

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ І ТЕХНОЛОГІЙ

КУРСОВИЙ ПРОЕКТ

з дисципліни: «Об'єктно-орієнтоване програмування»

на тему: «Розробка програмного додатку «Відеотека»

Студента 2 курсу групи 2ПР2

Спеціальності 121 Інженерія
програмного забезпечення

Мелешка Олександра Вікторовича

Керівник: к.т.н., доцент

Ляшенко О. М.

Національна шкала _____

Кількість балів: _____

Оцінка: ECTS _____

Члени Комісії

(підпис)

(прізвище та ініціали)

(підпис)

(прізвище та ініціали)

(підпис)

(прізвище та ініціали)

ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Кафедра	Програмних засобів і технологій				
Дисципліна	Об'єктно-орієнтоване програмування				
Спеціальність	121 «Інженерія програмного забезпечення»				
Курс	2	Група	2ПР2	Семестр	3

ЗАТВЕРДЖУЮ
завідувач кафедри ПЗіТ
_____ Шерстюк В.Г.

ЗАВДАННЯ

на курсову роботу студента

1. Тема роботи Розробка програмного додатку «Відеотека» мовами Java/SQL
2. Строк здачі студентом закінченого проекту (роботи) 23 грудня 2022 р.
3. Вхідні дані до проекту (роботи) У БД зберігається інформація про домашню відеотеку: фільми, актори, режисери. Для фільмів необхідно зберігати: назву, імена акторів, дату виходу, країну де випущено фільм. Для акторів та режисерів необхідно зберігати: ПІБ, дату народження.
4. Зміст пояснювальної записки (перелік питань, що необхідно розібрати)
 - 1) Концептуальне, логічне і фізичне проектування БД «Відеотека»;
 - 2) Нормалізація таблиць БД «Відеотека» до 3НФ або НФ Бойса-Кодда;
 - 3) Створення БД «Відеотека» за допомогою MySQL Workbench 8.0 CE;
 - 4) Створення класів для організації і налаштування з'єднання з БД;
 - 5) Створення класів для формування запитів до БД.
5. Перелік графічного матеріалу (з точною вказівкою обов'язкових креслень)
 - 1) ER-діаграма БД «Відеотека», діаграма класів програмного додатку;
 - 2) Креслення інтерфейсів програми.
6. Дата видачі завдання: 23 вересня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів курсового проекту (роботи)	Строк виконання етапів проекту	Примітки
1	Отримання завдання	21 – 23 вересня	
2	Пошук інформації	24 вересня – 10 жовтня	
3	Постановка задачі	11 – 18 жовтня	
4	Проектування БД	19 – 31 жовтня	
5	Складання програми	1 – 21 листопада	
6	Складання пояснювальної записки	22 листопада – 22 грудня	
7	Захист курсового проекту	23 грудня	

Студент Мелешко Олександр Вікторович

Керівник Ляшенко Олена Миколаївна

« ____ » _____ 2022 р.

АНОТАЦІЯ

Пояснювальна записка складається зі вступу, 4 розділів, висновку, списку використаних джерел і 3 додатків.

У вступі обґрунтовується актуальність обраної теми, мета, задачі, теоретична і практична бази курсового проекту.

Перший розділ «Аналіз предметної області. Постановка задачі» містить теоретичні аспекти об'єктно-орієнтованого програмування, особливості мов Java і SQL при розробці програмних додатків для роботи з реляційними базами даних. Визначено постановку задачі.

У другому розділі «Розробка та обґрунтування технічного завдання» викладено підставу для розробки, мету розробки, вимоги до програми, вимоги до функціональних характеристик, організацію вхідних і вихідних даних, критерії ефективності та якості програми, вимоги до надійності, умови експлуатації, вимоги до складу і параметрів технічних засобів, стадії та етапи розробки, моделі життєвого циклу, порядок контролю та приймання проекту.

У третьому розділі «Проектування системи» розповідається про етапи проектування БД «Відеотека» і системи програмного додатку.

Четвертий розділ «Технічна реалізація та впровадження програмного продукту» містить обґрунтування вибору програмних засобів, архітектуру розроблюваного додатку, алгоритми роботи програмного продукту, логічну структуру, інструкцію користувача, результати тестування програмного продукту.

У висновку підсумовується пророблена робота і внесок студента в проект.

У додатках наведено скріншоти роботи програми і вихідний код, розроблений в рамках даного курсового проекту.

Ключові слова: об'єктно-орієнтоване програмування, Java, бази даних.

Робота складається з 52 сторінок, містить 8 таблиць, 18 рисунків і 3 додатки.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ.....	6
ВСТУП.....	7
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАДАЧІ.....	9
1.1 Теоретичні аспекти об'єктно-орієнтованого програмування	9
1.2 Java як об'єктно-орієнтована мова програмування високого рівня	11
1.2.1 Історичні відомості.....	11
1.2.2 Особливості мови Java	11
1.3 Особливості мови SQL.....	13
1.4 Java і SQL. Підключення до бази даних.....	14
1.5 Постановка задачі.....	15
1.6 Висновок до розділу 1	16
2 РОЗРОБКА ТА ОБҐРУНТУВАННЯ ТЕХНІЧНОГО ЗАВДАННЯ	17
2.1 Технічне завдання на розробку програмного додатку «Відеотека»	17
2.2 Висновок до розділу 2	20
3 ПРОЕКТУВАННЯ СИСТЕМИ	21
3.1 Проектування БД «video_library»	21
3.2 Проектування системи програмного додатку. Діаграма класів.....	27
3.3 Висновок до розділу 3	30
4 ТЕХНІЧНА РЕАЛІЗАЦІЯ І ВПРОВАДЖЕННЯ ПРОГРАМНОГО ПРОДУКТУ	31
4.1 Обґрунтування вибору програмних засобів.....	31
4.2 Архітектура проекту.....	32
4.3 Алгоритми роботи програмного продукту	33
4.4 Логічна структура.....	34
4.5 Інструкція користувача	34
4.6 Тестування програмного продукту.....	37
4.7 Висновок до розділу 4	38
ВИСНОВОК	39
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	40
Додаток А. Екранні форми програми	41
Додаток Б. Вихідний код програми	44
Додаток В. SQL запити на створення і наповнення тестовими даними БД.....	50

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

а) англ.

API — Application Programming Interface (прикладний інтерфейс)
ER — Entity-Relationship model (модель «сутність-зв'язок»)
EXE — EXEcutable (виконуваний файл)
IDE — Integrated Development Environment (інтегроване середовище розробки)
JAR — Java ARchive (Java-архів)
JDBC — Java DataBase Connectivity (з'єднання з базами даних на Java)
JDK — Java Development Kit (комплект для розробки на Java)
JRE — Java Runtime Environment (середовище виконання Java)
JVM — Java Virtual Machine (віртуальна машина Java)
SQL — Structured Query Language (структурована мова запитів)
UML — Unified Modeling Language (уніфікована мова моделювання)

б) укр.

БД — база даних
НФ — нормальна форма
НФБК — нормальна форма Бойса-Кодда
ООП — об'єктно-орієнтоване програмування
ПЗ — програмне забезпечення
СКБД — система керування базами даних

ВСТУП

Більшість сучасних додатків у своїй роботі використовують бази даних. Це зумовлено тим, що часто виникає необхідність зберігати і опрацьовувати значні обсяги інформації. Звичайно, можна зберігати всі дані в окремі текстові файли, але тоді виникнуть проблеми з постійно зростаючим обсягом пам'яті, що займає додаток на пристрої, і занадто ускладняться або стануть взагалі неможливими операції обробки даних, наприклад, оновлення чи видалення пов'язаних записів.

Тому для вирішення таких завдань застосовують бази даних, як набір впорядкованої та структурованої інформації, що зберігається і обробляється обчислювальною системою (комп'ютером). Якщо БД (база даних) – це власне дані, що зберігаються відповідно до певної моделі даних, то система керування базами даних (СКБД) – програмне забезпечення, що керує даними, які зберігаються в БД.

Іноді виявляється зручним використання локального сервера для зберігання бази даних. Так можна створити БД, до якої матиме доступ лише власник комп'ютера, тобто програма працюватиме з даними, збереженими на локальному комп'ютері, а не на віддаленому сервері. Такий підхід припустимий, коли дані в БД призначені тільки для користувача програми і ніяк не пов'язані з даними інших користувачів. Наприклад, домашня відеотека може цілком використовувати локальну базу даних для зберігання в ній улюблених фільмів, акторів і режисерів.

Актуальність обраної теми — дана тема актуальна, тому що переважна більшість програмних і веб-додатків не можуть працювати належним чином без взаємодії з базами даних, у яких можуть зберігатися надзвичайно важливі дані про користувачів системи, категорії товарів і послуг, історію транзакцій тощо. При цьому з точки зору надійності програма не може керувати даними самостійно, оскільки через одну або кілька виключних ситуацій усі дані можуть бути втраченими без можливості їхнього відновлення. Усі свої дії програма повинна згаджувати із СКБД, надсилаючи відповідні запити до баз даних і зберігаючи внесені зміни. Так, при виникненні виключної ситуації і некоректного завершення

програми, усі дії будуть відновлені або повернені до початкового стану (точки збереження), що більше властиве транзакціям, наприклад, переказу коштів.

Об'єкт дослідження — програмні додатки, які працюють з БД.

Предмет дослідження — програмні додатки на мові Java, які працюють з локальними БД за допомогою SQL запитів на прикладі домашньої відеотеки, де зберігається інформація про фільми, акторів і режисерів.

Мета роботи — розробити програмний додаток «Відеотека» мовою Java, який працює з локальною БД за допомогою SQL запитів.

Задачі курсового проекту:

- 1) Визначити особливості розробки Java додатків, які працюють з БД за допомогою запитів структурованої мови запитів SQL;
- 2) Спроекувати і нормалізувати БД для програмного додатку;
- 3) Розробити ряд класів для встановлення і налаштування з'єднання з БД;
- 4) Розробити класи програми для модифікації й отримання даних з БД;
- 5) Розробити користувацький клас для взаємодії з програмним додатком;
- 6) Протестувати програмний продукт на всій множині можливих значень;
- 7) Виконати впровадження продукту – створити виконуваний файл (*.exe).

Теоретичною базою курсового проекту стали праці експертів в області програмування, які займалися Java розробкою (Герберт Шилдт, Барі Берд, Джошуа Блох) і програмних письменників, які розглядали як і роботу SQL з мовою Java, так і мову запитів окремо (Поль Дюбуа, Джоел Мурах).

Практичною базою курсового проекту стали лабораторні заняття з дисциплін «Об'єктно-орієнтоване програмування» та «Бази даних», де було отримано базові навички розробки програмних додатків мовою Java і принципи роботи з реляційними базами даних, зокрема з SQL.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАДАЧІ

1.1 Теоретичні аспекти об'єктно-орієнтованого програмування

З розвитком обчислювальної техніки поступово ускладнювалися програми, тому виникали нові підходи для розробки складних систем. Якщо структурний підхід до програмування ще можна було застосовувати для написання помірно складних програм, то, коли програмний код досягав певної довжини, даний підхід виявлявся неідеальним. Як рішення було розроблено принципи об'єктно-орієнтованого програмування (ООП) – новий підхід до розробки програм.

ООП дозволяє декомпонувати (розкласти) проблему на окремі частини – так з'являється можливість працювати з набагато більшими програмами. Кожен компонент перетворюється на самостійний об'єкт зі своїм кодом і даними.

Об'єкт – це замкнута незалежна сутність, що взаємодіє із зовнішнім світом через певний інтерфейс за допомогою прийнятих повідомлень.

Об'єкти володіють певним набором властивостей, методів і здатністю реагувати на події. На відміну від процедурного програмування, де порядок виконання операторів визначається за порядком їхнього розташування і командами керування, в ООП порядок виконання процедур і функцій визначається за подіями.

Далі об'єкти з однаковими властивостями і поведінкою об'єднуються в класи. З цього випливає, що програма написана на об'єктно-орієнтованій мові програмування, виглядає як сукупність описів класів. Класи, у свою чергу, виглядають як описи властивостей і поведінки їхніх складових – об'єктів. Властивості представляються іншими, більш простими об'єктами. Поведінка описується об'єктами, що обмінюються повідомленнями.

Звідси об'єкт – реальна поіменована сутність, яка володіє певними властивостями і виявляє свою поведінку.

Застосувавши поняття об'єкту і класу до об'єктно-орієнтованих мов програмування, отримуємо конкретніші визначення:

Об'єкт – поіменований набір даних (полів об'єкту), які фізично знаходяться в пам'яті комп'ютера, і методів, що мають до них (полів) доступ. Ім'я використовується для доступу до полів і методів, які складають об'єкт, тобто до об'єктів у програмі необхідно звертатися за їхнім ім'ям. В окремих випадках, об'єкт може не містити в собі полів або методів, а також не мати імені.

