**ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ**

**«КОЛОМИЙСЬКИЙ ПОЛІТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ**

**НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

**Освітньо-професійна програма** «Інженерія програмного забезпечення» **Спеціальність** 121 «Інженерія програмного забезпечення

**КУРСОВА РОБОТА**

**З ДИСЦИПЛІНИ «Об’єктно-орієнтоване програмування»**

на тему: АРМ бібліотекаря

виконав:

Студент 3 курсу група П-31

Павлюк В.І.

(прізвище та ініціали)

Керівник: Красничук В.В.

(прізвище та ініціали)  
Оцінка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члени комісії:

\_\_\_\_\_\_\_\_\_\_\_\_\_ /Ляшеник А.В/ .

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Смиковчук Т.В./

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Красничук В.В./

(підпис) (прізвище та ініціали)

м. Коломия  2023

**ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ**

**КОЛОМИЙСЬКИЙ ПОЛІТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ  
НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Відділення: | | денне, комп’ютерних технологій | | | | | |
| Циклова комісія: | | | інженерії програмного забезпечення | | | | |
| Освітньо-кваліфікаційний рівень: | | | | | фаховий молодший бакалавр | | |
| Освітньо-професійна програма | | | | | Інженерія програмного забезпечення | | |
| Спеціальність: | | | | 121 «Інженерія програмного забезпечення» | | |
| Галузь знань: | 12 «Інформаційні технології» | | | | |

**ЗАТВЕРДЖУЮ**Голова циклової комісії інженерії програмного забезпечення

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Т.Т. Дуб/

“\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 року

**ЗАВДАННЯ**

**НА КУРСОВУ РОБОТУ**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| студенту | Павлюку Володимиру Івановичу | | | | | | |
| 1. Тема роботи | | | | Автоматизоване робоче місце бібліотекаря | | | |
|  | | |  | | | | |
| керівник | | Красничук Вікторія Вікторівна | | | | | |
| 1. Термін подання студентом закінченої роботи | | | | | | 19 червня 2023р. | |
| 1. Вихідні дані до роботи: | | | | | | | |
| Створити автоматизоване робоче місце бібліотекаря | | | | | | | |
|  | | | | | | | |
|  | | | | | | | |
| 1. Зміст розрахунково-пояснювальної записки | | | | | | | |
| 1. Аналіз технічного завдання | | | | | | | |
| 2. Обґрунтування алґоритму й структури програми | | | | | | | |
| 3. Розроблення програми | | | | | | | |
| 3.1 Розроблення бази даних | | | | | | | |
| 3.2 Розроблення методів | | | | | | | |
| 3.3 Створення об'єктів | | | | | | | |
| * 1. Опис інтерфейсу програми | | | | | | | |
| 4. Тестування програми і результати її виконання | | | | | | | |
| Дата видачі завдання | | | | | 10.03.2023р | |
|  | | | | | | | |

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назва етапів курсової роботи | Строк виконання етапів роботи | Примітка |
| 1 | Постановка завдання. Аналіз специфікації вимог. Визначення структури даних, методів рішення | 25.03.2023 |  |
| 2 | Визначення структури програмних модулів Розробка алгоритму, визначення форми представлення даних, структури програми | 12.04.2023 |  |
| 3 | Програмування. Розробка документації. | 29.04.2023 |  |
| 4 | Тестування і верифікація програми | 06.05.2023 |  |
| 5 | Оформлення пояснювальної записки | 10.05.2023 |  |
| 6 | Виправлення недоліків та доробка програми. Представлення роботи до попереднього розгляду | 30.05.2023 |  |
| 7 | Захист роботи. | 20.06.2023 |  |

Студент:\_\_\_\_\_\_\_\_\_\_ Павлюк Володимир Іванович

(прізвище та ініціали)

Керівник:\_\_\_\_\_\_\_\_\_ Красничук Вікторія Вікторівна

(прізвище та ініціали)

**Зміст**

Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

3

**206 КР 121.01.17 ПЗ**

Розробив

Павлюк В.І.

Перевірив

Красничук В.В.

Реценз.

Н. Контр.

Затвердив

АРМ бібліотекаря

Літ.

Акрушів

74

КФПК НУ «ЛП»

ПЕРЕЛІК СКОРОЧЕНЬ4

ВСТУП5

1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ7

1.1 Визначення необхідності створення ПЗ7

1.2 Постановка задачі9

2 ОБҐРУНТУВАННЯ АЛГОРИТМУ Й СТРУКТУРИ ПРОГРАМИ 11

3 РОЗРОБЛЕННЯ ПРОГРАМИ12

3.1 Створення бази даних12

3.2 Розробка методів 15

3.3 Створення об’єктів 35

3.4 Опис інтерфейсу програми 38

4 ТЕСТУВАННЯ ПРОГРАМИ І РЕЗУЛЬТАТИ ЇЇ ВИКОНАННЯ 40

ВИСНОВОК 45

ЛІТЕРАТУРА 46

ДОДАТОК А47

**ПЕРЕЛІК СКОРОЧЕНЬ**

АРМ – Автоматизоване робоче місце

БД – База даних

ООП – Об’єктно орієнтоване програмування

ПЗ – Програмне забезпечення

ТЗ – Технічне завдання

EER – Enhanced Entity-Relationship

**ВСТУП**

АРМ є скороченням від "автоматизоване робоче місце" (АРМ) і використовується для опису системи, яка забезпечує комплексну автоматизацію робочого процесу або окремої діяльності. АРМ є програмним або апаратним засобом, розробленим з метою полегшити та підвищити продуктивність роботи користувача, спрямовану на виконання певних завдань.

АРМ може бути розроблено для різних галузей та видів діяльності, включаючи управління, бухгалтерію, медицину, освіту, виробництво, торгівлю тощо. Воно може включати в себе комплекс програмного забезпечення, апаратних засобів (наприклад, комп'ютери, принтери, сканери), комунікаційних засобів та інші технічні компоненти.

Основні переваги АРМ включають:

1. Підвищення ефективності роботи: АРМ надає користувачу інструменти та функціонал, що спрощують та прискорюють виконання завдань. Він допомагає автоматизувати рутинні операції, зменшує час, затрачений на виконання завдань та забезпечує більш точні результати.
2. Збереження даних та інформації: АРМ дозволяє зберігати, організовувати та швидко отримувати доступ до важливих даних та інформації. Це сприяє покращенню управління та прийняттю рішень на основі актуальних даних.
3. Автоматизація робочих процесів: АРМ дозволяє автоматизувати послідовність операцій та процесів, забезпечуючи їх послідовне виконання без необхідності ручного втручання користувача. Це допомагає знизити ймовірність помилок та покращує загальну продуктивність.
4. Покращений контроль та звітність: АРМ забезпечує можливість моніторингу, контролю та аналізу робочих процесів. Користувач може отримати детальну інформацію про виконання завдань, стан проектів, облік ресурсів тощо.

АРМ дозволяє покращити організацію робочого процесу, забезпечуючи ефективність та точність виконання завдань, а також зниження часу та зусиль, витрачених на виконання роботи.

