Київський нацiональний унiверситет iменi Тараса Шевченка Факультет комп’ютерних наук та кiбернетики

**ЛАБОРАТОРНА РОБОТА №7**

«Представлення даних в XML / Засоби доступу до даних»

Виконала:

Студентка групи ІПС-31

Ящук Ангеліна

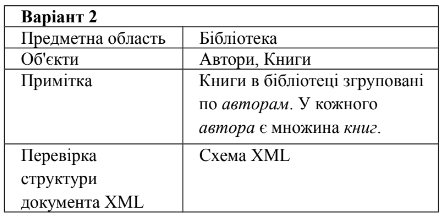
Київ, 2023

**ЛАБОРАТОРНА РОБОТА №7**

«Представлення даних в XML»

**Мета:** розробити програму, що забезпечує введення і редагування інформації про об'єкти відповідно до заданої предметної області. Інформація про об'єкти повинна зберігатися в окремому файлі в форматі XML.

Варіант завдання:



**Лістинг розробленої програми**

package org.labs;  
  
import javax.xml.bind.annotation.XmlElement;  
import javax.xml.bind.annotation.XmlElementWrapper;  
import javax.xml.bind.annotation.XmlRootElement;  
import java.util.ArrayList;  
import java.util.List;  
  
@XmlRootElement  
public class Author {  
 private String name;  
 private List<Book> books;  
  
 public Author(String name, List<Book> books) {  
 this.name = name;  
 this.books = books;  
 }  
  
 @XmlElement  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 @XmlElementWrapper(name = "books")  
 @XmlElement(name = "book")  
 public List<Book> getBooks() {  
 return books;  
 }  
  
 public void setBooks(List<Book> books) {  
 this.books = books;  
 }  
  
 public Author() {  
 books = new ArrayList<>();  
 }  
  
 public void addBook(Book newBook) {  
 books.add(newBook);  
 }  
}

package org.labs;  
  
import javax.xml.bind.annotation.XmlElement;  
import javax.xml.bind.annotation.XmlRootElement;  
  
@XmlRootElement  
public class Book {  
 private String title;  
 private String isbn;  
  
  
 public Book(String title, String isbn) {  
 this.title = title;  
 this.isbn = isbn;  
 }  
  
 @XmlElement  
 public String getTitle() {  
 return title;  
 }  
  
 public void setTitle(String title) {  
 this.title = title;  
 }  
  
 @XmlElement  
 public String getIsbn() {  
 return isbn;  
 }  
  
 public void setIsbn(String isbn) {  
 this.isbn = isbn;  
 }  
  
 public Book() {  
 }  
}

package org.labs;  
  
import javax.xml.bind.annotation.XmlElement;  
import javax.xml.bind.annotation.XmlElementWrapper;  
import javax.xml.bind.annotation.XmlRootElement;  
import java.util.List;  
  
@XmlRootElement  
public class Library {  
 private List<Author> authors;  
  
 public Library(List<Author> authors) {  
 this.authors = authors;  
 }  
  
 @XmlElementWrapper(name = "authors")  
 @XmlElement(name = "author")  
 public List<Author> getAuthors() {  
 return authors;  
 }  
  
 public void setAuthors(List<Author> authors) {  
 this.authors = authors;  
 }  
  
 public Library() {  
 }  
}

package org.labs;  
  
import java.io.File;  
import java.util.List;  
  
import javax.xml.bind.JAXBContext;  
import javax.xml.bind.JAXBException;  
import javax.xml.bind.Marshaller;  
import javax.xml.bind.Unmarshaller;  
  
import java.util.ArrayList;  
import java.util.Scanner;  
  
public class LibraryManager {  
 private List<Author> authors;  
  
 private void loadFromXML(String filename) {  
 try {  
 File file = new File(filename);  
 if (!file.exists()) {  
 authors = new ArrayList<>();  
 return;  
 }  
  