Кожен об'єкт належить до певного класу. Клас містить опис даних і операцій над цими даними. У класі дається узагальнений опис певного набору споріднених, реально існуючих об'єктів. Отже, об'єкт – це конкретний екземпляр класу.

У таблиці 1 наведено основні принципи ООП:

Таблиця 1 Основні принципи ООП

Принцип ООП	Коментар
Абстрагування даних	розподіл даних на важливі для вирішення задачі і несуттєві
Інкапсуляція (encapsulation)	об'єднання інтерфейсу і реалізації в один клас, а також приховання деталей реалізації (розмежування доступу)
Наслідування (inheritance)	об'єкт може успадковувати основні властивості іншого об'єкту і додавати до них власні властивості і методи
Поліморфізм (polymorphism)	використання одного і того ж імені методу для вирішення двох або більше схожих, проте трохи різних завдань
«Пізнє зв'язування»	у процесі виконання програми визначає приналежність об'єкту до конкретного класу і виконує виклик методів, що належать до класу, об'єкт якого було використано

Слід зауважити, що під час виклику методу класу відбувається його пошук в самому класі. Якщо метод існує в поточному класі, він викликається. Інакше звернення відбувається до батьківського класу, і пошук методу, що викликається, відбувається в ньому. Якщо пошук не дає результатів, то він продовжується вгору по ієрархічному дереву. Кінцевою точкою (коренем) є верхній клас за ієрархією.

1.2 Java як об'єктно-орієнтована мова програмування високого рівня

1.2.1 Історичні відомості

Мова Java була розроблена компанією Sun Microsystems на початку 90-х років XX століття. Провідну роль у розробці мови програмування відіграв канадський інженер Джеймс Гослінг. На ранніх етапах розробки Java мала іншу назву – Oak (з англ. Дуб). Через певний час назву змінили на Java в честь однойменного сорту кави. Звідси і походить логотип, зображений на рисунку 1:



Рисунок 1 Сучасний логотип мови Java

Джеймс Гослінг разом зі своїми однодумцями хотіли створити мову програмування із С-подібним синтаксисом. У той же час вона повинна бути більш простою і зрозумілою у порівнянні із С/С++. Розробники планували використовувати Java для програмування побутової електроніки, однак практично відразу після першого випуску (версія 1.0) у 1995 році, нову мову стали використовувати розробники серверного і клієнтського програмного забезпечення.

У 2010 році компанію Sun Microsystems купила американська корпорація Oracle – найбільший у світі розробник програмного забезпечення (ПЗ) для організацій, великий постачальник серверного обладнання. Головний розробник Java, Джеймс Гослінг, покинув свою компанію і перейшов працювати в транснаціональну корпорацію Google, звідки незабаром теж звільнився.

1.2.2 Особливості мови Java

Java – це мова програмування загального призначення, що належить до об'єктно-орієнтованих мов програмування із сильною типізацією.

Розробники реалізували принцип WORA: «Write once, run anywhere!». Дослівно перекладається: «Пиши один раз – запускай скрізь!». Це означає, що програмний додаток, написаний мовою Java, можна запускати на будь-якій платформі (кросплатформність), якщо на ній встановлене середовище виконання Java (JRE, Java Runtime Environment).

Дана задача вирішується за допомогою компіляції написаного Java коду в байт-код. Отриманий формат виконує JVM (віртуальна машина Java). JVM – частина середовища виконання Java (JRE). При цьому, віртуальна машина не залежить від платформи.

У Java реалізовано механізм керування пам'яттю, який називається збирач сміття (garbage collector). Суть полягає в тому, що розробник певної програми створює об'єкти, а JRE за допомогою збирача сміття очищує пам'ять, коли об'єкти припиняють використовуватись. Якщо розглядати докладніше, то існує таке поняття як циклічне сміття. У середині циклу на всі об'єкти є посилання, однак збирач сміття в Java видаляє ці посилання, коли об'єкти більше не можуть використовуватись із програми.

Нижче наведені синтаксичні особливості мови Java:

- Чутливість до регістру. Ідентифікатори Client і client у Java виглядають як зовсім різні сутності;
- Для іменування методів використовується lowerCamelCase. Якщо ім'я методу складається з одного слова, це слово має починатися з малої літери. Наприклад, method або getClientName;
- Для іменування класів використовується UpperCamelCase. Якщо ім'я класу складається з одного слова, це слово має починатися з великої літери. Наприклад, Client або SuperClient;
- Ім'я файлу програми повинне точно збігатися з ім'ям класу з урахуванням чутливості до регістру. Наприклад, якщо клас називається SuperClient, то файл

цього класу повинен називатися SuperClient.java. Якщо ж клас називається просто Client, то і файл тоді називається Client.java.

- Як правило, для іменування констант використовують UPPERCASE. Усі літери в імені константи записуються у верхньому регістрі. Наприклад, CONSTANT;
- Ідентифікатори завжди починаються з літери (A – Z, a – z), символу \$ або нижнього підкреслювання _. Наприклад, name, NAME, \$name, _name.

1.3 Особливості мови SQL

SQL (скорочення від англ. Structured Query Language) – це структурована мова запитів, яку застосовують для роботи з БД, структурованих певним чином. Головні задачі SQL – складати запити так, щоб знаходити серед великого об’єму інформації потрібну для конкретних цілей, сортувати її, структурувати і подавати для відображення в найбільш простому і зрозумілому вигляді.

Перший прототип мови SQL презентувала в 1979 році компанія-розробник Oracle. Спочатку це був звичайний інструмент для отримання потрібних даних. З роками він ускладнювався, і тепер його застосовують в якості одного з основних інструментів для обробки даних. За допомогою SQL можна:

- збирати і зберігати дані у вигляді таблиць;
- змінювати їхній вміст і структуру;
- об’єднувати дані і виконувати обчислення;
- захищати і розподіляти доступ.

Особливості мови SQL:

- SQL – мова запитів, а не програмування. Її використовують як доповнення до Python, Java, JavaScript, C++, але тільки для роботи з БД. Написати на цій мові повноцінний сайт або додаток неможливо.
- Чітка і зрозуміла структура. Це робить мову SQL для роботи з даними відносно простою для вивчення.

- Універсальність. Існують єдині стандарти побудови запитів для будь-яких БД, які дозволяють обробляти значні об'єми інформації.
- Спільний доступ. SQL дозволяє створювати інтерактивні запити. Це означає, що можна отримувати потрібні дані онлайн і приймати рішення на їхній основі.
- Керування доступом. За допомогою SQL можна надати, заборонити або обмежити доступ до даних для різних груп користувачів, а також надати їм певний набір функцій: перегляд, редагування, створення, видалення тощо. Це захищає БД від несанкціонованого доступу або неузгоджених дій.

1.4 Java і SQL. Підключення до бази даних

Для зберігання даних можна використовувати різні СКБД : Oracle, MS SQL Server, MySQL, Postgres. Усі перелічені системи керування базами даних мають свої особливості, але їх об'єднує взаємодія зі сховищем даних за допомогою команд SQL. Щоб визначити єдиний механізм взаємодії з цими СКБД у Java ще починаючи з 1996 року було введено спеціальний прикладний інтерфейс API, який називається JDBC – Java DataBase Connectivity.

Тобто для того, щоб у програмному додатку на мові Java взаємодіяти з БД, необхідно використовувати функціональні можливості JDBC. Для роботи з JDBC в програмі Java достатньо підключити пакет `java.sql`.

Однак не всі системи керування БД можуть підтримуватися через JDBC. Для роботи з певною СКБД також необхідний спеціальний драйвер. Кожен розробник певної СКБД зазвичай надає свій драйвер для роботи з JDBC. Тобто, якщо потрібно працювати з MySQL, то необхідно використовувати спеціальний драйвер для роботи саме з MySQL. Як правило, більшість драйверів доступні у вільному доступі на офіційних сайтах відповідних СКБД. Зазвичай драйвер виглядає як JAR-файл, який після встановлення треба додати до програми як зовнішню бібліотеку.

Для взаємодії з БД через JDBC використовуються SQL запити. Для різних СКБД існують свої різновиди структурованої мови запитів. Наприклад, у MS SQL Server це T-SQL, в Oracle – PL/SQL, але в цілому вони не сильно відрізняються.

1.5 Постановка задачі

Розробити програмний додаток «Відеотека» мовою Java, що працює з локальною БД за допомогою SQL запитів, виконавши наступні завдання:

1) Спроекувати БД «Відеотека»:

- a) визначити три сутності: режисер, актор і фільм,
- b) побудувати концептуальну модель для цих сутностей,
- c) для сутностей режисер і актор визначити атрибути: повне ім'я, дата народження; для сутності фільм визначити атрибути: назва фільму, режисер, актори, країна, рік виходу на екран,
- d) побудувати логічну і фізичну моделі для цих сутностей,
- e) нормалізувати отриману БД до 3НФ або нормальної форми Бойса-Кодда;

2) Створити БД «Відеотека» за допомогою MySQL Workbench 8.0 CE:

- a) створити порожню БД з ім'ям `video_library`,
- b) на основі фізичної моделі і результатів нормалізації створити таблиці,
- c) заповнити таблиці початковими даними, необхідними для роботи з БД;

3) Створити однойменний програмний додаток для роботи з БД «Відеотека»:

- a) створити клас, де встановлюються дані для налаштування з'єднання з БД, Визначити захищені поля класу: локальний сервер, порт, ім'я бази даних (`video_library`), ім'я користувача, пароль,
- b) створити клас, що наслідує попередній, метод якого повертає з'єднання з БД,
- c) створити клас зі статичними `final`-полями, де визначити назви таблиць та їхні атрибути для розробленої бази даних,
- d) створити клас для виконання запитів на вибірку даних з БД,
- e) створити клас для виконання запитів на модифікацію даних з БД,
- f) створити користувацький клас для взаємодії з програмним додатком;

- 4) Протестувати програмний продукт на всій множині можливих значень;
- 5) Виконати впровадження програмного продукту:
 - a) створити виконуваний JAR-файл,
 - b) завантажити і встановити програму «Launch 4j Executable Wrapper»,
 - c) за допомогою встановленої програми конвертувати JAR-файл в EXE-файл.

1.6 Висновок до розділу 1

У даному розділі було розглянуто теоретичні аспекти об'єктно-орієнтованого програмування, історичні відомості й особливості синтаксису Java, особливості й характеристики мови структурованих запитів (SQL), підключення Java-додатку до реляційної бази даних. Докладно визначено постановку задачі курсового проекту.

2 РОЗРОБКА ТА ОБҐРУНТУВАННЯ ТЕХНІЧНОГО ЗАВДАННЯ

2.1 Технічне завдання на розробку програмного додатку «Відеотека»

Підстава для розробки. Підставою для виконання курсового проекту є завдання, затверджене на кафедрі програмних засобів і технологій.

Мета і призначення розробки. Метою розробки даного продукту є розробка програмного додатку «Відеотека», який може використовуватись як домашня відеотека, що містить дані про фільми, режисерів і акторів.

Вимоги до програми

Користувацькі інтерфейси. Для користувача повинно бути виведено меню програми з переліком існуючих функцій для взаємодії з БД (операції з отримання і зміни даних) та можливість обрати певний пункт меню або закрити програму.

Апаратні інтерфейси. Щоб мати можливість використовувати даний продукт, необхідно мати клавіатуру, мишу, дисплей.

Програмні інтерфейси. Даний проект розробляється на об'єктно-орієнтованій мові програмування високого рівня Java версії 1.8.0_361 з JDK версії 19.0.1. Для зв'язку з локальною БД (DB MySQL) використовується драйвер MySQL Connector J версії 8.0.32. У зв'язку з тим, що даний проект написаний мовою Java, він може працювати на будь-якій платформі, де встановлений комплект розробки на Java (JDK) і встановлена СКБД MySQL з наявною там БД «video_library».

Комунікаційні інтерфейси. Комунікаційними інтерфейсами між користувачем і програмним продуктом є програмний додаток. Система спілкується з СКБД через спеціальний драйвер завдяки SQL запитам, де програмний додаток (клієнтська програма) надсилає запит до СКБД. У свою чергу СКБД обробляє запит і повертає результат клієнту. Програма-клієнт обробляє отриманий результат і виводить інформацію користувачеві.

Вимоги до функціональних характеристик

- REQ-1: Знаходити всі фільми, що вийшли на екран поточного та минулого року;
- REQ-2: Виводити інформацію про акторів, які знімалися в заданому фільмі;
- REQ-3: Виводити інформацію про акторів, які знімалися що найменше ніж у заданій кількості фільмів;
- REQ-4: Виводити інформацію про акторів, які були режисерами хоча б одного з фільмів;
- REQ-5: Видаляти всі фільми, дата виходу яких була більшою за задану кількість років.