1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

1.1 Визначення необхідності створення ПЗ

Створення АРМ (автоматизованої робочої місце) для бібліотекаря може бути корисним і ефективним рішенням для оптимізації та поліпшення роботи в бібліотеці. Ось кілька причин, чому може виникнути необхідність у створенні АРМ для бібліотекаря:

1. Управління каталогом: АРМ може надати інструменти для автоматизованого управління каталогом книг, періодичних видань та іншого матеріалу в бібліотеці. Це дозволить бібліотекарю легко додавати, видаляти та змінювати записи, контролювати наявність та розташування матеріалів, швидко знаходити інформацію про певну книгу або автора.
2. Облік видачі та повернення: АРМ може спростити процес видачі та повернення книг. За допомогою АРМ бібліотекар може швидко реєструвати видачу книг, вносити дані про читачів, контролювати терміни повернення та стягувати штрафи за запізнення. Це дозволяє автоматизувати процес, зменшує ймовірність помилок та сприяє більш ефективному обліку.
3. Онлайн-каталог та доступ до інформації: АРМ може підтримувати онлайн-каталог, який дозволяє читачам швидко знаходити та замовляти книги, переглядати інформацію про наявність та розташування матеріалів, резервувати книги тощо. Це зручно для користувачів бібліотеки та дозволяє бібліотекарям більш ефективно управляти запитами та попитом на матеріали.
4. Звітність та статистика: АРМ може забезпечити функціонал для генерації звітів та статистики про роботу бібліотеки, включаючи кількість видач, запізнення, популярність книжок тощо. Це допомагає аналізувати та вдосконалювати роботу бібліотеки, враховуючи потреби та попит читачів.

АРМ для бібліотекаря може значно спростити та покращити роботу в бібліотеці, забезпечуючи ефективне управління матеріалами, видачу та повернення, доступ до інформації та звітність.

1.2 Постановка задачі

1. Додавання нової книги або читача в каталог:

- Можливість внесення нових записів про книги, включаючи назву, автора, видавництво, кількість книг в наявності, рік видання та жанр.

- Внесення даних про нового читача, включаючи ім'я, прізвище, контактну інформацію та дату народження.

2. Редагування існуючих записів:

- Можливість змінювати дані про книги, такі як автор, видавництво, тощо.

- Редагування даних про читачів, наприклад, контактну інформацію.

3. Видалення записів:

- Можливість видаляти записи про книги, які більше не знаходяться в бібліотеці або були втрачені.

- Видалення даних про читачів, наприклад, у випадку їх вибуття або відмови від членства в бібліотеці.

4. Пошук за різними параметрами:

- Можливість здійснювати пошук книг за назвою, автором, жанром, та іншими параметрами.

- Пошук читачів за ім'ям, прізвищем або іншими відомостями.

5. Видача і повернення книг:

- Функціонал для оформлення процесу видачі книги читачеві, включаючи реєстрацію дати видачі та інших деталей.

- Можливість реєструвати повернення книги, оновлювати статус та виконувати необхідні облікові операції.

Ці функції допоможуть бібліотекарю ефективно керувати каталогом книг, виконувати операції видачі та повернення, знаходити необхідну інформацію та забезпечувати організований робочий процес в бібліотеці.

**2 ОБҐРУНТУВАННЯ АЛГОРИТМУ Й СТРУКТУРИ ПРОГРАМИ**

Для розробки програми АРМ бібліотекаря використовуватиметься мова програмування C++ в середовищі QT Creator з використанням фреймворку QT.

Алгоритм:

1. Програма починає роботу з виведення головного вікна, яке містить основні функції і таблиці АРМ бібліотекаря.
2. Програма надає можливість користувачу вибирати функції з меню.
3. Кожна функція меню реалізує певну операцію, таку як додавання, зміна і видалення книг і читачів, або перехід до іншого вікна де можна видати або повернути книгу.

Структура програми:

1. Програма буде орієнтована на об'єктно-орієнтований підхід, що дозволить організувати код у вигляді класів та об'єктів.
2. Використовуватиметься БД, що дозволить зручно зберігати та керувати даними.
3. Кожна таблиця БД матиме відповідні методи для додавання, зміни та видалення даних.
4. Для забезпечення взаємодії з користувачем буде реалізовано відповідний графічний інтерфейс, де за допомогою кнопок та полів для введення, користувач зможе працювати з програмою.

Такий алгоритм та структура програми дозволять створити АРМ бібліотекаря, що забезпечуватиме ефективне керування книгами, читачами та виданими книгами, а також зручний та інтуїтивно зрозумілий інтерфейс для користувача.

3 РОЗРОБЛЕННЯ ПРОГРАМИ

3.1 Створення бази даних

База даних (БД) є організованою колекцією даних, яка зберігається та управляється з використанням певної системи управління базами даних (СУБД). Вона служить для збереження, організації, керування та отримання доступу до інформації.

База даних складається з таблиць, які представляють сутності або об'єкти, про які ми збираємо інформацію. Кожна таблиця складається з рядків (записів) і стовпців (полів). Рядки представляють окремі екземпляри даних, а стовпці визначають типи даних, які можуть бути збережені.

Замість системи класів та файлової системи було використано базу даних, бо вона має такі переваги:

1. Збереження даних: База даних дозволяє зберігати великий обсяг даних ефективно. Вона надає механізми для структурування та організації даних, забезпечуючи ефективний доступ до них. Використання бази даних дозволяє зберігати інформацію про книги, читачів, видачу та повернення книг, історію та інші дані, які можуть бути необхідні для функціонування АРМ бібліотекаря.
2. Пошук та фільтрація даних: База даних дозволяє виконувати складні запити для пошуку, фільтрації та сортування даних. Наприклад, бібліотекар може шукати книги за назвою, автором, жанром або доступністю. База даних забезпечує швидкий та ефективний доступ до великого обсягу даних, дозволяючи здійснювати різноманітні запити.
3. Цілісність даних: База даних надає механізми для забезпечення цілісності даних. Вона може використовувати обмеження, правила та валідацію даних для запобігання некоректним або неповним записам. Наприклад, база даних може вимагати, щоб кожна книга мала унікальний ідентифікатор або вказувати обмеження на формат дати.
4. Безпека даних: База даних дозволяє забезпечувати захист даних шляхом реалізації механізмів автентифікації, авторизації та контролю доступу. Це дозволяє обмежити доступ до даних лише авторизованим користувачам та забезпечити конфіденційність інформації.
5. Розширюваність: Використання бази даних дає можливість розширювати функціональність системи шляхом додавання нових таблиць, полів та відносин між ними без значних змін у структурі програмного коду. Це робить систему більш гнучкою та легко модифікованою у майбутньому.

Загалом, використання бази даних у системі АРМ бібліотекаря дозволяє ефективно зберігати, керувати та отримувати доступ до даних, що забезпечує більш ефективну та функціональну роботу бібліотекаря. Тому замість системи класів було використано базу даних.

Для створення бази даних для АРМ бібліотекаря необхідно визначити структуру бази даних, яка відображатиме необхідну інформацію про книги, користувачів та інші деталі бібліотеки. Тому було визначено такі основні таблиці і поля для бази даних:

1. Таблиця "Books":
   1. Id: унікальний ідентифікатор книги
   2. Name: назва книги
   3. Author: автор книги
   4. Year: рік, коли була видана книга
   5. Genre: категорія або жанр книги
   6. Existence: показник, що вказує на кількість книг в наявності
   7. Edition: видавництво де була надрукована книга
2. Таблиця "Readers":
   1. Id: унікальний ідентифікатор користувача
   2. First\_name: ім'я користувача
   3. Last\_name: прізвище користувача
   4. Phone\_num: номер телефону користувача
   5. Birthday: Дата народження користувача
3. Таблиця "Taken\_books":
   1. Id: унікальний ідентифікатор видачі
   2. Reader\_id: ідентифікатор користувача, який отримав книгу
   3. Book\_id: ідентифікатор книги, яка була видана
   4. Date: дата, коли була здійснена видача
   5. Status: статус видачі

Для наглядності можна переглянути EER-модель на Рис 3.1.1

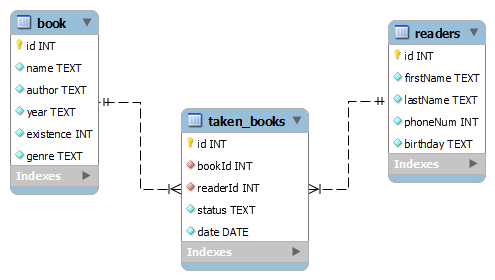


Рис 3.1.1 EER-модель

**3.2 Розробка методів**

Методи для додавання

Для додавання записів в БД використовуються окремі методи:

- void MainWindow::on\_pushButton\_4\_clicked() :для додавання книг

- void MainWindow::on\_pushButton\_5\_clicked() :для додавання читачів

І оскільки ці два методи схожі їх можна розібрати на прикладі одного з них.