 JAXBContext context = JAXBContext.newInstance(Library.class);  
 Unmarshaller unmarshaller = context.createUnmarshaller();  
 Library library = (Library) unmarshaller.unmarshal(file);  
 authors = library.getAuthors();  
 } catch (JAXBException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private void saveToXML(String filename) {  
 try {  
 JAXBContext context = JAXBContext.newInstance(Library.class);  
 Marshaller marshaller = context.createMarshaller();  
 Library library = new Library(authors);  
 marshaller.marshal(library, new File(filename));  
 } catch (JAXBException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void addAuthor(Author author) {  
 if (authors == null) {  
 authors = new ArrayList<>();  
 }  
 authors.add(author);  
 }  
  
 public void removeAuthor(Author author) {  
 if (authors != null) {  
 authors.remove(author);  
 }  
 }  
  
 public void updateAuthor(Author updatedAuthor) {  
 if (authors != null) {  
 for (int i = 0; i < authors.size(); i++) {  
 Author existingAuthor = authors.get(i);  
 if (existingAuthor.getName().equals(updatedAuthor.getName())) {  
 authors.set(i, updatedAuthor);  
 break;  
 }  
 }  
 }  
 }  
  
 public Author findAuthorByName(String name) {  
 if (authors != null) {  
 for (Author author : authors) {  
 if (author.getName().equals(name)) {  
 return author;  
 }  
 }  
 }  
 return null;  
 }  
  
 public List<Author> getAllAuthors() {  
 return authors;  
 }  
  
 private void printAllBooks() {  
 if (authors != null) {  
 for (Author author : authors) {  
 for (Book book : author.getBooks()) {  
 System.out.println("Title: " + book.getTitle() + ", ISBN: " + book.getIsbn());  
 }  
 }  
 } else {  
 System.out.println("No authors in the library.");  
 }  
 }  
  
 private void printAllAuthors() {  
 if (authors != null) {  
 for (Author author : authors) {  
 System.out.println("Author: " + author.getName());  
 }  
 } else {  
 System.out.println("No authors in the library.");  
 }  
 }  
  
 private void addNewAuthor() {  
 Scanner scanner = new Scanner(System.in);  
 Author newAuthor = new Author();  
  
 System.out.print("Enter author name: ");  
 newAuthor.setName(scanner.nextLine());  
  
  
 if (authors == null) {  
 authors = new ArrayList<>();  
 }  
  
 authors.add(newAuthor);  
 System.out.println("Author added successfully.");  
 }  
  
 private void removeAuthor() {  
 Scanner scanner = new Scanner(System.in);  
 System.out.print("Enter author name to remove: ");  
 String authorNameToRemove = scanner.nextLine();  
  
 if (authors != null) {  
 authors.removeIf(author -> author.getName().equals(authorNameToRemove));  
 System.out.println("Author removed successfully.");  
 } else {  
 System.out.println("No authors in the library.");  
 }  
 }  
  
 private void updateAuthor() {  
 Scanner scanner = new Scanner(System.in);  
 System.out.print("Enter author name to update: ");  
 String authorNameToUpdate = scanner.nextLine();  
  
 if (authors != null) {  
 for (Author existingAuthor : authors) {  
 if (existingAuthor.getName().equals(authorNameToUpdate)) {  
 System.out.print("Enter new name for the author: ");  
 String newAuthorName = scanner.nextLine();  
 existingAuthor.setName(newAuthorName);  
 System.out.println("Author updated successfully.");  
 return;  
 }  
 }  
 System.out.println("Author not found.");  
 } else {  
 System.out.println("No authors in the library.");  
 }  
 }  
  
 private void findAuthorByName() {  
 Scanner scanner = new Scanner(System.in);  
 System.out.print("Enter author name to find: ");  
 String authorNameToFind = scanner.nextLine();  
  
 Author foundAuthor = findAuthorByName(authorNameToFind);  
  
 if (foundAuthor != null) {  
 System.out.println("Author found: " + foundAuthor.getName());  
 } else {  
 System.out.println("Author not found.");  
 }  
 }  
  
 private void addBookToAuthor(Author author) {  
 Scanner scanner = new Scanner(System.in);  
 Book newBook = new Book();  
  
 System.out.print("Enter book title: ");  
 newBook.setTitle(scanner.nextLine());  
  
 System.out.print("Enter book ISBN: ");  
 newBook.setIsbn(scanner.nextLine());  
  
 author.addBook(newBook);  
 System.out.println("Book added to the author successfully.");  
 }  
  
 public static void main(String[] args) {  
 LibraryManager manager = new LibraryManager();  
 manager.loadFromXML("library.xml");  
  