Організація вхідних і вихідних даних. Вхідними даними є команди користувача на отримання і модифікацію даних, дані необхідні для формування запитів (назва фільму, рік виходу на екран, кількість зіграних ролей). Вихідними даними є інформація, отримана завдяки запитам до БД.

Критерії ефективності та якості програми. Програма має високу ефективність, оскільки споживає незначну частину ресурсів системи. Зокрема, на диску вона може займати не більше 10Мб вільного простору, включаючи усі необхідні компоненти.

Атрибути якості для даного програмного продукту є завершеність і цілісність усієї системи, здатність самостійно і коректно відновлювати роботу після збоїв, відмовостійкість.

Вимоги до надійності. Унаслідок неправильних дій користувача можливе некоректне завершення програми, яке не має вплинути на дані, збережені в БД. Тобто внаслідок неправильно введених даних неможливо пошкодити дані в БД завдяки ізолюваності компонентів системи: неправильні дані не можуть вплинути на формування запиту, доступ до якого обмежується всередині відповідного класу.

Умови експлуатації. Забороняється вносити зміни до компонентів програмного продукту: видаляти, перейменовувати файли й директорії, видаляти бази даних. У разі зміни певних параметрів: ім'я, пароль користувача MySQL, порт, назва БД, назви таблиць, атрибутів, відповідні зміни необхідно вносити у файли конфігурації, призначені для налаштування з'єднання і роботи з базою даних.

Вимоги до складу і параметрів технічних засобів. Для функціонування програмного додатку необхідна 64-х розрядна архітектура, сучасна підтримувана операційна система платформи Windows, macOS або Linux і встановлене наступне ПЗ: JDK (Java Development Kit, комплект розробки на Java версії 17 і вище) і MySQL Community Edition з усіма його складовими (обов'язковою є наявність MySQL Workbench) версії 8.0 і вище.

Стадії та етапи розробки

- 1) Аналіз і формування вимог;
- 2) Проектування системи;
- 3) Реалізація;
- 4) Тестування;
- 5) Впровадження програмного продукту;
- 6) Експлуатація і супровід ПЗ.

Моделі життєвого циклу

Каскадна модель. Запропонована у 1970 році Вінстоном Ройсом. Передбачає послідовне виконання всіх етапів проекту в суворо визначеному порядку. Перехід на наступний етап можливий лише за умови повного завершення робіт на попередньому. Вимоги, визначені на стадії формування вимог, документуються у вигляді технічного завдання і фіксуються на весь час розробки проекту. Кожна стадія завершується випуском повного комплекту документації, достатньої для того, щоб роботу над проектом могла продовжити вже інша команда розробників.

Перевагами даної моделі є повна і налагоджена документація на кожному етапі розробки, можливість визначити терміни і витрати на проект.

Недоліками є те, що перехід від однієї фази проекту до іншої передбачає повну завершеність попередньої. Однак неточність певної вимоги або некоректна її інтерпретація в результаті призводить до повернення до ранньої фази проекту. Так переробка всього проекту призводить до порушення термінів виконання і зростання витрат, а часто й до відмови від проекту в тій формі, у якій він першочергово був визначений. Таким чином каскадна модель для великих проектів не дієва, і може бути ефективно використана лиш для створення невеликих систем.

Порядок контролю та приймання проекту. Готовий проект перевіряється на відповідність вимогам, зазначеним у технічному завданні, і приймається керівником курсового проекту.

2.2 Висновок до розділу 2

У даному розділі було розроблено і обґрунтовано технічне завдання на розробку програмного додатку «Відеотека», а саме розглянуто і проаналізовано наступні пункти: підставу для розробки, мету і призначення розробки, вимоги до програми, вимоги до функціональних характеристик, організацію вхідних і вихідних даних, критерії ефективності та якості програми, вимоги до надійності, умови експлуатації, вимоги до складу і параметрів технічних засобів, стадії та етапи розробки, моделі життєвого циклу, порядок контролю та приймання проекту.

3 ПРОЕКТУВАННЯ СИСТЕМИ

3.1 Проектування БД «video_library»

Концептуальне проектування. Для побудови концептуальної моделі необхідно визначити наступні сутності: фільм, режисер і актор. За допомогою ER-діаграми (Entity-Relationship model, модель «сутність-зв'язок») зобразимо ці сутності і зв'язки між ними на рисунку 3.1:



Рисунок 3.1 Концептуальна модель для БД «video_library»

З побудованої діаграми бачимо, що між сутностями «режисер» і «фільм» існує зв'язок один до багатьох з обов'язковим класом приналежності для сутності «фільм». Дійсно, один режисер може знімати багато фільмів; режисер повинен зняти хоча б один фільм; фільм повинен мати одного режисера. Між сутностями «актор» і «фільм» існує зв'язок багато до багатьох з обов'язковим класом приналежності для обох сутностей. Тобто, один актор може зніматися в багатьох фільмах; в одному фільмі можуть зніматися багато акторів; актор повинен знятися хоча б в одному фільмі; у фільмі повинно бути не менше одного актора.

Логічне проектування. Для кожної з сутностей, визначених у концептуальній моделі для бази даних, додамо атрибути і позначимо індекси, як наведено у таблиці 3.1:

Таблиця 3.1 Атрибути й індекси таблиць БД для логічної моделі

Сутність	Індекс	Атрибут
director	PK	Director_ID
	U	Full_name
		Birthday
film	PK	Film_ID
	U	Film_name

Сутність	Індекс	Атрибут
film	FK	Director_ID
		Country
		_Year
actor	PK	Actor_ID
	U	Full_name
		Birthday

Таким чином отримуємо логічну модель, зображену на рисунку 3.2. На ER-діаграмі є позначення індексів: PK – Primary Key (первинний ключ), FK – Foreign Key (зовнішній ключ), U – Unique index (унікальний індекс).

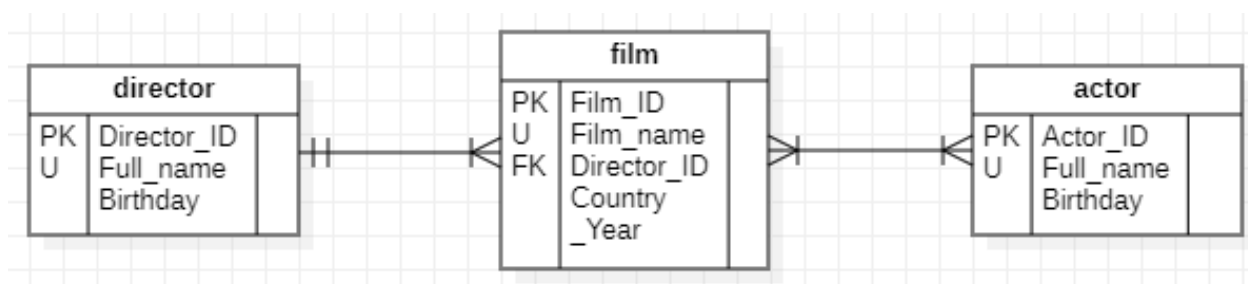


Рисунок 3.2 Логічна модель для БД «video_library»

Фізичне проектування. На основі логічної моделі можна побудувати фізичну модель розроблюваної БД. Для цього слід додати тип даних до атрибутів для кожної з сутностей і правильно сформулювати відношення.

Для формування відношень уважно аналізуємо кожен зв'язок.

Зв'язок між сутностями «режисер» і «фільм» раніше було визначено як один до багатьох з обов'язковим класом приналежності для сутності «фільм». За правилами формування відношень, достатньо сформулювати 2 відношення, де кожне відношення відповідає сутності. Первинними ключами цих відношень стануть первинні ключі цих сутностей. Окрім цього, до відношення, що відповідає сутності «фільм» необхідно додати як атрибут (зовнішній ключ) ключ сутності «режисер», що й було зроблено при логічному проектуванні (див. рисунок 3.2, ст. 22).

Зв'язок між сутностями «актор» і «фільм» раніше було визначено як багато до багатьох з обов'язковим класом приналежності для обох сутностей. За правилами формування відношень, незалежно від класу приналежності сутностей (обов'язковому чи ні) достатньо сформувати 3 відношення. Два з них відповідатимуть кожній із сутностей, ключі яких будуть первинними ключами для цих відношень. Третє відношення пов'язує перші два, а його ключ є складеним, що об'єднує ключові атрибути зв'язаних відношень. Отже, потрібно відокремити фільми від акторів, а потім створити відношення «фільми-актори», де пов'язати ці відношення між собою за допомогою складеного зовнішнього ключа.

Фізичну модель наведено на рисунку 3.3:

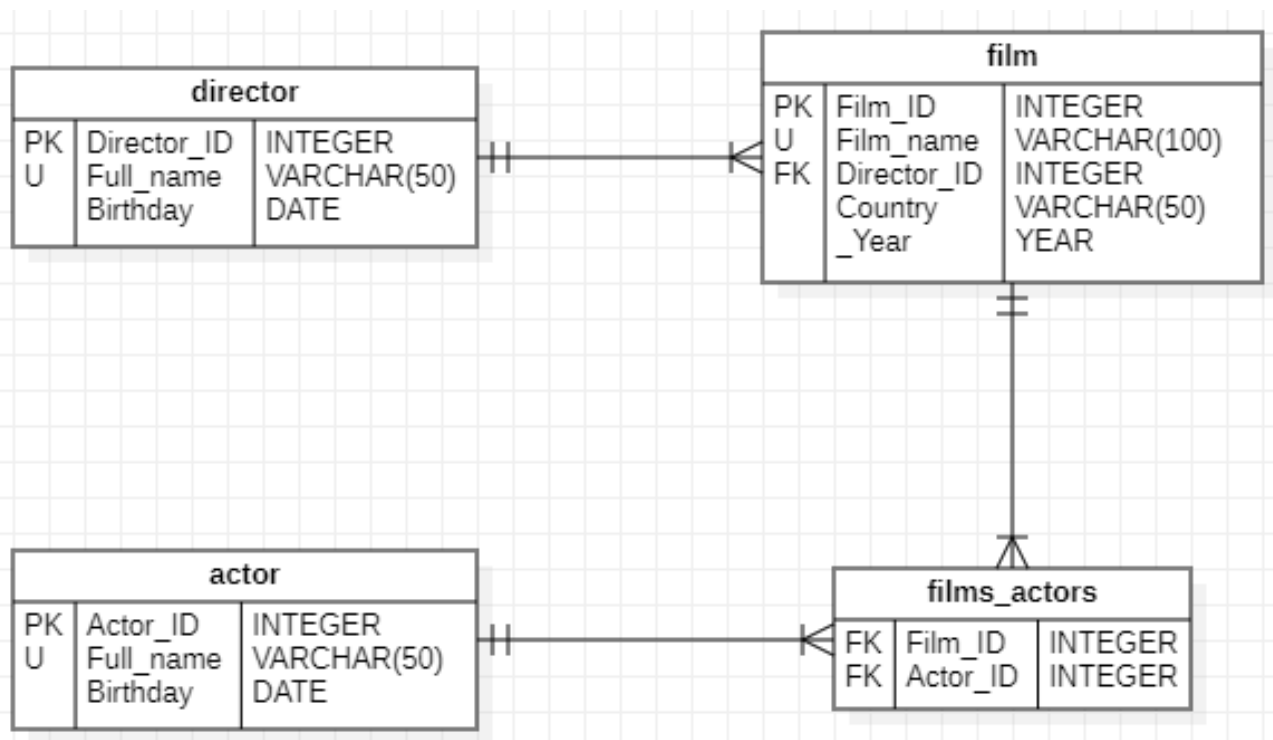


Рисунок 3.3 Фізична модель для БД «video_library»

Фізична модель є кінцевою при проектуванні бази даних. На її основі створюються запити на створення БД і таблиць до неї. У додатку В.1 наведено SQL запити на створення БД «video_library», а в додатку В.2 – запити на наповнення створеної бази даних початковими даними за допомогою СКБД MySQL.

Нормалізація бази даних. Нормалізація – це вирішення проблем із збитковістю даних, яка часто призводить до збільшення простору, зайнятого на диску, та аномаліям, пов'язаним з додаванням, видаленням і оновленням даних.

Як правило, існують нормальні форми, у яких можуть перебувати таблиці певної бази даних. Зазвичай виділяють 1-5 нормальні форми і нормальну форму Бойса-Кодда. Також існують й інші нормальні форми: домено-ключова, 6 і навіть нульова нормальні форми. Нульова нормальна форма – умовне поняття, яке означає, що таблиця не відповідає першій нормальній формі.

У концепції нормальних форм присутній принцип підмножин, де кожна наступна нормальна форма має виконувати вимоги попередньої. Так, щоб таблиця перебувала в четвертій нормальній формі, вона має перебувати в першій, другій, третій НФ і нормальній формі Бойса-Кодда, яку ще називають 3-ю посиленою НФ.