Наприклад void MainWindow::on\_pushButton\_4\_clicked().

Код:

void MainWindow::on\_pushButton\_4\_clicked()

{

QString name1 = ui->lineEdit->text();

int year = (ui->lineEdit\_2->text()).toInt();

QString author=ui->lineEdit\_8->text();

int existence = ui->lineEdit\_3->text().toInt();

QString genre = ui->lineEdit\_4->text();

QString edition = ui->lineEdit\_5->text();

QSqlQuery q1;

q1.prepare("INSERT INTO books (name,author, year, existence, genre, edition) VALUES (:name,:author, :year, :existence, :genre, :edition)");

q1.bindValue(":name", name1);

q1.bindValue(":year", year);

q1.bindValue(":author",author);

q1.bindValue(":existence", existence);

q1.bindValue(":genre", genre);

q1.bindValue(":edition", edition);

q1.exec();

model->select();

}

Цей метод «on\_pushButton\_4\_clicked()» відповідає за додавання нового запису до таблиці «books» у базі даних. Основний алгоритм методу наступний:

1. Отримання значень полів з користувацького інтерфейсу:

* 1. «name1»: назва книги
  2. «year»: рік видання книги
  3. «author»: автор книги
  4. «existence»: кількість примірників книги, що є в наявності
  5. «genre»: жанр книги
  6. «edition»: видавництво книги

2. Підготовка SQL-запиту для вставки нового запису в таблицю «books». Запит містить параметри, які будуть замінені на конкретні значення.

q1.prepare("INSERT INTO books (name, author, year, existence, genre, edition) VALUES (:name, :author, :year, :existence, :genre, :edition)");

3. Прив'язка значень до параметрів запиту:

q1.bindValue(":name", name1);

q1.bindValue(":year", year);

q1.bindValue(":author", author);

q1.bindValue(":existence", existence);

q1.bindValue(":genre", genre);

q1.bindValue(":edition", edition);

4. Виконання SQL-запиту для вставки запису в таблицю:

q1.exec();

5. Оновлення моделі, що відображає дані з бази даних у користувацькому інтерфейсі:

model->select();

Цей метод використовує об'єкт «QSqlQuery» для взаємодії з базою даних та оновлює модель, що відображає дані у відповідному інтерфейсі користувача.

Методи для видалення

Для видалення записів з бази даних використовуються окремі методи:

- void MainWindow::on\_pushButton\_7\_clicked()

- void MainWindow::on\_deleteReader\_clicked()

І оскільки ці два методи схожі їх можна розібрати на прикладі одного з них.

Наприклад void MainWindow::on\_pushButton\_7\_clicked().

Код:

void MainWindow::on\_pushButton\_7\_clicked()

{

QItemSelectionModel\* selectionModel = ui->tableView->selectionModel();

QModelIndexList selectedIndexes = selectionModel->selectedIndexes();

QList<int> selectedIds;

foreach(const QModelIndex& index, selectedIndexes) {

int id = index.sibling(index.row(), 0).data().toInt();

selectedIds.append(id);

}

foreach(int id, selectedIds) {

QSqlQuery query;

query.prepare("DELETE FROM books WHERE id = :id");

query.bindValue(":id", id);

query.exec();

}

model->select();

}

Цей метод відповідає за обробку події натискання кнопки "Видалити" («pushButton\_7») у головному вікні («MainWindow»). Короткий опис його функціональності:

1. Отримання вибраних елементів таблиці:
   1. Отримуємо об'єкт «QItemSelectionModel», який відповідає за вибірку в таблиці.
   2. Отримуємо список «selectedIndexes», який містить індекси вибраних елементів у таблиці.
2. Отримання ідентифікаторів вибраних книг:
   1. Створюємо порожній список «selectedIds», який буде містити ідентифікатори вибраних книг.
   2. Проходимо по кожному індексу у списку «selectedIndexes».
   3. Отримуємо значення ідентифікатора книги з першого стовпця поточного індексу за допомогою «index.data().toInt()».
   4. Додаємо отриманий ідентифікатор до списку «selectedIds».
3. Видалення вибраних книг:
   1. Проходимо по кожному ідентифікатору у списку «selectedIds».
   2. Створюємо об'єкт «QSqlQuery» для виконання SQL-запиту.
   3. Підготовлюємо запит на видалення запису з таблиці «books», де ідентифікатор дорівнює поточному значенню «id».
   4. Прив'язуємо значення «id» до підстановочного параметра «:id» за допомогою «bindValue()».
   5. Виконуємо запит за допомогою «exec()».
4. Оновлення моделі таблиці:
   1. Після виконання видалення книг оновлюємо модель «model», яка відповідає за відображення даних у таблиці.
   2. Викликаємо метод «select()» моделі, щоб оновити дані у таблиці.

Отже, метод отримує вибрані елементи таблиці, отримує їх ідентифікатори та видаляє відповідні записи з бази даних. Після цього оновлюється відображення таблиці.

Методи для редагування

Для редагування записів з бази даних використовуються окремі методи:

* void MainWindow::on\_pushButton\_6\_clicked() :для отримання даних про книгу для редагування,

void MainWindow::on\_pushButton\_8\_clicked() :для зміни і підтвердження даних про книгу,

void MainWindow::on\_pushButton\_9\_clicked() :для скасування редагування даних про книгу.

* void MainWindow::on\_reductReader\_clicked() :для отримання даних про читача для редагування,

void MainWindow::on\_AcceptResuctReader\_clicked() : для зміни і підтвердження даних про читача,

void MainWindow::on\_pushButton\_10\_clicked() :для скасування редагування даних про читача.

І оскільки ці два набори методів схожі їх можна розібрати на прикладі одного з них.

Наприклад void MainWindow::on\_pushButton\_6\_clicked(),

void MainWindow::on\_pushButton\_8\_clicked(),

void MainWindow::on\_pushButton\_9\_clicked().