 Scanner scanner = new Scanner(System.in);  
 int choice;  
  
 do {  
 System.out.println("Library Manager Menu:");  
 System.out.println("1. Print all books");  
 System.out.println("2. Print all authors");  
 System.out.println("3. Add new author");  
 System.out.println("4. Remove author");  
 System.out.println("5. Update author");  
 System.out.println("6. Find author by name");  
 System.out.println("7. Add book to author");  
 System.out.println("0. Exit");  
 System.out.print("Enter your choice: ");  
  
 choice = scanner.nextInt();  
 scanner.nextLine();  
  
 switch (choice) {  
 case 1:  
 manager.printAllBooks();  
 break;  
 case 2:  
 manager.printAllAuthors();  
 break;  
 case 3:  
 manager.addNewAuthor();  
 break;  
 case 4:  
 manager.removeAuthor();  
 break;  
 case 5:  
 manager.updateAuthor();  
 break;  
 case 6:  
 manager.findAuthorByName();  
 break;  
 case 7:  
 System.out.print("Enter author name:");  
 Author author = manager.findAuthorByName(scanner.nextLine());  
 manager.addBookToAuthor(author);  
 break;  
 case 0:  
 System.out.println("Exiting Library Manager.");  
 break;  
 default:  
 System.out.println("Invalid choice. Try again.");  
 }  
 } while (choice != 0);  
  
 manager.saveToXML("library.xml");  
 }  
}

**Клас Book:**

* **Конструктор Book(String title, String isbn):**
  + Створює об'єкт книги з заданою назвою та ISBN.
* **Геттер getTitle():**
  + Повертає назву книги.
* **Сеттер setTitle(String title):**
  + Встановлює назву книги.
* **Геттер getIsbn():**
  + Повертає ISBN книги.
* **Сеттер setIsbn(String isbn):**
  + Встановлює ISBN книги.

**Клас Author:**

* **Конструктор Author(String name, List<Book> books):**
  + Створює об'єкт автора з іменем та списком його книг.
* **Геттер getName():**
  + Повертає ім'я автора.
* **Сеттер setName(String name):**
  + Встановлює ім'я автора.
* **Геттер getBooks():**
  + Повертає список книг автора.
* **Сеттер setBooks(List<Book> books):**
  + Встановлює список книг автора.
* **Метод addBook(Book newBook):**
  + Додає нову книгу до списку книг автора.

**Клас Library:**

* **Геттер getAuthors():**
  + Повертає список авторів у бібліотеці.
* **Сеттер setAuthors(List<Author> authors):**
  + Встановлює список авторів у бібліотеці.

**Клас LibraryManager:**

* **Метод loadFromXML(String filename):**
  + Завантажує дані бібліотеки з XML-файлу та ініціалізує список авторів.
* **Метод saveToXML(String filename):**
  + Зберігає дані бібліотеки у XML-файл.
* **Метод addAuthor(Author author):**
  + Додає нового автора до бібліотеки.
* **Метод removeAuthor(Author author):**
  + Видаляє автора з бібліотеки.
* **Метод updateAuthor(Author updatedAuthor):**
  + Оновлює інформацію про автора.
* **Метод findAuthorByName(String name): Author:**
  + Знаходить автора за ім'ям та повертає його об'єкт.
* **Метод getAllAuthors(): List<Author>:**
  + Повертає список всіх авторів у бібліотеці.
* **Метод printAllBooks():**
  + Виводить інформацію про всі книги та їх авторів.
* **Метод printAllAuthors():**
  + Виводить інформацію про всіх авторів.
* **Метод addNewAuthor():**
  + Запитує користувача про ім'я нового автора та додає його до бібліотеки.
* **Метод removeAuthor():**
  + Запитує користувача про ім'я автора та видаляє його з бібліотеки.
* **Метод updateAuthor():**
  + Запитує користувача про ім'я автора та змінює його ім'я.
* **Метод findAuthorByName():**
  + Запитує користувача про ім'я автора та виводить інформацію про нього.
* **Метод addBookToAuthor(Author author):**
  + Додає нову книгу до автора, якого вказав користувач.
* **Метод main(String[] args):**
  + Викликається при запуску програми.
  + Ініціалізує менеджера бібліотеки, виводить меню для користувача та обробляє вибір користувача.