Відповідно простежується наступна залежність: чим вище нормальна форма (НФ), у якій перебуває конкретна таблиця, тим менше проблем зі збитковістю й аномаліями даних. Однак не має жодної необхідності приводити таблиці до 6 (максимальної) НФ, оскільки збільшення продуктивності БД відбувається лише до 3НФ або нормальної форми Бойса-Кодда, потім починається критичне зниження продуктивності бази даних. Виходячи з цього, працювати з БД, нормалізованою до 6НФ, буде дуже незручно й практично неможливо.

Отже, для продуктивної роботи з БД і відсутністю проблем зі збитковістю й аномаліями даних, достатньо нормалізувати базу даних до 3НФ або за необхідності до НФ Бойса-Кодда. Тобто спочатку необхідно переконатися, що таблиці виконують вимоги 1-2НФ, а лише потім перевірити відповідність вимогам 3НФ. Для НФ Бойса-Кодда необхідна відповідність ще й вимогам 3НФ.

Приведемо вихідні таблиці до БД «video_library», розробленої на основі фізичної моделі (див. рисунок 3.3) – таблиці 3.2-3.5 відповідно, ст. 25-26:

Таблиця 3.2 Режисери

Director_ID	Full_name	Birthday
1	Mark Cooper	1956-05-12
2	Chris Garik	1968-04-28
3	Mike Dandy	1971-12-10
4	Lisa Kraft	1973-09-15
5	Silver McArthur	1960-02-01
6	John Smith	1977-07-25

Таблиця 3.3 Фільми

Film_ID	Film_name	Director_ID	Country	_Year
1	The_old_man	1	Canada	2008
2	The_sea	2	USA	2010
3	The_red_fox	3	Great Britain	2011
4	What_is_that?	4	USA	2013
5	The_secret_of_the_black_box	5	USA	2015
6	The_crime_is_solved!	4	USA	2018
7	I_love_you,_Jennifer!	3	Great Britain	2020
8	It*s_my_life	6	Australia	2022
9	Penguins_don*t_fly!	4	USA	2023

Таблиця 3.4 Актори

Actor_ID	Full_name	Birthday
1	Bob Ruber	1950-10-05
2	Michelle Andrade	1990-01-07
3	George Schwartz	1947-03-17
4	Lucy Chado	1968-05-26
5	Linda Rowling	1972-12-31
6	John Smith	1977-07-25
7	Mary Star	1988-08-19

Продовження таблиці 3.4

Actor_ID	Full_name	Birthday
8	Lisa Kraft	1973-09-15
9	William Bertie	1970-04-29
10	Christie Pop	1981-10-16
11	Robert Ricci	1966-07-10
12	Chris Edison	1976-11-28
13	Bill Krumen	1977-08-05
14	Mike Dandy	1971-12-10
15	Kate Scroll	1983-01-19
16	Steve Gretchen	1996-03-30
17	Peter Parker	1985-04-24
18	Claude Van Damme	1972-06-12
19	Spike Person	1971-08-18
20	Roberto Montini	1982-12-15
21	Amelie King	1992-05-22
22	Chris Garik	1968-04-28

Таблиця 3.5 Фільми-Актори

Film_ID	Actor_ID
1	1
1	2
1	3
1	4
2	5
2	6
2	7
2	8
2	9
3	10
3	11
3	1

Film_ID	Actor_ID
3	12
3	13
4	14
4	15
4	16
4	17
5	8
5	18
5	7
5	6
6	19
6	12

Film_ID	Actor_ID
6	4
7	12
7	2
7	20
7	5
8	8
8	14
8	7
8	1
9	21
9	13
9	22

Проаналізувавши наведені вище таблиці, можна дійти наступних висновків:

1. Кожна з таблиць знаходиться в 1НФ, оскільки виконується умова: для будь-якого допустимого значення відношення кожен його кортеж містить тільки одне значення для кожного з атрибутів;
2. У свою чергу кожна з таблиць знаходиться в 2НФ, оскільки вона знаходиться в 1НФ і виконується умова: кожен неключовий атрибут відношення функціонально повно залежить від потенційного ключа цього відношення;
3. У свою чергу кожна з таблиць знаходиться в 3НФ, оскільки вона знаходиться в 2НФ і виконується умова: у відношенні відсутні транзитивні функціональні залежності неключових атрибутів від ключових;
4. У свою чергу таблиця 3.5, ст. 26 знаходиться в НФ Бойса-Кодда, оскільки вона знаходиться в 3НФ і виконується умова: у відношенні відсутні будь-які залежності ключових атрибутів від неключових. Перевіряти таблиці на відповідність до НФБК має зміст, тільки у тому випадку, коли таблиця має складений потенційний ключ (таблиця 3.5), інакше можна вважати, що таблиця автоматично знаходиться в посиленій 3НФ (НФБК), (таблиці 3.2-3.4, ст. 25-26).

Отже, таблиці, розроблені за фізичною моделлю бази даних (див. рисунок 3.3, ст. 23) достатньо нормалізовані, а БД «video_library» має найвищу продуктивність, при цьому відсутня проблема збитковості й аномалій даних.

3.2 Проектування системи програмного додатку. Діаграма класів

Опираючись на принципи ООП, для створення системи додатку необхідно створити відповідні Java-класи. Спочатку слід виділити наступні типи класів:

1. Класи для роботи з БД:
 - 1.1. Клас для модифікації даних у БД,
 - 1.2. Клас для вибірки даних з БД,
 - 1.3. Класи для налаштування параметрів роботи з БД:
 - 1.3.1. Клас для налаштування параметрів підключення до БД,

1.3.2. Клас для задання назв таблиць та їхніх атрибутів у БД.

1.4. Класи для встановлення з'єднання з БД:

1.4.1. Клас, метод якого повертає з'єднання з БД.

2. Головний клас програми (так званий Main-клас, точка входу в програму).

Виходячи з даної ієрархії, цілком логічним і правильним є рішення розподілити усі ці класи за пакетами. Наприклад, головним пакетом, у який будуть вкладені усі визначені програмні класи, визначити пакет `com.companu`, тобто основний пакет програми. Далі в ньому створити головний клас (точку входу в програму) і пакет, у який будуть вкладені класи, пов'язані з базою даних. Так ми відокремимо головний клас від інших класів. Аналогічну структуру залишається створити всередині пакету для класів для взаємодії з базою даних. Таким чином вибудовується ієрархія, наведена вище.

Визначивши класи системи програмного додатку і вибудувавши певну ієрархічну структуру між ними, можна створити діаграму класів з пакетами.

Діаграма класів UML – це структурна діаграма UML (Unified Modeling Language, уніфікована мова моделювання), що відображає загальну структуру ієрархії класів системи, їхні кооперації, поля, методи і відношення між ними.



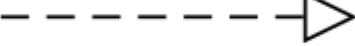
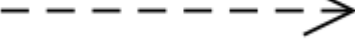


Для правильного задання видимості, перед полями і методами кожного класу записують позначення, наведені в таблиці 3.6:

Таблиця 3.6 Позначення видимості в UML

Позначення	Видимість	Коментар
-	private	видимий тільки у своєму класі
#	protected	видимий у всіх класах, що наслідують цей клас
+	public	видимий і доступний усюди
~	package	видимий тільки в межах свого пакету

В UML для зображення зв'язків між класами діаграми використовують різні типи стрілок, наведених в таблиці 3.7 на наступній сторінці:

Таблиця 3.7 Позначення зв'язків в UML

Позначення	Зв'язок
	Асоціація
	Наслідування
	Реалізація
	Залежність
	Агрегація
	Композиція

Діаграма класів програмного додатку «Відеотека» зображена на рисунку 3.4:

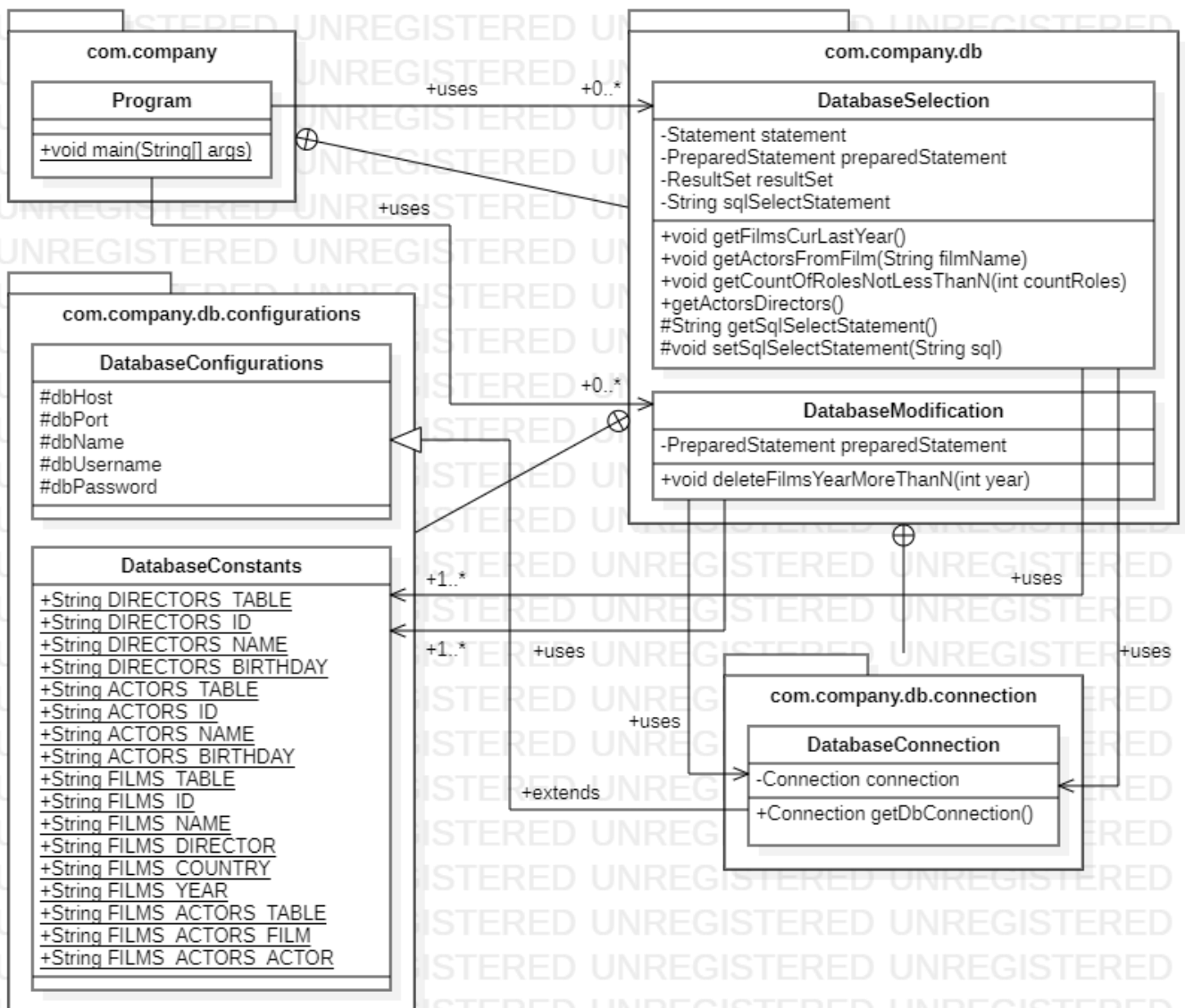


Рисунок 3.4 Діаграма класів системи програмного додатку «Відеотека»

3.3 Висновок до розділу 3

У даному розділі було виконано концептуальне, логічне і фізичне проектування бази даних «video_library», приведено таблиці 3НФ. Також виконане проектування системи програмного додатку, визначено позначення, що застосовуються в UML для позначення видимості полів, методів і зв'язків між класами (інтерфейсами), наведено діаграму класів програмного додатку.

4 ТЕХНІЧНА РЕАЛІЗАЦІЯ І ВПРОВАДЖЕННЯ ПРОГРАМНОГО ПРОДУКТУ

4.1 Обґрунтування вибору програмних засобів

Для створення і роботи з БД було обрано СКБД MySQL. Окрім універсальності і поширеності, дана система має важливі переваги у порівнянні з іншими системами керування базами даних. Найголовніші з них:

- 1) Простота і зручність у використанні. MySQL достатньо легко інсталюється, а наявність багатьох плагінів і допоміжних додатків спрощує роботу з БД;
- 2) Широкий функціонал. Система MySQL володіє практично всіма необхідними інструментами, що можуть знадобитися для реалізації різних проектів;
- 3) Безпека. Система першочергово була створена таким чином, що більшість вбудованих функцій безпеки в ній працюють за замовчуванням;
- 4) Масштабованість. Дана СКБД у рівній мірі легко може використовуватись для роботи і з невеликими, і зі значними об'ємами даних;
- 5) Швидкість взаємодії. Висока продуктивність системи забезпечується за рахунок спрощення певних, використовуваних у ній, стандартів.