Код:

void MainWindow::on\_pushButton\_6\_clicked()

{

QItemSelectionModel\* selectionModel = ui->tableView->selectionModel();

QModelIndexList selectedIndexes = selectionModel->selectedIndexes();

foreach(const QModelIndex& index, selectedIndexes) {

id = index.sibling(index.row(), 0).data().toInt();

}

QSqlQuery query;

query.prepare("SELECT \* FROM books WHERE id = :id");

query.bindValue(":id", id);

query.exec();

if (query.next()) {

ui->lineEdit->setText(query.value("name").toString());

ui->lineEdit\_2->setText(query.value("year").toString());

ui->lineEdit\_8->setText(query.value("author").toString());

ui->lineEdit\_3->setText(query.value("existence").toString());

ui->lineEdit\_4->setText(query.value("genre").toString());

ui->lineEdit\_5->setText(query.value("edition").toString());

ui->pushButton\_8->show();

ui->pushButton\_9->show();

ui->pushButton\_4->hide();

ui->pushButton\_6->hide();

ui->pushButton\_7->hide();

}

}

void MainWindow::on\_pushButton\_8\_clicked()

{

QSqlQuery updateQuery;

updateQuery.prepare("UPDATE books SET name = :name,author=:author,year=:year,existence=:existence,genre=:genre,edition=:edition WHERE ID = :id");

updateQuery.bindValue(":id", id);

updateQuery.bindValue(":name", ui->lineEdit->text());

updateQuery.bindValue(":author", ui->lineEdit\_8->text());

updateQuery.bindValue(":year", ui->lineEdit\_2->text());

updateQuery.bindValue(":existence", ui->lineEdit\_3->text());

updateQuery.bindValue(":genre", ui->lineEdit\_4->text());

updateQuery.bindValue(":edition", ui->lineEdit\_5->text());

updateQuery.exec();

model->select();

ui->pushButton\_8->hide();

ui->pushButton\_9->hide();

ui->pushButton\_4->show();

ui->pushButton\_6->show();

ui->pushButton\_7->show();

ui->lineEdit->setText("");

ui->lineEdit\_2->setText("");

ui->lineEdit\_8->setText("");

ui->lineEdit\_3->setText("");

ui->lineEdit\_4->setText("");

ui->lineEdit\_5->setText("");

id=-1;

}

void MainWindow::on\_pushButton\_9\_clicked()

{

id=-1;

ui->pushButton\_8->hide();

ui->pushButton\_9->hide();

ui->lineEdit->setText("");

ui->lineEdit\_2->setText("");

ui->lineEdit\_8->setText("");

ui->lineEdit\_3->setText("");

ui->lineEdit\_4->setText("");

ui->lineEdit\_5->setText("");

ui->pushButton\_4->show();

ui->pushButton\_6->show();

ui->pushButton\_7->show();

model->select();

}

Перші два методи («on\_pushButton\_6\_clicked» та «on\_pushButton\_8\_clicked») відповідають за редагування вибраного запису в таблиці, а останній метод («on\_pushButton\_9\_clicked») скасовує редагування і повертає інтерфейс у початковий стан.

Метод «on\_pushButton\_6\_clicked»:

1. Отримання вибраного елемента таблиці:
   1. Отримуємо об'єкт «QItemSelectionModel», який відповідає за вибірку в таблиці.
   2. Отримуємо список «selectedIndexes», який містить індекси вибраних елементів у таблиці.
2. Отримання ідентифікатора вибраного запису:
   1. Проходимо по кожному індексу у списку «selectedIndexes».
   2. Отримуємо значення ідентифікатора запису з першого стовпця поточного індексу за допомогою «index.data().toInt()».
   3. Зберігаємо отриманий ідентифікатор в змінній «id».
3. Виконання запиту на отримання даних запису:
   1. Створюємо об'єкт «QSqlQuery» для виконання SQL-запиту.
   2. Підготовлюємо запит на вибірку всіх даних з таблиці «books», де ідентифікатор дорівнює значенню «id».
   3. рив'язуємо значення «id» до підстановочного параметра «:id» за допомогою «bindValue()».
   4. Виконуємо запит за допомогою «exec()».
4. Встановлення значень полів у віджеті:
   1. Якщо запит успішний та повертає результат, то встановлюємо значення полів віджета з відповідних стовпців запиту.
   2. Встановлюємо видимість та приховування певних кнопок («pushButton\_8», «pushButton\_9», «pushButton\_4», «pushButton\_6», «pushButton\_7») для відображення відповідного стану інтерфейсу.

Метод «on\_pushButton\_8\_clicked»:

1. Виконання запиту на оновлення запису:
   1. Створюємо об'єкт «QSqlQuery» для виконання SQL-запиту на оновлення даних запису.
   2. Підготовлюємо запит на оновлення таблиці «books», встановлюючи нові значення полів.
   3. Прив'язуємо значення «id» до підстановочного параметра «:id» за допомогою «bindValue()».
   4. Виконуємо запит за допомогою «exec()».
2. Оновлення моделі таблиці:
   1. Після виконання редагування запису оновлюємо модель «model», яка відповідає за відображення даних у таблиці.
   2. Викликаємо метод «select()» моделі, щоб оновити дані у таблиці.
3. Встановлення видимості та приховування певних кнопок та очищення полів:
   1. Приховуємо кнопки «pushButton\_8» та «pushButton\_9», а показуємо кнопки «pushButton\_4», «pushButton\_6» та «pushButton\_7».
   2. Очищуємо значення полів у віджеті.

Метод «on\_pushButton\_9\_clicked»:

1. Скасування редагування:
   1. Змінна «id» встановлюється в «-1», для запобіганню повторення.
   2. Очищуємо значення полів у віджеті.
2. Встановлення видимості та приховування певних кнопок:
   1. Показуємо кнопки «pushButton\_4», «pushButton\_6» та «pushButton\_7».
   2. Приховуємо кнопки «pushButton\_8» та «pushButton\_9».
3. Оновлення моделі таблиці:
   1. Викликаємо метод «select()» моделі, щоб оновити дані у таблиці.

Отже, перший метод дозволяє витягнути дані вибраного запису з бази даних та відобразити їх у віджеті для редагування. Другий метод виконує оновлення вибраного запису у базі даних на основі внесених змін. Третій метод відміняє редагування та повертає інтерфейс у початковий стан.

Методи для зміни меню

Для зміни меню використовуються окремі методи:

- void MainWindow::on\_pushButton\_2\_clicked() :меню з читачами

- void MainWindow::on\_pushButton\_clicked() :меню з книгами

- void MainWindow::on\_pushButton\_3\_clicked() :меню з виданими книгами

- void GiveBook::on\_pushButton\_4\_clicked(),

void GiveBook::on\_pushButton\_3\_clicked() :меню для видачі/повернення книг

- void GiveBook::on\_back\_clicked(),void GiveBook::on\_back2\_clicked() :повернення до меню для видачі/повернення книг

І оскільки ці сім методів схожі їх можна розібрати на прикладі одного з них.

Наприклад void void MainWindow::on\_pushButton\_clicked().

Код:

void MainWindow::on\_pushButton\_clicked()

{

ui->readersWidget->hide();

ui->booksWidget->show();

ui->TakenBooks->hide();

}

Цей метод відповідає за відображення віджетів, що стосуються книг, та приховання віджетів, що стосуються читачів та взятих книг.

1. «ui->readersWidget->hide();»: Цей рядок коду приховує віджет, який відповідає за відображення даних про читачів. За допомогою «ui->readersWidget» ми отримуємо доступ до віджету у головному вікні та викликаємо метод «hide()», щоб приховати його з відображення.

2. «ui->booksWidget->show();»: Цей рядок коду показує віджет, який відповідає за відображення даних про книги. За допомогою «ui->booksWidget» ми отримуємо доступ до віджету у головному вікні та викликаємо метод «show()», щоб показати його.

3. «ui->TakenBooks->hide();»: Цей рядок коду приховує віджет, який відповідає за відображення даних про взяті книги.

Отже, при виклику цього методу будуть показані віджети, що стосуються книг, а віджети про читачів та взяті книги будуть приховані.

