl = new Label(2, 4, "Genre:");  genreInput = new TextField("")  {  X = rightColumnX,  Y = Pos.Top(genreLbl),  Width = 40,  ReadOnly = true,  };  this.Add(genreLbl, genreInput);  Label dateLbl = new Label(2, 6, "Release date:");  dateInput = new DateField()  {  X = rightColumnX,  Y = Pos.Top(dateLbl),  Width = 40,  IsShortFormat = false,  ReadOnly = true,  };  this.Add(dateLbl, dateInput);  Button editButton = new Button(2, 12, "Edit");  editButton.Clicked += OnMovieEdit;  this.Add(editButton);  Button deleteButton = new Button("Delete")  {  X = Pos.Right(editButton) + 2,  Y = Pos.Top(editButton),  };  if (!currentUser.isModerator)  {  deleteButton.Visible = false;  editButton.Visible = false;  }  deleteButton.Clicked += OnMovieDelete;  this.Add(deleteButton);  Label userLbl = new Label($"You logged as {currentUser.fullname}");  this.Add(userLbl);  }  private void ReviewsAndActorsSetButtons()  {  Button reviewButton = new Button("Reviews")  {  Y = 1,  };  this.Add(reviewButton);  reviewButton.Clicked += OnReviews;  Button actorsButton = new Button("Actors")  {  X = Pos.Right(reviewButton) + 1,  Y = 1  };  if (movie == null) actorsButton.Visible = false;  this.Add(actorsButton);  actorsButton.Clicked += OnActors;  }  private void OnActors()  {  try  {  ActorsOfTheMovie win = new ActorsOfTheMovie(currentUser, movie) { connection = connection };  win.SetRepository(new ActorRepository(connection));  Application.Run(win);  }  catch  {  MessageBox.ErrorQuery("ERROR", "Can not do this, try to use another way to see the data.", "OK");  }  }  private void OnReviews()  {  ReviewsMainWindow win = new ReviewsMainWindow(currentUser, this.movie);  win.SetRepository(new ReviewRepository(connection));  Application.Run(win);  }  public void SetMovie(Movie movie)  {  this.movie = movie;  this.nameInput.Text = movie.name;  this.genreInput.Text = movie.genre;  this.dateInput.Text = movie.releaseDate.ToShortDateString();  ReviewsAndActorsSetButtons();  }  private void OnOpenDialogBack()  {  Application.RequestStop();  }  private void OnMovieEdit()  {  EditMovieDialog dialog = new EditMovieDialog(currentUser);  dialog.SetMovie(this.movie);  Application.Run(dialog);  if (!dialog.canceled)  {  Movie updatedMovie = dialog.GetMovie();  this.updated = true;  this.SetMovie(updatedMovie);  }  }  private void OnMovieDelete()  {  int index = MessageBox.Query("Delete movie", "Are you sure?", "No", "Yes");  if (index == 1)  {  this.deleted = true;  this.updated = false;  Application.RequestStop();  }  }  public Movie GetMovie()  {  return this.movie;  }  } |