Для розробки програмного додатку мовою Java було використано інтегроване середовище розробки (IDE) IntelliJ IDEA Community Edition. Серед переваг даної IDE можна виділити наступні:

- Функціональність. У середовищі IntelliJ IDEA можна розробляти додатки на Java й інших мовах програмування, які працюють на платформі JVM: Kotlin, Scala, Groovy. Передбачено підтримку мов і фреймворків для веб-розробки;
- Широкий вибір інструментів для роботи з кодом. Дана IDE надає інтелектуальну допомогу під час написання програмного коду. В області налагодження і тестування коду середовище також може запропонувати кілька цікавих рішень;
- Ергономічність. В IntelliJ IDEA легко налаштувати доступ до необхідних і часто використовуваних функцій. Додаткові плагіни, сполучення клавіш, налаштування інтерфейсу дозволяють зробити робочий процес максимально зручним для розробника. Наприклад, функція глобального пошуку, що

викликається подвійним натисканням клавіші Shift, допомагає знайти об'єкт у будь-якому компоненті проекту і за його межами;

- Комфортність. Низка опцій для зниження навантаження на очі, понад 100 різних тем оформлення, можливість синхронізувати колірну схему з налаштуваннями операційної системи, наявність спеціальних можливостей: читання з екрану, гнучке налаштування окремих елементів інтерфейсу.

4.2 Архітектура проекту

Архітектуру даного проекту можна визначити як консольний клієнт-серверний додаток, що взаємодіє з базою даних через прикладний програмний інтерфейс за допомогою SQL запитів.

З означення випливає, що консольний додаток – це додаток, що запускається з командного рядка, а клієнт-серверна програма – така програма, яка складається з програми-клієнта, з якою взаємодіє користувач системи, і програми-сервера (у даному випадку це СКБД MySQL).

Механізм взаємодії можна описати так: програма-клієнт отримує дані від користувача, обробляє їх і надсилає на сервер через спеціальний прикладний інтерфейс (канал зв'язку між клієнтом і сервером). Сервер обробляє отримані дані від програми-клієнта і повертає результат клієнтові. Клієнт, у свою чергу, обробляє отриманий результат і відображає кінцеву інформацію.

Вище було зазначено, що клієнт спілкується з сервером для взаємодії з БД за допомогою SQL запитів. Користувач при цьому не повинен задавати ці запити вручну, щоб отримати відповідь. У цьому полягає важливість такої архітектури, оскільки в протилежному випадку користувачу довелося б спілкуватися з БД напряму за допомогою MySQL Workbench або консольного клієнта MySQL.

Проте не можна сказати, що користувач взагалі не повинен взаємодіяти з БД напряму. Така необхідність виникатиме під час встановлення даного ПЗ, при

створені бази даних «video_library» і наповненні її необхідними даними за допомогою готових SQL запитів, що додаються до програмного продукту.

4.3 Алгоритми роботи програмного продукту

Основний алгоритм роботи додатку закладений у головному класі програми. Його можна описати за допомогою діаграм активності UML.

Діаграми активності (діяльності) – це діаграми UML, що описують динамічні аспекти поведінки системи у вигляді блок-схем, які відображають бізнес-процеси, логіку процедур і потоки робіт – перехід від однієї діяльності до іншої. Тобто, це певний алгоритм дій (логіка поведінки) системи або взаємодії кількох систем.

Приклад діаграми активності (діяльності) для відображення алгоритмів роботи програмного продукту наведено на рисунку 4.1:

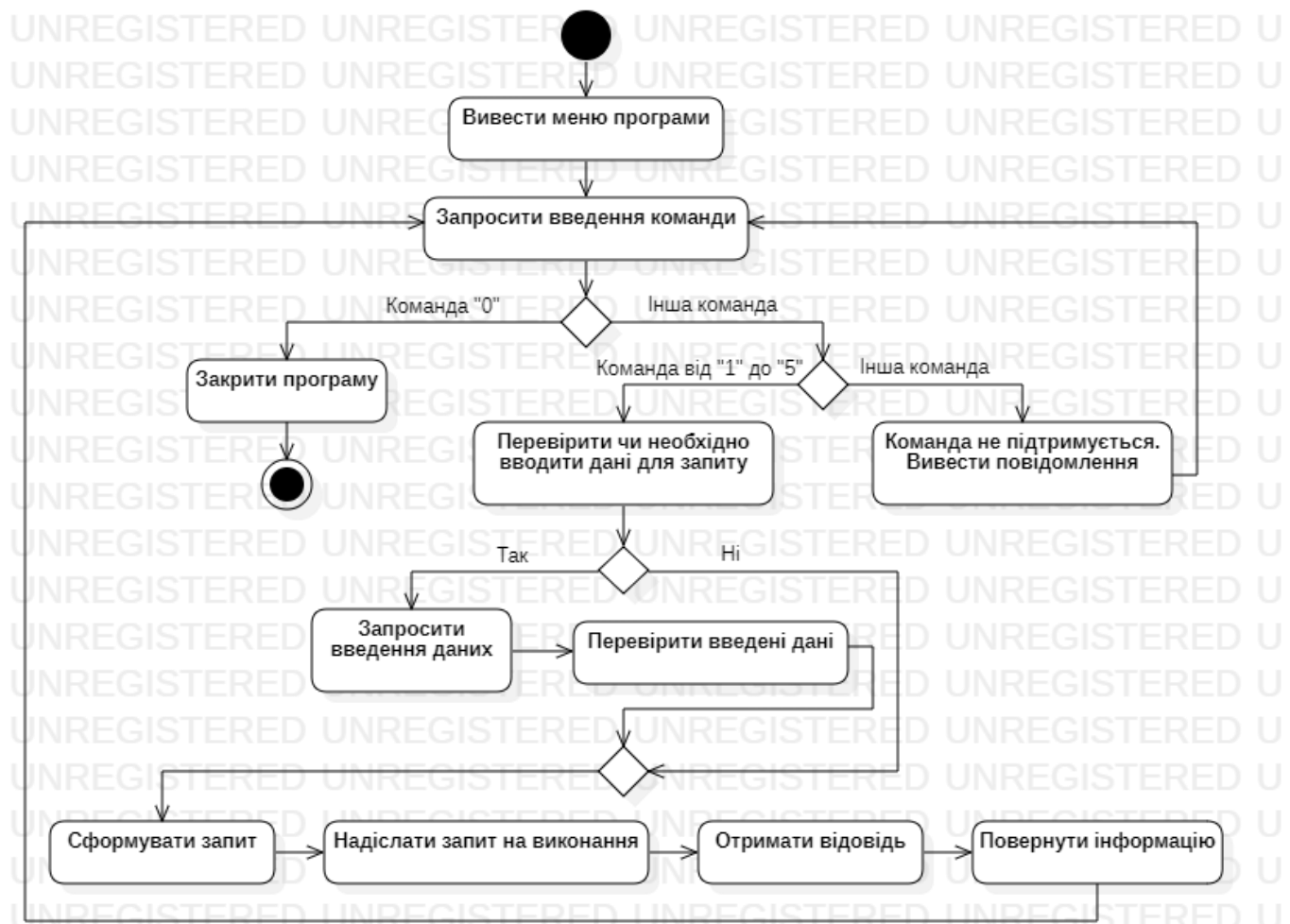


Рисунок 4.1 Діаграма активності, що відображає алгоритми роботи програмного продукту

4.4 Логічна структура

Логічну структуру програмного додатку також можна зобразити за допомогою діаграми активності з «доріжками» як на рисунку 4.2:

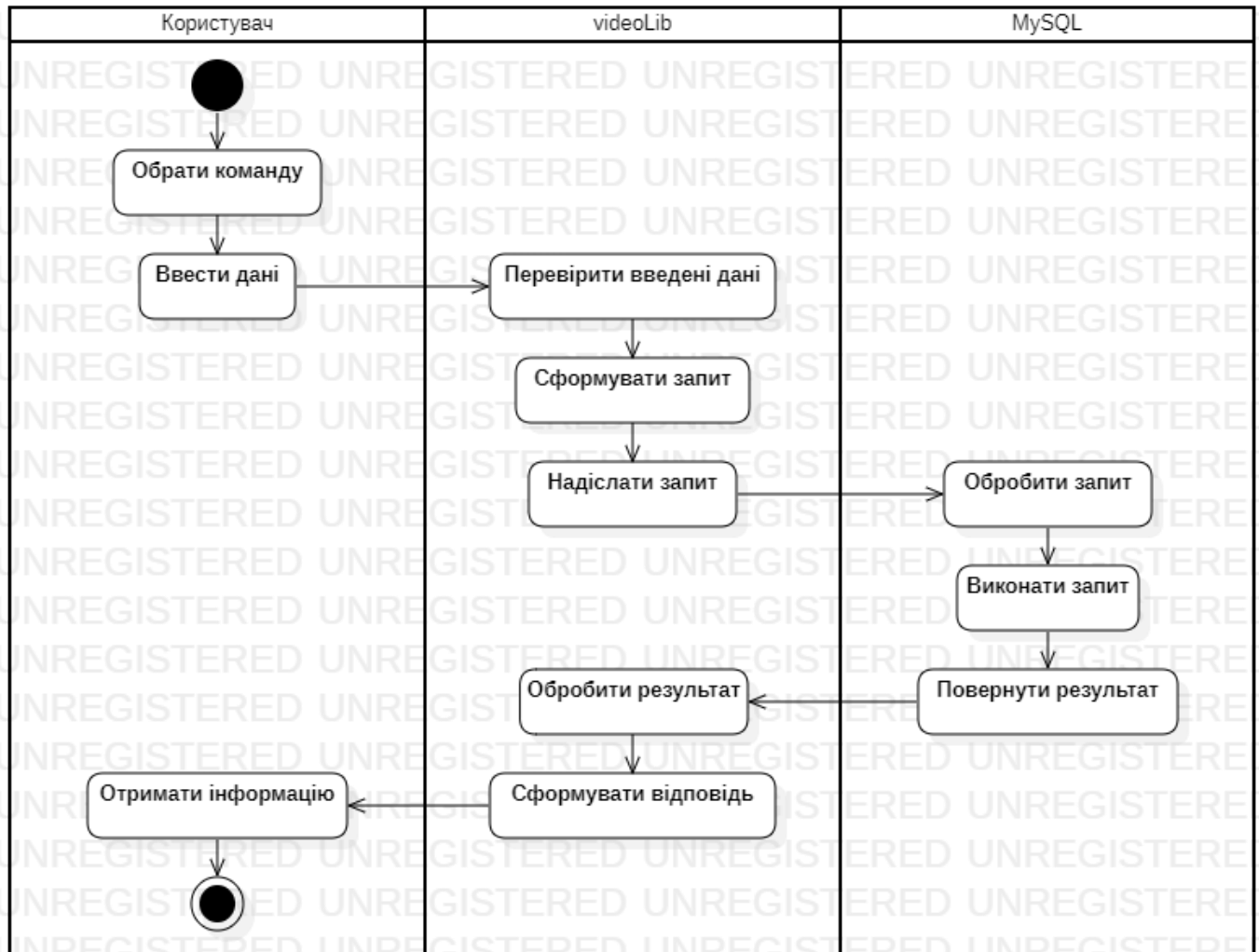


Рисунок 4.2 Діаграма активності, що відображає логіку роботи програмного продукту

4.5 Інструкція користувача

Перед початком інсталяції програмного продукту, переконайтеся, що ваш комп'ютер відповідає зазначеним системним вимогам: 64-х розрядна архітектура, операційна система платформи Windows, macOS або Linux.

Також перед початком використання необхідно встановити наступне ПЗ (безкоштовно):

- JDK версії 17.0.2 і вище. Офіційний сайт Java SE Development Kit (Oracle): <https://www.oracle.com/java/technologies/downloads/>
- MySQL версії 8.0.32 і вище. Офіційний сайт MySQL Installer for Windows (Oracle): <https://cdn.mysql.com//Downloads/MySQLInstaller/mysql-installer-community-8.0.32.0.msi> або <https://dev.mysql.com/downloads/>

Під час інсталяції програмних продуктів дотримуйтесь інструкцій розробника або переглядайте відповідну інформацію на офіційних сайтах. Також можна використовувати й інші інтернет-ресурси, наприклад відео-огляди чи статті.

Після успішної інсталяції допоміжного ПЗ, необхідно завантажити пакет videoLib у вигляді архіву і розпакувати його, наприклад, на системному носіїві.