Методи для пошуку

Для пошуку даних з БД використовуються окремі методи:

- void MainWindow::on\_lineEdit\_6\_textEdited(const QString &arg1),

void GiveBook::on\_lineEdit\_8\_textEdited(const QString &arg1) :для пошуку даних по книгам,

- void MainWindow::on\_lineEdit\_7\_textEdited(const QString &arg1),

void GiveBook::on\_lineEdit\_7\_textEdited(const QString &arg1) : для пошуку даних по читачам,

- void MainWindow::on\_lineEdit\_10\_textEdited(const QString &arg1) :для пошуку даних по виданим книгам.

І оскільки ці три методи схожі їх можна розібрати на прикладі одного з них.

Наприклад void MainWindow::on\_lineEdit\_6\_textEdited(const QString &arg1).

Код:

void MainWindow::on\_lineEdit\_6\_textEdited(const QString &arg1)

{

QString columnName;

switch (ui->comboBox\_findBook->currentIndex()) {

case 0:

columnName="name";

break;

case 1:

columnName="author";

break;

case 2:

columnName="year";

break;

case 3:

columnName="existence";

break;

case 4:

columnName="genre";

break;

case 5:

columnName="edition";

break;

default:

break;

}

model->setFilter(""+columnName+" LIKE '%"+ui->lineEdit\_6->text()+"%'");

}

Цей метод відповідає за фільтрацію даних у таблиці за введеним користувачем значенням у полі редагування «lineEdit\_6».

1. «QString columnName;»: Створюється змінна «columnName», яка буде містити назву стовпця, за яким буде виконуватись фільтрація.

2. «switch (ui->comboBox\_findBook->currentIndex())»: Використовується значення індексу вибраного елемента у комбінованому списку «comboBox\_findBook» для визначення відповідного стовпця, за яким буде виконуватись фільтрація. Залежно від індексу встановлюється значення для змінної «columnName».

3. «model->setFilter(""+columnName+" LIKE '%"+ui->lineEdit\_6->text()+"%'");»: Застосовується фільтр до моделі даних («model»), що використовується для відображення таблиці. Фільтр встановлюється на основі значення, введеного користувачем у полі редагування «lineEdit\_6» та вибраного стовпця «columnName». Використовується вираз «LIKE» з оператором «%», що дозволяє знайти записи, які містять введене значення відповідного стовпця.

Отже, при введенні тексту у поле «lineEdit\_6», відповідні дані у таблиці будуть фільтруватись відповідно до вибраного стовпця та введеного значення.

Метод для переходу на інше вікно

Для переходу на вікно з видачею і поверненням книг використовується окремий метод:

- void MainWindow::on\_giveBook\_clicked().

Код:

void MainWindow::on\_giveBook\_clicked()

{

GiveBook w;

w.setModal(true);

w.setWindowTitle("Видача/Повернення книг");

w.exec();

model->select();

model3->select();

}

Цей метод відкриває діалогове вікно «GiveBook», оновлює моделі даних та відображає зміни у відповідних таблицях.

1. «GiveBook w;»: Створюється об'єкт діалогового вікна «GiveBook».

2. «w.setModal(true);»: Встановлюється режим модального вікна для об'єкта «w», що означає, що користувач повинен завершити роботу з цим вікном, перш ніж повернутися до головного вікна програми.

3. «w.setWindowTitle("Видача/Повернення книг");»: Встановлюється заголовок для діалогового вікна «w».

4. «w.exec();»: Запускається виконання діалогового вікна. Виконання головного вікна програми зупиняється, поки користувач не закриє діалогове вікно «w».

5. «model->select();»: Оновлюється модель даних «model», що використовується для відображення таблиці книг. Це дозволяє оновити дані у таблиці після виконання змін у діалоговому вікні «GiveBook».

6. «model3->select();»: Оновлюється ще одна модель даних «model3», яка також використовується для відображення таблиці про видані книги. Це дозволяє оновити дані у цій таблиці після виконання змін у діалоговому вікні «GiveBook».

Отже, цей метод відкриває діалогове вікно, де, відбувається процес видання або повернення книги, після чого оновлюються дані у відповідних таблицях, щоб відобразити зміни.

Метод для вибору читача

Для вибору читача для видачі або повернення книг використовується окремий метод:

- void GiveBook::on\_pushButton\_clicked()

Код:

void GiveBook::on\_pushButton\_clicked()

{

QItemSelectionModel\* selectionModel = ui->tableView->selectionModel();

QModelIndexList selectedIndexes = selectionModel->selectedIndexes();

int id;

foreach(const QModelIndex& index, selectedIndexes) {

id = index.sibling(index.row(), 0).data().toInt();

}

QSqlQuery query;

query.prepare("Select \* FROM readers WHERE id = :id");

query.bindValue(":id", id);

query.exec();

ui->readers->hide();

ui->choose->show();

idReader=id;

model3->setFilter(" reader\_id="+QString::number(idReader)+" AND status=1");

model3->select();

}

Цей метод полягає в отриманні вибраного читача з таблиці.

1. «QItemSelectionModel\* selectionModel = ui->tableView->selectionModel();»: Створюється об'єкт «selectionModel», який містить вибрані елементи у таблиці «tableView» в діалоговому вікні «GiveBook».

2. «QModelIndexList selectedIndexes = selectionModel->selectedIndexes();»: Отримується список індексів вибраних елементів у таблиці.

3. «int id;»: Оголошується змінна «id», яка буде використовуватися для збереження ідентифікатора читача.

4. «foreach(const QModelIndex& index, selectedIndexes)»: Цикл перебирає всі індекси вибраних елементів.

5. «id = index.sibling(index.row(), 0).data().toInt();»: Значення ідентифікатора виділеного елемента зчитується з першої (індекс 0) колонки вибраного рядка.

6. «QSqlQuery query;»: Створюється об'єкт «query», який використовується для виконання запитів до бази даних.

7. «query.prepare("Select \* FROM readers WHERE id = :id");»: Підготовлюється запит до бази даних для вибору рядка з таблиці «readers», де ідентифікатор дорівнює значенню «id».

8. «query.bindValue(":id", id);»: Прив'язується значення «id» до плейсхолдера «:id» в запиті.

9. «query.exec();»: Виконується запит до бази даних.

10. «ui->readers->hide();»: Сховується віджет «readers» в діалоговому вікні.

11. «ui->choose->show();»: Відображається віджет «choose» в діалоговому вікні.

12. «idReader=id;»: Значення «id» присвоюється змінній «idReader», щоб його можна було використовувати в інших методах класу GiveBook.

13. model3->setFilter(" reader\_id="+QString::number(idReader)+" AND status=1");: Встановлюється фільтр для моделі даних model3, який вибирає рядки, де значення стовпця reader\_id дорівнює idReader, а значення стовпця status дорівнює 1.

14. model3->select();: Оновлюється модель даних model3, щоб відобразити результати запиту.

Отже, цей метод отримує вибрані елементи у таблиці, виконує запит до бази даних для отримання відповідних даних, оновлює модель даних та відображає результати у відповідних віджетах діалогового вікна GiveBook.