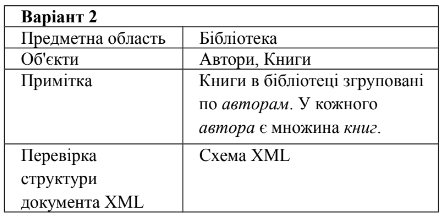
**Отриманий XML документ в результаті виконання**

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>  
<library>  
 <authors>  
 <author>  
 <books>  
 <book>  
 <isbn>1234567890</isbn>  
 <title>Book1</title>  
 </book>  
 <book>  
 <isbn>0987654321</isbn>  
 <title>Book2</title>  
 </book>  
 </books>  
 <name>Author1</name>  
 </author>  
 <author>  
 <books>  
 <book>  
 <isbn>9876543210</isbn>  
 <title>Book3</title>  
 </book>  
 </books>  
 <name>Author2</name>  
 </author>  
 <author>  
 <books>  
 <book>  
 <isbn>123456789</isbn>  
 <title>New Book</title>  
 </book>  
 </books>  
 <name>New Author</name>  
 </author>  
 </authors>  
</library>

**«**Засоби доступу до даних**»**

**Мета:** розробити програму, що забезпечує введення і редагування інформації про об'єкти відповідно до заданої предметної області. Інформація про об'єкти повинна зберігатися в окремій базі даних. Доступ до даних здійснюється з використанням засобів JDBC або ODBC.

**Варіант завдання:**



**Лістинг розробленої програми**

package org.labs;  
  
import java.util.ArrayList;  
import java.util.List;  
  
class Author {  
 private int id;  
 private String name;  
  
 private List<Book> books;  
  
 public Author(int id, String name) {  
 this.id = id;  
 this.name = name;  
 books = new ArrayList<>();  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public List<Book> getBooks() {  
 return books;  
 }  
}

package org.labs;  
  
class Book {  
 private int id;  
 private String title;  
 private String isbn;  
 private Author author;  
  
 public Book(int id, String title, String isbn, Author author) {  
 this.id = id;  
 this.title = title;  
 this.isbn = isbn;  
 this.author = author;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public String getTitle() {  
 return title;  
 }  
  
 public String getIsbn() {  
 return isbn;  
 }  
  
 public Author getAuthor() {  
 return author;  
 }  
}

package org.labs;  
  
import java.sql.\*;  
import java.util.Scanner;  
  
public class LibraryManager {  
  
 static final String *JDBC\_DRIVER* = "com.mysql.cj.jdbc.Driver";  
 static final String *DB\_URL* = "jdbc:mysql://localhost/library\_db";  
 static final String *USER* = "root";  
 static final String *PASS* = "12345";  
  
  
  
 private static void addAuthor(Statement stmt, String authorName) throws SQLException {  
 String sql = "INSERT INTO authors (name) VALUES ('" + authorName + "')";  
 stmt.executeUpdate(sql);  
 System.*out*.println("Додано нового автора: " + authorName);  
 }  
  
 private static void addBook(Statement stmt, String title, String isbn, int authorID) throws SQLException {  
 String sql = "INSERT INTO books (title, isbn, author\_id) VALUES ('" + title + "', '" + isbn + "', " + authorID + ")";  
 stmt.executeUpdate(sql);  
 System.*out*.println("Додано нову книгу: " + title);  
 }  
  
 private static int getAuthorID(Statement stmt, String authorName) throws SQLException {  
 String sql = "SELECT id FROM authors WHERE name = '" + authorName + "'";  
 ResultSet rs = stmt.executeQuery(sql);  
  
 if (rs.next()) {  
 return rs.getInt("id");  
 }  
  
 return -1;  
 }  
  
 private static void displayBooksByAuthor(Statement stmt, String authorName) throws SQLException {  
 int authorID = *getAuthorID*(stmt, authorName);  
 if (authorID != -1) {  
 String sql = "SELECT \* FROM books WHERE author\_id = " + authorID;  
 ResultSet rs = stmt.executeQuery(sql);  
  
 System.*out*.println("Книги автора " + authorName + ":");  
 while (rs.next()) {  
 System.*out*.println("Назва: " + rs.getString("title") + ", ISBN: " + rs.getString("isbn"));  
 }  
 } else {  
 System.*out*.println("Автор " + authorName + " не знайдений.");  
 }  
 }  
  
 public static void main(String[] args) {  
 Connection conn = null;  
 Statement stmt = null;  
 Scanner scanner = new Scanner(System.*in*);  
  
 try {  
 Class.*forName*(*JDBC\_DRIVER*);  
 conn = DriverManager.*getConnection*(*DB\_URL*, *USER*, *PASS*);  
 stmt = conn.createStatement();  
  