Наступним кроком може бути створення ярлика на робочому столі для цієї програми. Щоб створити ярлик, який посилається на videoLib, необхідно відкрити розташування, куди ви зберегли папку з програмою, і перейти до videoLib — app. У даній папці можна знайти виконуваний файл «videoLib64.exe». Це і є шуканий виконуваний файл. Тепер необхідно скопіювати шлях до програми. На робочому столі створіть новий ярлик. Вставте скопійований шлях і допишіть в кінці «\videoLib64.exe» без лапок. Назва ярлика у вас першочергово буде збігатися з назвою виконуваного файлу. Ви можете за бажанням змінити її просто на videoLib.




Тепер перевірте, чи спрацював додаток при клацанні на ярлик. Якщо він не відкривається і система повідомляє про помилку типу «не знайдено файл, на який посилається цей ярлик», треба повторно виконати дії зі створення ярлика, тільки на цей раз уважніше прослідкувати за шляхом до програми. Нагадую, що він має закінчуватися на «\videoLib\app\videoLib64.exe» без лапок.

Якщо з додатком все гаразд і він запускається, вийдіть з нього натиснувши 0 і клавішу Enter. На даному моменті з ним можна працювати, але він ще не може взаємодіяти з базою даних. Щоб виправити це, необхідно переконатися, що ви встановили усі необхідні для роботи програмні продукти MySQL. Для нормальної роботи потрібно інсталювати хоча б MySQL Server і MySQL Workbench.

Ви можете доінстальювати певні програмні продукти за допомогою MySQL Installer for Windows або на офіційному сайті <https://dev.mysql.com/downloads/>.

Під час інсталяції можна налаштувати сервер для роботи з MySQL. Якщо ви цього не зробили, то необхідно відкрити програму MySQL Workbench і створити нове з'єднання, задати для нього пароль. Якщо з'єднання вже існує, просто відкрийте його, введіть пароль і дочекайтеся завантаження програми.

Якщо все пройшло успішно і програма MySQL Workbench завантажилася, можна переходити до створення бази даних «video_library», з якою буде взаємодіяти програмний додаток. Для цього необхідно відкрити готові SQL запити на створення і наповнення початковою інформацією бази даних.

Щоб відкрити ці SQL запити потрібно натиснути на кнопку . Далі необхідно обрати відповідні скрипти, що знаходяться в папці програмного додатку videoLib — sql. Спочатку відкрийте файл «createDB.sql». Для створення БД «video_library» необхідно запустити скрипт на виконання. У вкладці з відкритим SQL кодом натисніть на кнопку запуску . У вікні повідомлень, що знаходиться внизу має з'явитися успішне повідомлення зі значком .

Аналогічні дії потрібно виконати з файлом «insertDB.sql», який знаходиться у тій же папці, що й попередній скрипт.

Додаток готовий до роботи. Спробуйте відкрити програмний додаток videoLib і ввести, наприклад, команду «1» і натиснути клавішу Enter. Через секунду виведеться таблиця з тестовими даними, як на рисунку 4.3:

```
FILM_ID /// NAME /// DIRECTOR_ID /// COUNTRY /// YEAR
8 /// It*s_my_life /// 6 /// Australia /// 2022
9 /// Penguins_don*t_fly! /// 4 /// USA /// 2023
... the end of the table ...
```

Рисунок 4.3 Результат виконання команди «1» у додатку videoLib

Зверніть увагу! Після виведення подібної таблиці програма знову запрошує введення команди. Щоб закрити програму, треба ввести команду «0» і натиснути клавішу Enter. Робота з іншими командами відбувається аналогічним чином.

4.6 Тестування програмного продукту

Тестування програмного продукту відбувалося на всій множині можливих значень. Зокрема розглядалися такі випадки:

- введення команд типу -1, 10 (непідтримувані команди). Програма при цьому відповідала повідомленням про непідтримувані команди і кожного разу запрошувала повторне введення при виникненні подібних ситуацій;
- введення команд типу -10.10, Hello (невідповідність типів даних). Програма завершувала свою роботу, не даючи можливості пошкодити базу даних;
- введення даних для формування запитів, які не відповідають своєму типу даних (наприклад, рік виходу фільму на екрани = year12345). Програма реагувала на такі дії як потенційно небезпечні, що можуть спровокувати помилку на сервері;
- введення не дійсних даних, наприклад неіснуюча назва фільму. Програма виконувала запит, але за відсутності даних таблиця виводилася порожньою;
- вихід з програми за допомогою закриття вікна програми вручну. Такий варіант розглядається як примусове завершення програми або завершення програми не за сценарієм. Це не є критичною помилкою, але розглядається як виключення.

Також було розглянуто поведінку програми при недотриманні зазначених вимог:

- На комп'ютері не було встановлено JDK потрібної версії, або встановлено застаріле ПЗ. При відкритті програми нічого не відбувалося або консольне вікно миттєво закривалося, не відображаючи жодної інформації;
- На комп'ютері не встановлено СКБД MySQL. Програма при виконанні певної команди, де потрібно взаємодіяти з БД, замість таблиці виводить помилку типу «FATAL ERROR /// SQLException: ... ». Одна й та ж помилка відображається при виконанні різних команд, пов'язаних з роботою з базою даних;

- На комп'ютері встановлено СКБД MySQL, але не налаштовано з'єднання. Програма при виконанні команд видає помилку встановлення з'єднання з СКБД;
- На комп'ютері встановлено СКБД MySQL, налаштовано з'єднання, але не збігаються паролі. Програма при виконанні команд не може встановити з'єднання через заборону доступу до системи (пароль не підходить). Дана помилка зникає, якщо вручну задати новий пароль;
- На комп'ютері встановлено СКБД MySQL, налаштовано з'єднання і не виникає проблем з відповідністю пароля, але не створено БД «video_library». Програма при виконанні команд, видає помилку, оскільки не може підключитися до бази даних video_library, якої насправді не існує. Помилка зникає після створення і наповнення цієї БД за допомогою готових SQL запитів.

Результати тестування програмного продукту показали високу надійність системи і відмовостійкість при належному використанні. Також було встановлено, що більшість критичних збоїв виникають не через неправильні дії користувача, а загалом через відмінні від зазначених налаштувань з'єднання з СКБД MySQL та відсутності встановлених допоміжних програмних продуктів.

Даний продукт було визнано успішним і підтримано його супровід. Зокрема першу тестову версію у майбутніх випусках програми буде перероблено на графічний додаток з розширеним функціоналом і випущено програму-installer для встановлення додатку videoLib. Обов'язково буде додано інтерфейс для налаштування параметрів з'єднання з базою даних, додано можливість керування оновленнями, створено довідку і центр підтримки для користувачів.

4.7 Висновок до розділу 4

У даному розділі було обґрунтовано вибір програмних засобів для розробки проекту, детально розглянуто алгоритми роботи програмного продукту, його логічну структуру. Також було розроблено інструкцію користувача додатком videoLib, наведено результати тестування, визначено можливість супроводу ПЗ.

ВИСНОВОК

У першому розділі «Аналіз предметної області. Постановка задачі» було розглянуто теоретичні аспекти об'єктно-орієнтованого програмування, історичні відомості й особливості синтаксису Java, особливості й характеристики мови структурованих запитів (SQL), підключення Java-додатку до реляційної бази даних. Докладно визначено постановку задачі курсового проекту.

У другому розділі «Розробка та обґрунтування технічного завдання» було розроблено і обґрунтовано технічне завдання на розробку програмного додатку «Відеотека», а саме розглянуто і проаналізовано наступні пункти: підставу для розробки, мету і призначення розробки, вимоги до програми, вимоги до функціональних характеристик, організацію вхідних і вихідних даних, критерії ефективності та якості програми, вимоги до надійності, умови експлуатації, вимоги до складу і параметрів технічних засобів, стадії та етапи розробки, моделі життєвого циклу, порядок контролю та приймання проекту.

У третьому розділі «Проектування системи» було виконано концептуальне, логічне і фізичне проектування бази даних «video_library», приведено таблиці ЗНФ. Також виконане проектування системи програмного додатку, визначено позначення, що застосовуються в UML для позначення видимості полів, методів і зв'язків між класами (інтерфейсами), наведено діаграму класів програми.

У четвертому розділі «Технічна реалізація і впровадження програмного продукту» було обґрунтовано вибір програмних засобів для розробки проекту, детально розглянуто алгоритми роботи програмного продукту, його логічну структуру. Також було розроблено інструкцію користувача додатком videoLib, наведено результати тестування, визначено можливість супроводу ПЗ.

Отже, у курсовому проекті було розглянуто повний цикл зі створення програмного забезпечення, починаючи з аналізу і формування вимог (розробки технічного завдання) і закінчуючи впровадженням програмного продукту.

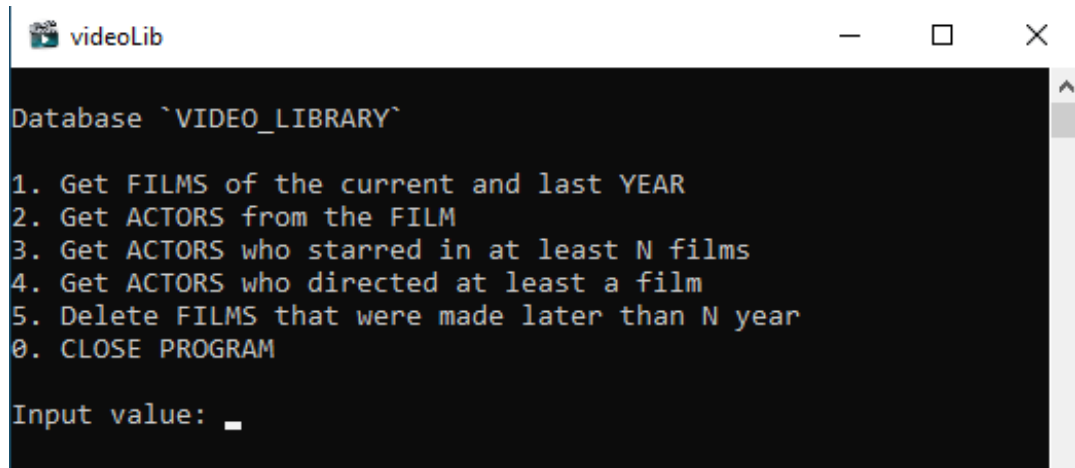
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Шилдт, Герберт. Java 8. Повне керівництво, 9-те вид.: Пер. з англ. – М.: ТОВ «В.Д. Вільямс», 2015. – 1376 с.: іл. – Парал. тит. англ.
2. Берд, Барі. Java для початківців, 5-те вид.: Пер. з англ. – М.: ТОВ «В.Д. Вільямс», 2013. – 368 с.: іл. – Парал. тит. англ.
3. Блох, Джошуа. Java: ефективне програмування, 3-тє вид.: Пер. з англ. – СПб.: ТОВ «Діалектика», 2019. – 464 с.: іл. – Парал. тит. англ.
4. Дюбуа П. MySQL. Збірник рецептів. – Пер. з англ. – СПб.: Символ-Плюс, 2006. – 1056 с., іл.
5. Мурах, Джоел. Murach's MySQL, 3-тє вид. – USA: Mike Murach & Associates, Inc., 2019. – 628 с., іл.
6. Особливості мови Java — Хекслет [Електронний ресурс] – Режим доступу до ресурсу: <https://ru.hexlet.io/blog/posts/yazyk-programmirovaniya-java-osobennosti-populyarnost-situatsiya-na-rynke-truda>
7. Особливості структурованої мови запитів SQL [Електронний ресурс] – Режим доступу до ресурсу: <https://practicum.yandex.ru/blog/chto-takoe-sql>
8. Діаграми UML для моделювання процесів й архітектури проекту — Evergreen [Електронний ресурс] – Режим доступу до ресурсу: <https://evergreens.com.ua/ru/articles/uml-diagrams.html>
9. Бази даних — R Class Tech [Електронний ресурс] – Режим доступу до ресурсу: <https://youtube.com/playlist?list=PLf30vI0hEi1v435cBmZSHkr1QAJdOk9mb>
10. Правила формування відношень у БД — Автор24 [Електронний ресурс] – Режим доступу до ресурсу: https://spravochnick.ru/bazy_dannyh/metod_suschnost-svyaz_osnovnye_ponyatiya_metoda/pravila_formirovaniya_otnosheniy
11. Переваги СКБД MySQL — Depix [Електронний ресурс] – Режим доступу до ресурсу: https://depix.ru/articles/sistema_upravleniya_bazami_dannyh_mysql
12. Переваги IDE IntelliJ IDEA — SkillFactory [Електронний ресурс] – Режим доступу до ресурсу: <https://blog.skillfactory.ru/glossary/intellij-idea>

Додаток А

Екранні форми програми

На рисунках А.1-А.10 наведено екранні форми програмного додатку videoLib:




```

videoLib
Database `VIDEO_LIBRARY`

1. Get FILMS of the current and last YEAR
2. Get ACTORS from the FILM
3. Get ACTORS who starred in at least N films
4. Get ACTORS who directed at least a film
5. Delete FILMS that were made later than N year
0. CLOSE PROGRAM

Input value: _
  