Метод для видачі книг

Для видачі книг використовується окремий метод:

- void GiveBook::on\_pushButton\_2\_clicked()

Код:

void GiveBook::on\_pushButton\_2\_clicked()

{

QItemSelectionModel\* selectionModel = ui->tableView\_2->selectionModel();

QModelIndexList selectedIndexes = selectionModel->selectedIndexes();

int id;

foreach(const QModelIndex& index, selectedIndexes) {

id = index.sibling(index.row(), 0).data().toInt();

}

QSqlQuery q1,q2;

QDate currentDate = QDate::currentDate();

q1.prepare("INSERT INTO taken\_books VALUES (NULL,:reader\_id,:book\_id,1,:date)");

q1.bindValue(":reader\_id", idReader);

q1.bindValue(":book\_id", id);

q1.bindValue(":date", currentDate);

q1.exec();

q2.prepare("UPDATE books SET existence=existence-1 WHERE id=:book\_id");

q2.bindValue(":book\_id", id);

q2.exec();

model2->select();

model3->select();

}

Цей метод полягає в отриманні вибраної книги з таблиці, виконанні запитів до бази даних для видачі книг та оновленні моделей даних для відображення результатів.

1. «QItemSelectionModel\* selectionModel = ui->tableView\_2->selectionModel();»: Створюється об'єкт «selectionModel», який містить вибрані елементи у таблиці «tableView\_2» в діалоговому вікні «GiveBook».

2. «QModelIndexList selectedIndexes = selectionModel->selectedIndexes();»: Отримується список індексів вибраних елементів у таблиці.

3. «int id;»: Оголошується змінна «id», яка буде використовуватися для збереження ідентифікатора виділеного елемента.

4. «foreach(const QModelIndex& index, selectedIndexes)»: Цикл перебирає всі індекси вибраних елементів.

5. «id = index.sibling(index.row(), 0).data().toInt();»: Значення ідентифікатора виділеного елемента зчитується з першої (індекс 0) колонки вибраного рядка.

6. «QSqlQuery q1, q2;»: Створюються об'єкти «q1» та «q2», які використовуються для виконання запитів до бази даних.

7. «QDate currentDate = QDate::currentDate();»: Отримується поточна дата.

8. «q1.prepare("INSERT INTO taken\_books VALUES (NULL,:reader\_id,:book\_id,1,:date)");»: Підготовлюється запит до бази даних для вставки нового запису в таблицю «taken\_books».

9. «q1.bindValue(":reader\_id", idReader);»: Прив'язується значення «idReader» до плейсхолдера «:reader\_id» в запиті.

10. «q1.bindValue(":book\_id", id);»: Прив'язується значення «id» до плейсхолдера «:book\_id» в запиті.

11. «q1.bindValue(":date", currentDate);»: Прив'язується значення «currentDate» до плейсхолдера «:date» в запиті.

12. «q1.exec();»: Виконується запит до бази даних для вставки нового запису.

13. «q2.prepare("UPDATE books SET existence=existence-1 WHERE id=:book\_id");»: Підготовлюється запит до бази даних для оновлення значення стовпця «existence» у таблиці «books».

14. «q2.bindValue(":book\_id", id);»: Прив'язується значення «id» до плейсхолдера «:book\_id» в запиті.

15. «q2.exec();»: Виконується запит до бази даних для оновлення запису.

16. «model2->select();»: Оновлюється модель даних «model2», щоб відобразити оновлені дані в таблиці.

17. «model3->select();»: Оновлюється модель даних «model3», щоб відобразити оновлені дані в таблиці.

Отже, цей метод дозволяє додати новий запис про видану книгу до бази даних, зменшити кількість доступних примірників книги та оновити відповідні моделі даних для відображення оновлених результатів у віджетах діалогового вікна «GiveBook».

Метод для повернення книг

Для повернення книг використовується окремий метод:

- void GiveBook::on\_pushButton\_6\_clicked()

Код:

void GiveBook::on\_pushButton\_6\_clicked()

{

QItemSelectionModel\* selectionModel = ui->tableView\_3->selectionModel();

QModelIndexList selectedIndexes = selectionModel->selectedIndexes();

QList<int> selectedIds;

foreach(const QModelIndex& index, selectedIndexes) {

int id = index.sibling(index.row(), 0).data().toInt();

selectedIds.append(id);

}

foreach(int id, selectedIds) {

QSqlQuery query;

query.prepare("UPDATE taken\_books SET status=0 WHERE id = :id");

query.bindValue(":id", id);

query.exec();

}

foreach (QModelIndex i, selectedIndexes) {

QVariant value = model3->data(model3->index(i.row(), 2));

QString id = value.toString();

QSqlQuery query;

query.prepare("UPDATE books SET existence=existence+1 WHERE id=:book\_id");

query.bindValue(":book\_id", id);

query.exec();

}

model3->select();

model2->select();

}

Цей полягає в обробці виділених виданих книг читача, виконанні запитів до бази даних для оновлення записів та оновленні моделей даних для відображення результатів.

1. «QItemSelectionModel\* selectionModel = ui->tableView\_3->selectionModel();»: Створюється об'єкт «selectionModel», який містить вибрані елементи у таблиці «tableView\_3» в діалоговому вікні «GiveBook».

2. «QModelIndexList selectedIndexes = selectionModel->selectedIndexes();»: Отримується список індексів вибраних елементів у таблиці.

3. «QList<int> selectedIds;»: Оголошується список «selectedIds», який буде використовуватися для збереження ідентифікаторів виділених елементів.

4. «foreach(const QModelIndex& index, selectedIndexes)»: Цикл перебирає всі індекси вибраних елементів.

5. «int id = index.sibling(index.row(), 0).data().toInt();»: Значення ідентифікатора виділеного елемента зчитується з першої (індекс 0) колонки вибраного рядка.

6. «selectedIds.append(id);»: Додається ідентифікатор до списку «selectedIds».

7. «foreach(int id, selectedIds)»: Цикл перебирає всі ідентифікатори в списку «selectedIds».

8. «QSqlQuery query;»: Створюється об'єкт «query», який використовується для виконання запитів до бази даних.

9. «query.prepare("UPDATE taken\_books SET status=0 WHERE id = :id");»: Підготовлюється запит до бази даних для оновлення значення стовпця «status» в таблиці «taken\_books».

10. «query.bindValue(":id", id);»: Прив'язується значення «id» до плейсхолдера «:id» в запиті.

11. «query.exec();»: Виконується запит до бази даних для оновлення запису.

12. «foreach (QModelIndex i, selectedIndexes)»: Цикл перебирає всі індекси вибраних елементів у таблиці.

13. «QVariant value = model3->data(model3->index(i.row(), 2));»: Зчитується значення з моделі «model3» за вказаним індексом «(i.row(), 2)».

14. «QString id = value.toString();»: Значення перетворюється в рядок.

15. «query.prepare("UPDATE books SET existence=existence+1 WHERE id=:book\_id");»: Підготовлюється запит до бази даних для оновлення значення стовпця «existence» в таблиці «books».

16. «query.bindValue(":book\_id", id);»: Прив'язується значення «id» до плейсхолдера «:book\_id» в запиті.

17. «query.exec();»: Виконується запит до бази даних для оновлення запису.

18. «model3->select();»: Оновлюється модель даних «model3», щоб відобразити оновлені дані в таблиці.

19. «model2->select();»: Оновлюється модель даних «model2», щоб відобразити оновлені дані в таблиці.

Отже, цей метод дозволяє оновити статус записів про видані книги і кількість доступних примірників книг у відповідних таблицях бази даних, а також оновити моделі даних для відображення оновлених результатів у віджетах діалогового вікна «GiveBook».

Метод для повернення

Для повернення до основного вікно використовується окремий метод:

- void GiveBook::on\_pushButton\_5\_clicked()

Код:

void GiveBook::on\_pushButton\_5\_clicked()

{

accept();

}

При натисканні кнопки "Завершити" викликається цей метод, який підтверджує діалог і закриває його.