 int choice;  
  
 do {  
 System.*out*.println("Library Manager Menu:");  
 System.*out*.println("1. Print all books");  
 System.*out*.println("2. Print all authors");  
 System.*out*.println("3. Add new author");  
 System.*out*.println("4. Remove author");  
 System.*out*.println("5. Update author");  
 System.*out*.println("6. Find author by name");  
 System.*out*.println("7. Add book to author");  
 System.*out*.println("0. Exit");  
 System.*out*.print("Enter your choice: ");  
  
 choice = scanner.nextInt();  
 scanner.nextLine();  
  
 switch (choice) {  
 case 1:  
 *printAllBooks*(stmt);  
 break;  
 case 2:  
 *printAllAuthors*(stmt);  
 break;  
 case 3:  
 *addNewAuthor*(stmt);  
 break;  
 case 4:  
 *removeAuthor*(stmt);  
 break;  
 case 5:  
 *updateAuthor*(stmt);  
 break;  
 case 6:  
 *findAuthorByName*(stmt);  
 break;  
 case 7:  
 *addBookToAuthor*(stmt);  
 break;  
 case 0:  
 System.*out*.println("Exiting Library Manager.");  
 break;  
 default:  
 System.*out*.println("Invalid choice. Try again.");  
 }  
 } while (choice != 0);  
  
 } catch (SQLException | ClassNotFoundException se) {  
 se.printStackTrace();  
 } finally {  
 try {  
 if (stmt != null) stmt.close();  
 if (conn != null) conn.close();  
 if (scanner != null) scanner.close();  
 } catch (SQLException se) {  
 se.printStackTrace();  
 }  
 }  
 }  
  
 public static void printAllBooks(Statement stmt) throws SQLException {  
 String sql = "SELECT books.\*, authors.name AS author\_name " +  
 "FROM books JOIN authors ON books.author\_id = authors.id";  
 ResultSet rs = stmt.executeQuery(sql);  
  
 while (rs.next()) {  
 int bookId = rs.getInt("id");  
 String title = rs.getString("title");  
 String isbn = rs.getString("isbn");  
 String authorName = rs.getString("author\_name");  
  
 System.*out*.println("Book ID: " + bookId + ", Title: " + title +  
 ", ISBN: " + isbn + ", Author: " + authorName);  
 }  
 }  
  
 public static void printAllAuthors(Statement stmt) throws SQLException {  
 String sql = "SELECT \* FROM authors";  
 ResultSet rs = stmt.executeQuery(sql);  
  
 while (rs.next()) {  
 int authorId = rs.getInt("id");  
 String authorName = rs.getString("name");  
  
 System.*out*.println("Author ID: " + authorId + ", Name: " + authorName);  
 }  
 }  
  
 public static void addNewAuthor(Statement stmt) throws SQLException {  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.print("Enter author name: ");  
 String authorName = scanner.nextLine();  
  
 String sql = "INSERT INTO authors (name) VALUES ('" + authorName + "')";  
 stmt.executeUpdate(sql);  
 System.*out*.println("New author added: " + authorName);  
 }  
  
 public static void removeAuthor(Statement stmt) throws SQLException {  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.print("Enter author ID to remove: ");  
 int authorId = scanner.nextInt();  
  
 String sql = "DELETE FROM authors WHERE id = " + authorId;  
 int affectedRows = stmt.executeUpdate(sql);  
  
 if (affectedRows > 0) {  
 System.*out*.println("Author removed successfully.");  
 } else {  
 System.*out*.println("No author found with ID: " + authorId);  
 }  
 }  
  
 public static void updateAuthor(Statement stmt) throws SQLException {  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.print("Enter author ID to update: ");  
 int authorId = scanner.nextInt();  
 scanner.nextLine();  
  
 System.*out*.print("Enter new author name: ");  
 String newAuthorName = scanner.nextLine();  
  
 String sql = "UPDATE authors SET name = '" + newAuthorName + "' WHERE id = " + authorId;  
 int affectedRows = stmt.executeUpdate(sql);  
  
 if (affectedRows > 0) {  
 System.*out*.println("Author updated successfully.");  
 } else {  
 System.*out*.println("No author found with ID: " + authorId);  
 }  
 }  
  
 public static void findAuthorByName(Statement stmt) throws SQLException {  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.print("Enter author name to find: ");  
 String authorName = scanner.nextLine();  
  