```

Рисунок А.1 Початковий екран. Меню програми VideoLib. Запрошення введення команди



```

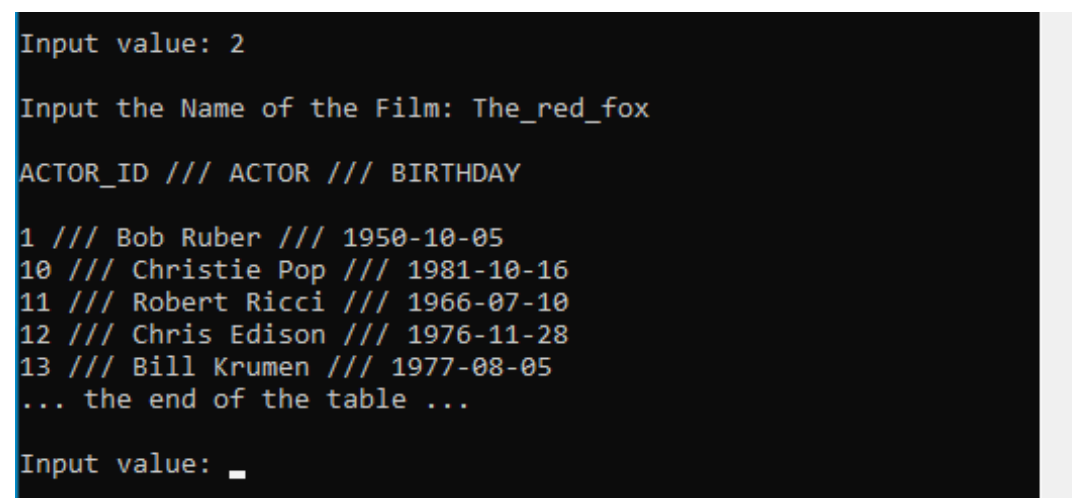
Input value: 1

FILM_ID /// NAME /// DIRECTOR_ID /// COUNTRY /// YEAR

8 /// It*s_my_life /// 6 /// Australia /// 2022
9 /// Penguins_don*t_fly! /// 4 /// USA /// 2023
... the end of the table ...

Input value: _
  
```

Рисунок А.2 Результат виконання команди 1 : «Фільми, зняті поточного і минулого років»



```

Input value: 2

Input the Name of the Film: The_red_fox

ACTOR_ID /// ACTOR /// BIRTHDAY

1 /// Bob Ruber /// 1950-10-05
10 /// Christie Pop /// 1981-10-16
11 /// Robert Ricci /// 1966-07-10
12 /// Chris Edison /// 1976-11-28
13 /// Bill Krumen /// 1977-08-05
... the end of the table ...

Input value: _
  
```

Рисунок А.3 Результат виконання команди 2 : «Актори, що знімалися у заданому фільмі»

```

Input value: 3

Input the count of roles: 2

ACTOR_ID /// ACTOR /// BIRTHDAY /// ROLES

1 /// Bob Ruber /// 1950-10-05 /// 3
2 /// Michelle Andrade /// 1990-01-07 /// 2
4 /// Lucy Chado /// 1968-05-26 /// 2
5 /// Linda Rowling /// 1972-12-31 /// 2
6 /// John Smith /// 1977-07-25 /// 2
7 /// Mary Star /// 1988-08-19 /// 3
8 /// Lisa Kraft /// 1973-09-15 /// 3
12 /// Chris Edison /// 1976-11-28 /// 3
13 /// Bill Krumen /// 1977-08-05 /// 2
14 /// Mike Dandy /// 1971-12-10 /// 2
... the end of the table ...

Input value: _

```

Рисунок А.4 Результат виконання команди 3 : «Актори, що зіграли не менше N ролей»

```

Input value: 4

ACTOR_ID /// ACTOR /// BIRTHDAY /// FILMS

6 /// John Smith /// 1977-07-25 /// 1
8 /// Lisa Kraft /// 1973-09-15 /// 3
14 /// Mike Dandy /// 1971-12-10 /// 2
22 /// Chris Garik /// 1968-04-28 /// 1
... the end of the table ...

Input value: _

```

Рисунок А.5 Результат виконання команди 4 : «Актори, що були ще й режисерами фільмів»

```

Input value: 5

Input the year: 2019

WARNING /// Selected films will be permanently deleted!
Confirm deleting. Input 1 to delete or 0 to cancel: _

```

Рисунок А.6 Результат виконання команди 5 : «Видалення фільмів, що виходили на екран, починаючи із вказаного року». Приклад видалення фільмів, дата виходу яких починається з 2019 року. Виведення попередження про видалення обраних фільмів

```
WARNING /// Selected films will be permanently deleted!
Confirm deleting. Input 1 to delete or 0 to cancel: 0

... deleting operation has been canceled ...

Input value: _
```

Рисунок А.7 Результат виконання команди 5 : «Видалення фільмів, що виходили на екран, починаючи із вказаного року». Приклад видалення фільмів, дата виходу яких починається з 2019 року. Скасування видалення фільмів (введення команди 0)

```
WARNING /// Selected films will be permanently deleted!
Confirm deleting. Input 1 to delete or 0 to cancel: 1

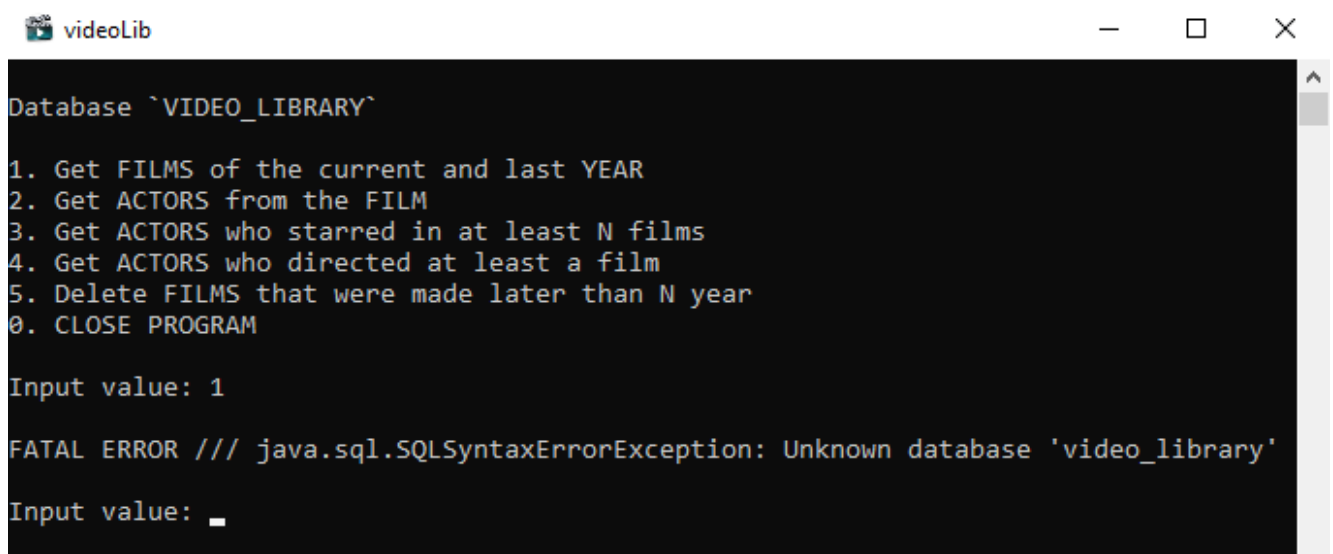
... 3 films have been just deleted permanently ...

Input value: _
```

Рисунок А.8 Результат виконання команди 5 : «Видалення фільмів, що виходили на екран, починаючи із вказаного року». Приклад видалення фільмів, дата виходу яких починається з 2019 року. Підтвердження видалення фільмів (введення команди 1)

```
Input value: 0
```

Рисунок А.9 Результат виконання команди 0 : «Вихід і завершення роботи програми»



```
videoLib
Database `VIDEO_LIBRARY`

1. Get FILMS of the current and last YEAR
2. Get ACTORS from the FILM
3. Get ACTORS who starred in at least N films
4. Get ACTORS who directed at least a film
5. Delete FILMS that were made later than N year
0. CLOSE PROGRAM

Input value: 1

FATAL ERROR /// java.sql.SQLException: Unknown database 'video_library'

Input value: _
```

Рисунок А.10 Помилка при підключенні до БД. База даних «video_library» не існує

Додаток Б

Вихідний код програми

Program.java

```

package com.company;

import com.company.db.DatabaseModification;
import com.company.db.DatabaseSelection;

import java.util.Scanner;

public class Program {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        DatabaseSelection dbSelection = new DatabaseSelection();
        DatabaseModification dbModification = new DatabaseModification();

        String menu = ""

            Database `VIDEO_LIBRARY`

            1. Get FILMS of the current and last YEAR
            2. Get ACTORS from the FILM
            3. Get ACTORS who starred in at least N films
            4. Get ACTORS who directed at least a film
            5. Delete FILMS that were made later than N year
            0. CLOSE PROGRAM
            "";
        System.out.print(menu);
        String filmName;
        int countRoles;
        int filmYear;

        boolean status = true;
        int value;

        while (status) {
            System.out.print("\nInput value: ");
            value = scanner.nextInt();
            if (value == 1)
                dbSelection.getFilmsCurLastYear();
            else if (value == 2) {
                System.out.print("\nInput the Name of the Film: ");
                filmName = scanner.next();
                dbSelection.getActorsFromFilm(filmName);
            }
            else if (value == 3) {
                System.out.print("\nInput the count of roles: ");
                countRoles = scanner.nextInt();
                dbSelection.getCountOfRolesNotLessThanN(countRoles);
            }
            else if (value == 4)

```

```

        dbSelection.getActorsDirectors();
    else if (value == 5) {
        System.out.print("\nInput the year: ");
        filmYear = scanner.nextInt();
        System.out.println("\nWARNING    ///    Selected    films    will    be
permanently deleted!");
        System.out.print("Confirm deleting. Input 1 to delete or 0 to
cancel: ");
        value = scanner.nextInt();
        if (value == 1)
            dbModification.deleteFilmsYearMoreThanN(filmYear);
        else
            System.out.println("\n... deleting operation has been canceled
...");
    }
    else if (value == 0) {
        status = false;
    }
    else {
        System.out.println("\nThis command is not supported!\nInput other
command, please.");
    }
}
}
}

```

DatabaseSelection.java

```

package com.company.db;

import com.company.db.configurations.DatabaseConstants;
import com.company.db.connection.DatabaseConnection;

import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class DatabaseSelection {
    DatabaseConnection dbConnection = new DatabaseConnection();
    Statement statement;
    PreparedStatement preparedStatement;
    ResultSet resultSet;
    private String sqlSelectStatement;

    public void getFilmsCurLastYear() {
        try {
            statement = dbConnection.getDbConnection().createStatement();
            setSqlSelectStatement("SELECT * FROM " + DatabaseConstants.FILMS_TABLE
+ " WHERE " + DatabaseConstants.FILMS_YEAR + " = year(curdate()) OR " +
DatabaseConstants.FILMS_YEAR + " = year(curdate())-1 ORDER BY " +
DatabaseConstants.FILMS_ID + ";");
            resultSet = statement.executeQuery(getSqlSelectStatement());