**3.3** **Створення об’єктів**

1. Основним об’єктом програми є «MainWindow».

Об'єкт «MainWindow» відноситься до класу «MainWindow» і є головним вікном програми. Він представляє основне вікно інтерфейсу користувача, з яким він взаємодіє при роботі з програмою.

Об'єкт «MainWindow» містить всі основні компоненти і елементи уявлення програми, такі як кнопки, тексти, таблиці, поля введення і т. д. Можна маніпулювати цими елементами для створення інтерактивного інтерфейсу користувача.

Об'єкт «MainWindow» також має певний набір методів, які виконуються при взаємодії з елементами уявлення. Наприклад, коли користувач натискає кнопку або вводить текст у поле введення, відповідний метод «on\_pushButton\_clicked()» або «on\_lineEdit\_textEdited()» виконується, і ви можете реагувати на ці події в коді цих методів.

Зазвичай об'єкт «MainWindow» створюється при запуску програми і відображається на екрані користувача як головне вікно програми. Можна налаштовувати його зовнішній вигляд, розташування елементів і взаємодію з ними за допомогою методів і властивостей, які надаються класом «MainWindow».

Загалом, об'єкт «MainWindow» є ключовим компонентом програм і надає основну рамку для реалізації функцій та інтерфейсу вашої програми.

1. «QItemSelectionModel»: Це об'єкт моделі вибору елементів. Він використовується для відстеження вибраних елементів у таблицях або спискових віджетах. За допомогою нього можна отримати вибрані елементи, їх індекси та здійснити відповідні дії з цими елементами.
2. «QModelIndexList»: Це список об'єктів «QModelIndex», які представляють індекси вибраних елементів. Використовується для отримання індексів вибраних елементів з моделі даних.
3. «QSqlQuery»: Це об'єкт запиту до бази даних. Він використовується для виконання запитів до бази даних, таких як вставка, вибірка, оновлення або видалення даних. Запити готуються з використанням підготовлених запитів з певними параметрами.
4. «QDate»: Це об'єкт, що представляє дату. Використовується для роботи з датами, таких як отримання поточної дати або представлення дати в певному форматі.
5. «QVariant»: Це об'єкт, який може зберігати значення різних типів даних. Використовується для отримання значень з моделей даних і передачі їх між різними компонентами програми.
6. «QList»: Це клас-контейнер, що представляє список елементів. Використовується для зберігання списку ідентифікаторів або інших значень для подальшої обробки.
7. Об'єкт «QSqlDatabase» використовується для взаємодії з базою даних. Він представляє підключення до бази даних і надає методи для виконання запитів і маніпулювання даними. «QSqlDatabase» дозволяє встановлювати параметри підключення до бази даних, такі як тип бази даних, ім'я сервера, порт, ім'я користувача, пароль тощо. Крім того, він надає можливість управління декількома підключеннями до різних баз даних.
8. «QSqlTableModel» є класом у фреймворку Qt, який надає зручний спосіб зв'язування бази даних з моделлю даних. Він дозволяє працювати з таблицями бази даних у вигляді моделі, що може бути використана для відображення та редагування даних у інтерфейсі користувача.
9. «QString» є класом у фреймворку Qt, який використовується для роботи з рядками символів. Він надає розширений функціонал для роботи з текстовими даними, забезпечуючи зручні методи для операцій з рядками.

«QString» дозволяє зберігати, маніпулювати та отримувати доступ до рядків символів. Він підтримує різноманітні операції, такі як конкатенація рядків, вставка, видалення та заміна підрядків, отримання підрядка за індексом чи діапазоном, розділення рядка на підрядки і багато іншого.

1. «QRegularExpressionValidator» є класом у фреймворку Qt, який використовується для перевірки та валідації рядків тексту за допомогою регулярних виразів. Він дозволяє встановлювати регулярний вираз, який буде використовуватись для перевірки введеного користувачем тексту.

«QRegularExpressionValidator» забезпечує можливість встановлення валідаційного шаблону, в якому можна вказати правила для допустимого формату тексту. При введенні тексту в поле введення, валідатор перевіряє, чи відповідає введений текст заданому регулярному виразу. Якщо текст не задовольняє вимогам виразу, то поле введення буде відзначене як помилкове.

1. «QDialog» є класом у фреймворку Qt, який використовується для створення модальних або немодальних діалогових вікон. Він є базовим класом для створення власних діалогових вікон у програмі.

«QDialog» надає основний контейнер для розміщення вмісту діалогового вікна, таких як кнопки, поля введення, списки, таблиці та інші елементи керування. Він може містити інші вбудовані віджети, які відображаються у діалоговому вікні.

**3.4 Опис інтерфейсу програми**

Інтерфейс програми - це спосіб, яким користувач взаємодіє з програмою, щоб виконувати певні дії і отримувати результати. Описуючи інтерфейс програми, ми звертаємо увагу на розміщення елементів керування, їх типи, функціональність та способи взаємодії з користувачем.

Інтерфейс програми складається з наступних елементів:

1. Вікна - є основними контейнерами у програмі, які відображаються на екрані і надають користувачу можливість взаємодіяти з програмою. У фреймворку Qt, вікна реалізовані за допомогою класу «QMainWindow», а також інших класів, таких як «QDialog» для діалогових вікон.

«QMainWindow» є основним вікном програми. Воно має заголовок, панель меню, панель інструментів, робочу область та рядок стану. Це типовий шаблон для створення головного вікна додатка, яке містить основні елементи керування та відображає головні функції програми.

«QDialog» є класом для створення діалогових вікон. Діалогові вікна зазвичай використовуються для виконання конкретних завдань або відображення додаткової інформації. Вони можуть бути модальними або немодальними. Модальні діалоги блокують взаємодію з іншими вікнами, поки їх не буде закрито, тоді як немодальні діалоги дозволяють взаємодіяти з іншими вікнами програми.

Кожне вікно може містити інші вбудовані віджети, такі як кнопки, поля введення, списки, таблиці та інші елементи керування, які надають функціональність програми. Вікна можуть бути розміщені одне в середині іншого, утворюючи ієрархію вікон, яка відповідає структурі програми.

1. «QLineEdit» є класом у фреймворку Qt, який представляє поле введення однорядкового тексту. Воно дозволяє користувачу вводити або редагувати текстову інформацію.
2. «QLabel» є класом у фреймворку Qt, який використовується для відображення тексту або зображень на екрані. Він є статичним текстовим елементом, який не можна редагувати безпосередньо користувачем.
3. QPushButton є класом у фреймворку Qt, який представляє кнопку, що може бути натиснутою користувачем для виконання певних дій у програмі
4. QTableView є класом у фреймворку Qt, який використовується для відображення табличних даних у вигляді таблиці. Він надає можливість користувачу переглядати, редагувати та взаємодіяти з даними, відображеними у вигляді таблиці.
5. ComboBox є класом у фреймворку Qt, який надає випадаючий список для вибору одного елемента зі списку доступних значень.
6. QSpacerItem є класом у фреймворку Qt, який використовується для створення прогалин у макеті або між віджетами. Він дозволяє регулювати розташування та розмір прогалини у макеті.

**4 ТЕСТУВАННЯ ПРОГРАМИ І РЕЗУЛЬТАТИ ЇЇ ВИКОНАННЯ**

При запуску програми перше що бачить користувач це меню для роботи з книгами і кнопки для навігації між меню(Див. Рис. 4.1).