 String sql = "SELECT \* FROM authors WHERE name = '" + authorName + "'";  
 ResultSet rs = stmt.executeQuery(sql);  
  
 while (rs.next()) {  
 int authorId = rs.getInt("id");  
 String foundAuthorName = rs.getString("name");  
  
 System.*out*.println("Author found - ID: " + authorId + ", Name: " + foundAuthorName);  
 }  
 }  
  
 public static void addBookToAuthor(Statement stmt) throws SQLException {  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.print("Enter author ID to add a book: ");  
 int authorId = scanner.nextInt();  
 scanner.nextLine();  
  
 System.*out*.print("Enter book title: ");  
 String bookTitle = scanner.nextLine();  
  
 System.*out*.print("Enter book ISBN: ");  
 String bookISBN = scanner.nextLine();  
  
 String sql = "INSERT INTO books (title, isbn, author\_id) VALUES ('" + bookTitle + "', '" + bookISBN + "', " + authorId + ")";  
 stmt.executeUpdate(sql);  
 System.*out*.println("New book added to the author.");  
 }  
  
}

**Клас Author:**

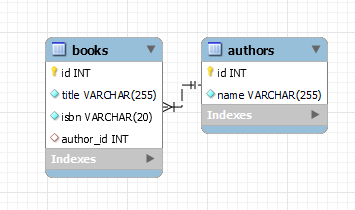
* **Конструктор Author(int id, String name):**
  + Створює об'єкт автора з заданим ідентифікатором та ім'ям.
* **Геттер getId():**
  + Повертає ідентифікатор автора.
* **Геттер getName():**
  + Повертає ім'я автора.
* **Геттер getBooks():**
  + Повертає список книг автора.

**Клас Book:**

* **Конструктор Book(int id, String title, String isbn, Author author):**
  + Створює об'єкт книги з заданим ідентифікатором, назвою, ISBN та автором.
* **Геттер getId():**
  + Повертає ідентифікатор книги.
* **Геттер getTitle():**
  + Повертає назву книги.
* **Геттер getIsbn():**
  + Повертає ISBN книги.
* **Геттер getAuthor():**
  + Повертає об'єкт автора, до якого належить книга.

**Клас LibraryManager:**

* **Статичні поля:**
  + **JDBC\_DRIVER**, **DB\_URL**, **USER**, **PASS**: конфігураційні параметри для з'єднання з базою даних.
* **Метод addAuthor(Statement stmt, String authorName):**
  + Додає нового автора до бази даних.
* **Метод addBook(Statement stmt, String title, String isbn, int authorID):**
  + Додає нову книгу до бази даних з вказаним автором.
* **Метод getAuthorID(Statement stmt, String authorName): int:**
  + Повертає ідентифікатор автора за його ім'ям.
* **Метод displayBooksByAuthor(Statement stmt, String authorName):**
  + Виводить на екран усі книги вказаного автора з бази даних.
* **Метод printAllBooks(Statement stmt):**
  + Виводить на екран інформацію про всі книги та їх авторів.
* **Метод printAllAuthors(Statement stmt):**
  + Виводить на екран інформацію про всіх авторів.
* **Метод addNewAuthor(Statement stmt):**
  + Додає нового автора до бази даних.
* **Метод removeAuthor(Statement stmt):**
  + Видаляє автора з бази даних за його ідентифікатором.
* **Метод updateAuthor(Statement stmt):**
  + Оновлює інформацію про автора у базі даних.
* **Метод findAuthorByName(Statement stmt):**
  + Знаходить автора за його ім'ям та виводить інформацію про нього.
* **Метод addBookToAuthor(Statement stmt):**
  + Додає нову книгу до автора за його ідентифікатором.
* **Метод main(String[] args):**
  + Викликається при запуску програми.
  + Забезпечує взаємодію з користувачем та викликає інші методи для виконання операцій над бібліотекою.

**ER модель бази даних**

**Висновок**

Виконуючи дану лабораторну роботу було розроблено програму, що забезпечує введення і редагування інформації про об'єкти відповідно до заданої предметної області (Бібліотека). Інформація про об'єкти зберігається в окремому файлі в форматі XML. Другою частиною лабораторної роботи стала розробка програму, що забезпечує введення і редагування інформації про об'єкти відповідно до заданої предметної області. Інформація про об'єкти зберігається в окремій базі даних. Доступ до даних здійснюється з використанням засобів JDBC.