```

```

        System.out.println("\nFILM_ID /// NAME /// DIRECTOR_ID /// COUNTRY ///
YEAR\n");
        while (resultSet.next()) {
            int filmId = resultSet.getInt(DatabaseConstants.FILMS_ID);
            String name = resultSet.getString(DatabaseConstants.FILMS_NAME);
            int directorId
resultSet.getInt(DatabaseConstants.FILMS_DIRECTOR);
            String country
resultSet.getString(DatabaseConstants.FILMS_COUNTRY);
            int year = resultSet.getInt(DatabaseConstants.FILMS_YEAR);
            System.out.printf("%d /// %s /// %d /// %s /// %d\n", filmId, name,
directorId, country, year);
        }
        System.out.println("... the end of the table ...");
    }
    catch (ClassNotFoundException | SQLException e) {
        System.out.println("\nFATAL ERROR /// " + e);
    }
}

public void getActorsFromFilm(String filmName) {
    try {
        setSqlSelectStatement("SELECT " + DatabaseConstants.ACTOR_ID + ", " +
DatabaseConstants.ACTOR_NAME + " AS Actor, " + DatabaseConstants.ACTOR_BIRTHDAY
+ " FROM " + DatabaseConstants.ACTOR_TABLE + " JOIN " +
DatabaseConstants.FILMS_ACTORS_TABLE + " ON " + DatabaseConstants.ACTOR_ID + " =
" + DatabaseConstants.FILMS_ACTORS_ACTOR + " JOIN " + DatabaseConstants.FILMS_TABLE
+ " ON " + DatabaseConstants.FILMS_ID + " = " + DatabaseConstants.FILMS_ACTORS_FILM
+ " WHERE " + DatabaseConstants.FILMS_NAME + " = ? ORDER BY " +
DatabaseConstants.ACTOR_ID + ";");
        preparedStatement
dbConnection.getConnection().prepareStatement(getSqlSelectStatement());
        preparedStatement.setString(1, filmName);
        resultSet = preparedStatement.executeQuery();
        System.out.println("\nACTOR_ID /// ACTOR /// BIRTHDAY\n");
        while (resultSet.next()) {
            int id = resultSet.getInt(DatabaseConstants.ACTOR_ID);
            String actor = resultSet.getString("Actor");
            String birthday
resultSet.getString(DatabaseConstants.ACTOR_BIRTHDAY);
            System.out.printf("%d /// %s /// %s\n", id, actor, birthday);
        }
        System.out.println("... the end of the table ...");
    }
    catch (ClassNotFoundException | SQLException e) {
        System.out.println("\nFATAL ERROR /// " + e);
    }
}

public void getCountOfRolesNotLessThanN(int countRoles) {
    try {
        setSqlSelectStatement("SELECT " + DatabaseConstants.ACTOR_ID + ", " +
DatabaseConstants.ACTOR_NAME + " AS Actor, " +
DatabaseConstants.ACTOR_BIRTHDAY + ", count(*) AS Roles FROM " +
DatabaseConstants.ACTOR_TABLE + " JOIN " + DatabaseConstants.FILMS_ACTORS_TABLE +

```

```

" ON " + DatabaseConstants.ACTOR_ID + " = " + DatabaseConstants.FILMS_ACTORS_ACTOR
+ " JOIN " + DatabaseConstants.FILMS_TABLE + " ON " + DatabaseConstants.FILMS_ID +
" = " + DatabaseConstants.FILMS_ACTORS_FILM + " GROUP BY " +
DatabaseConstants.ACTOR_ID + " HAVING Roles >= ? ORDER BY " +
DatabaseConstants.ACTOR_ID + ";");
    preparedStatement
    =
dbConnection.getConnection().prepareStatement(getSqlSelectStatement());
    preparedStatement.setInt(1, countRoles);
    resultSet = preparedStatement.executeQuery();
    System.out.println("\nACTOR_ID /// ACTOR /// BIRTHDAY /// ROLES\n");
    while (resultSet.next()) {
        int id = resultSet.getInt(DatabaseConstants.ACTOR_ID);
        String actor = resultSet.getString("Actor");
        String birthday
        =
resultSet.getString(DatabaseConstants.ACTOR_BIRTHDAY);
        int roles = resultSet.getInt("Roles");
        System.out.printf("%d /// %s /// %s /// %d\n", id, actor, birthday,
roles);
    }
    System.out.println("... the end of the table ...");
}
catch (ClassNotFoundException | SQLException e) {
    System.out.println("\nFATAL ERROR /// " + e);
}
}

public void getActorsDirectors() {
    try {
        statement = dbConnection.getConnection().createStatement();
        setSqlSelectStatement("SELECT " + DatabaseConstants.ACTOR_ID + ", " +
            DatabaseConstants.ACTOR_NAME + " AS Actor, " +
DatabaseConstants.DIRECTORS_BIRTHDAY + ", count(*) AS Films FROM " +
DatabaseConstants.DIRECTORS_TABLE + " JOIN " + DatabaseConstants.FILMS_TABLE + " ON
" + DatabaseConstants.DIRECTORS_ID + " = " + DatabaseConstants.FILMS_DIRECTOR + "
JOIN " + DatabaseConstants.ACTOR_TABLE + " ON " + DatabaseConstants.ACTOR_NAME +
" = " + DatabaseConstants.DIRECTORS_NAME + " GROUP BY " +
DatabaseConstants.ACTOR_ID + " ORDER BY " + DatabaseConstants.ACTOR_ID + ";");
        resultSet = statement.executeQuery(getSqlSelectStatement());
        System.out.println("\nACTOR_ID /// ACTOR /// BIRTHDAY /// FILMS\n");
        while (resultSet.next()) {
            int id = resultSet.getInt(DatabaseConstants.ACTOR_ID);
            String actor = resultSet.getString("Actor");
            String birthday
            =
resultSet.getString(DatabaseConstants.DIRECTORS_BIRTHDAY);
            int films = resultSet.getInt("Films");
            System.out.printf("%d /// %s /// %s /// %d\n", id, actor, birthday,
films);
        }
        System.out.println("... the end of the table ...");
    }
    catch (ClassNotFoundException | SQLException e) {
        System.out.println("\nFATAL ERROR /// " + e);
    }
}
}

```

```

protected String getSqlSelectStatement() {
    return sqlSelectStatement;
}

protected void setSqlSelectStatement(String sql) {
    sqlSelectStatement = sql;
}
}

```

DatabaseModification.java

```

package com.company.db;

import com.company.db.configurations.DatabaseConstants;
import com.company.db.connection.DatabaseConnection;

import java.sql.PreparedStatement;
import java.sql.SQLException;

public class DatabaseModification {
    DatabaseConnection dbConnection = new DatabaseConnection();
    PreparedStatement preparedStatement;
    public void deleteFilmsYearMoreThanN(int year) {
        try {
            String deleteFilmsYearMoreThanNStr = "DELETE FROM " +
DatabaseConstants.FILMS_TABLE + " WHERE " + DatabaseConstants.FILMS_YEAR + " > ?;";
            preparedStatement =
dbConnection.getDbConnection().prepareStatement(deleteFilmsYearMoreThanNStr);
            preparedStatement.setInt(1, year);
            int deleteRows = preparedStatement.executeUpdate();
            System.out.printf("\n... %d films have been just deleted
permanently ...\n", deleteRows);
        }
        catch (ClassNotFoundException | SQLException e) {
            System.out.println("\nFATAL ERROR /// " + e);
        }
    }
}

```

DatabaseConnection.java

```

package com.company.db.connection;

import com.company.db.configurations.DatabaseConfigurations;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DatabaseConnection extends DatabaseConfigurations {
    Connection dbConnection;
}

```



```

        public Connection getDbConnection() throws ClassNotFoundException,
SQLException {
            String url = "jdbc:mysql://" + dbHost + ":" + dbPort + "/" + dbName;
            Class.forName("com.mysql.cj.jdbc.Driver");
            dbConnection = DriverManager.getConnection(url, dbUsername,
dbPassword);
            return dbConnection;
        }
    }
}

```

DatabaseConfigurations.java

```

package com.company.db.configurations;

public class DatabaseConfigurations {
    protected String dbHost = "localhost";
    protected String dbPort = "3306";
    protected String dbName = "video_library";
    protected String dbUsername = "root";
    protected String dbPassword = "12345";
}

```

DatabaseConstants.java

```

package com.company.db.configurations;

public class DatabaseConstants {
    public static final String DIRECTORS_TABLE = "directors";
    public static final String DIRECTORS_ID = DIRECTORS_TABLE +
".Director_ID";
    public static final String DIRECTORS_NAME = DIRECTORS_TABLE +
".Full_name";
    public static final String DIRECTORS_BIRTHDAY = DIRECTORS_TABLE +
".Birthday";
    public static final String ACTORS_TABLE = "actors";
    public static final String ACTORS_ID = ACTORS_TABLE + ".Actor_ID";
    public static final String ACTORS_NAME = ACTORS_TABLE + ".Full_name";
    public static final String ACTORS_BIRTHDAY = ACTORS_TABLE + ".Birthday";
    public static final String FILMS_TABLE = "films";
    public static final String FILMS_ID = FILMS_TABLE + ".Film_ID";
    public static final String FILMS_NAME = "Film_name";
    public static final String FILMS_DIRECTOR = FILMS_TABLE + ".Director_ID";
    public static final String FILMS_COUNTRY = "Country";
    public static final String FILMS_YEAR = "_Year";
    public static final String FILMS_ACTORS_TABLE = "films_actors";
    public static final String FILMS_ACTORS_FILM = FILMS_ACTORS_TABLE +
".Film_ID";
    public static final String FILMS_ACTORS_ACTOR = FILMS_ACTORS_TABLE +
".Actor_ID";
}

```

Додаток В

SQL запити на створення і наповнення тестовими даними БД

createDB.sql

```
CREATE DATABASE video_library;
USE video_library;
CREATE TABLE directors (
Director_ID INT AUTO_INCREMENT,
Full_name VARCHAR(50) NOT NULL,
Birthday DATE,
CONSTRAINT pk_directors PRIMARY KEY (Director_ID),
CONSTRAINT uq_director_name UNIQUE INDEX (Full_name)
);
CREATE TABLE actors (
Actor_ID INT AUTO_INCREMENT,
Full_name VARCHAR(50) NOT NULL,
Birthday DATE,
CONSTRAINT pk_actors PRIMARY KEY (Actor_ID),
CONSTRAINT uq_actor_name UNIQUE INDEX (Full_name)
);
CREATE TABLE films (
Film_ID INT AUTO_INCREMENT,
Film_name VARCHAR(100) NOT NULL,
Director_ID INT NOT NULL,
Country VARCHAR(50),
_Year YEAR,
CONSTRAINT pk_films PRIMARY KEY (Film_ID),
CONSTRAINT fk_directors FOREIGN KEY (Director_ID)
REFERENCES directors(Director_ID) ON UPDATE RESTRICT ON DELETE CASCADE,
CONSTRAINT uq_film_name UNIQUE INDEX (Film_name)
);
CREATE TABLE films_actors (
Film_ID INT NOT NULL,
Actor_ID INT NOT NULL,
```

```

CONSTRAINT fk_films FOREIGN KEY (Film_ID)
REFERENCES films(Film_ID) ON UPDATE RESTRICT ON DELETE CASCADE,
CONSTRAINT fk_actors FOREIGN KEY (Actor_ID)
REFERENCES actors(Actor_ID) ON UPDATE RESTRICT ON DELETE CASCADE
);

```

insertDB.sql

```

INSERT INTO directors (Full_name, Birthday) VALUES
('Mark Cooper', '1956-05-12'),
('Chris Garik', '1968-04-28'),
('Mike Dandy', '1971-12-10'),
('Lisa Kraft', '1973-09-15'),
('Silver McArthur', '1960-02-01'),
('John Smith', '1977-07-25');
INSERT INTO actors (Full_name, Birthday) VALUES
('Bob Ruber', '1950-10-05'),
('Michelle Andrade', '1990-01-07'),
('George Schwartz', '1947-03-17'),
('Lucy Chado', '1968-05-26'),
('Linda Rowling', '1972-12-31'),
('John Smith', '1977-07-25'),
('Mary Star', '1988-08-19'),
('Lisa Kraft', '1973-09-15'),
('William Bertie', '1970-04-29'),
('Christie Pop', '1981-10-16'),
('Robert Ricci', '1966-07-10'),
('Chris Edison', '1976-11-28'),
('Bill Krumen', '1977-08-05'),
('Mike Dandy', '1971-12-10'),
('Kate Scroll', '1983-01-19'),
('Steve Gretchen', '1996-03-30'),
('Peter Parker', '1985-04-24'),
('Claude Van Damme', '1972-06-12'),

```

```

('Spike Person', '1971-08-18'),
('Roberto Montini', '1982-12-15'),
('Amelie King', '1992-05-22'),
('Chris Garik', '1968-04-28');
INSERT INTO films (Film_name, Director_ID, Country, _Year) VALUES
('The_old_man', 1, 'Canada', 2008),
('The_sea', 2, 'USA', 2010),
('The_red_fox', 3, 'Great Britain', 2011),
('Who_is_that?', 4, 'USA', 2013),
('The_secret_of_the_black_box', 5, 'USA', 2015),
('The_crime_is_solved!', 4, 'USA', 2018),
('I_love_you,_Jennifer!', 3, 'Great Britain', 2020),
('It*s_my_life', 6, 'Australia', 2022),
('Penguins_don*t_fly!', 4, 'USA', 2023);
INSERT INTO films_actors VALUES
(1, 1), (1, 2), (1, 3), (1, 4),
(2, 5), (2, 6), (2, 7), (2, 8), (2, 9),
(3, 10), (3, 11), (3, 1), (3, 12), (3, 13),
(4, 14), (4, 15), (4, 16), (4, 17),
(5, 8), (5, 18), (5, 7), (5, 6),
(6, 19), (6, 12), (6, 4),
(7, 12), (7, 2), (7, 20), (7, 5),
(8, 8), (8, 14), (8, 7), (8, 1),
(9, 21), (9, 13), (9, 22);

```