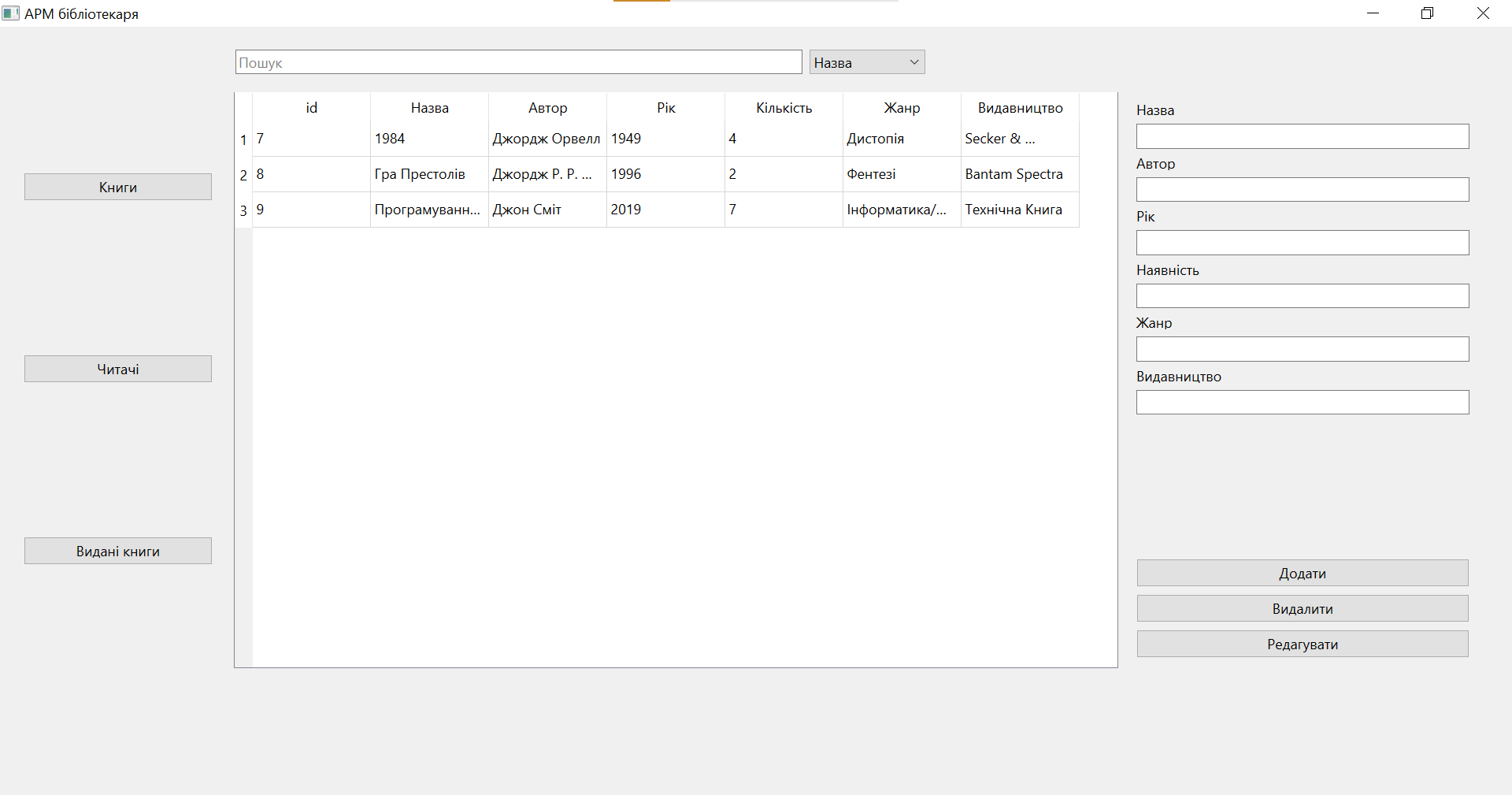


Рис. 4.1 Меню для роботи з книгами

В меню для роботи з книгами перше що бачить користувач це таблиця для перегляду даних про книги і вибору їх.

Також в меню є поле для пошуку і віджет для вибору колонки для пошуку(Див. Рис. 4.2).

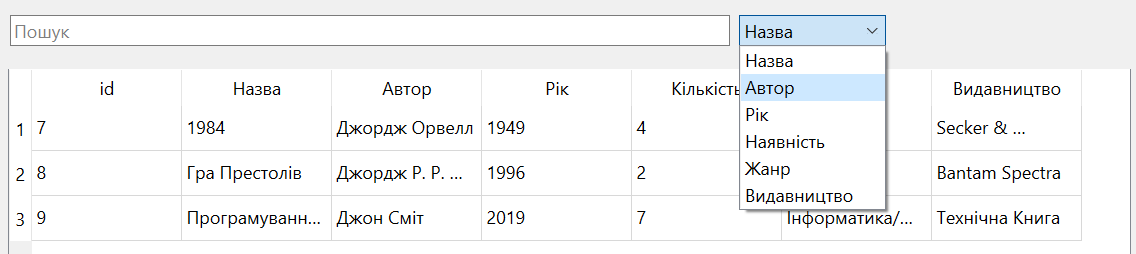
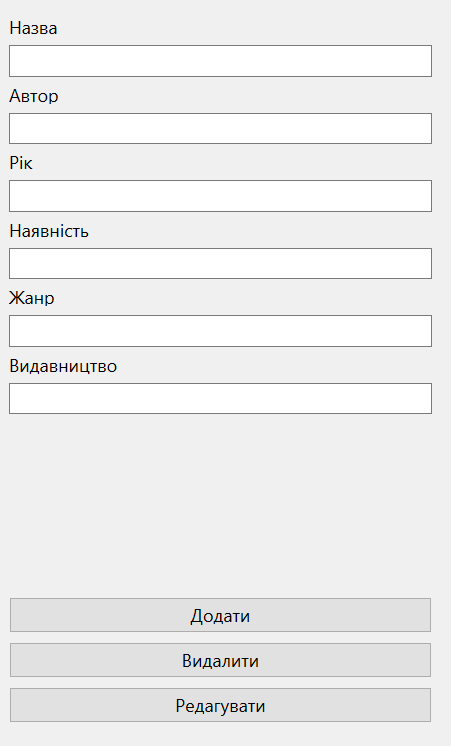


Рис. 4.2 Пошук книг

Правіше від таблиці є поля для введення інформації про книгу та три кнопки:

* Додати
* Видалити
* Редагувати

(Див. Рис. 4.3)

Рис. 4.3 Робота з книгами

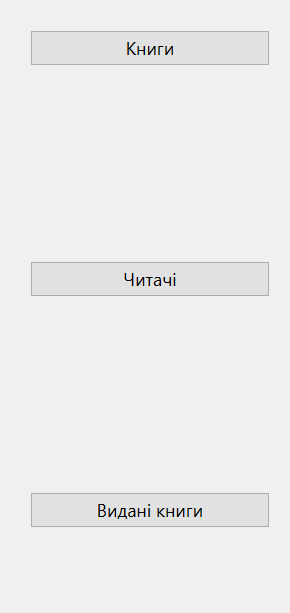
Зліва від таблиці є кнопки для навігації між меню. Перша кнопка зараз неактивна оскільки ми й так знаходимось на меню книг, друга кнопка перенесе нас на меню для роботи з читачами, а третя – з виданими книгами (Див. Рис. 4.4).

Рис. 4.4 Кнопки для навігації

При переході на меню з читачами кнопки для навігації залишаться такими самими а меню з книгами заміниться на меню з читачами (Див. Рис. 4.5).

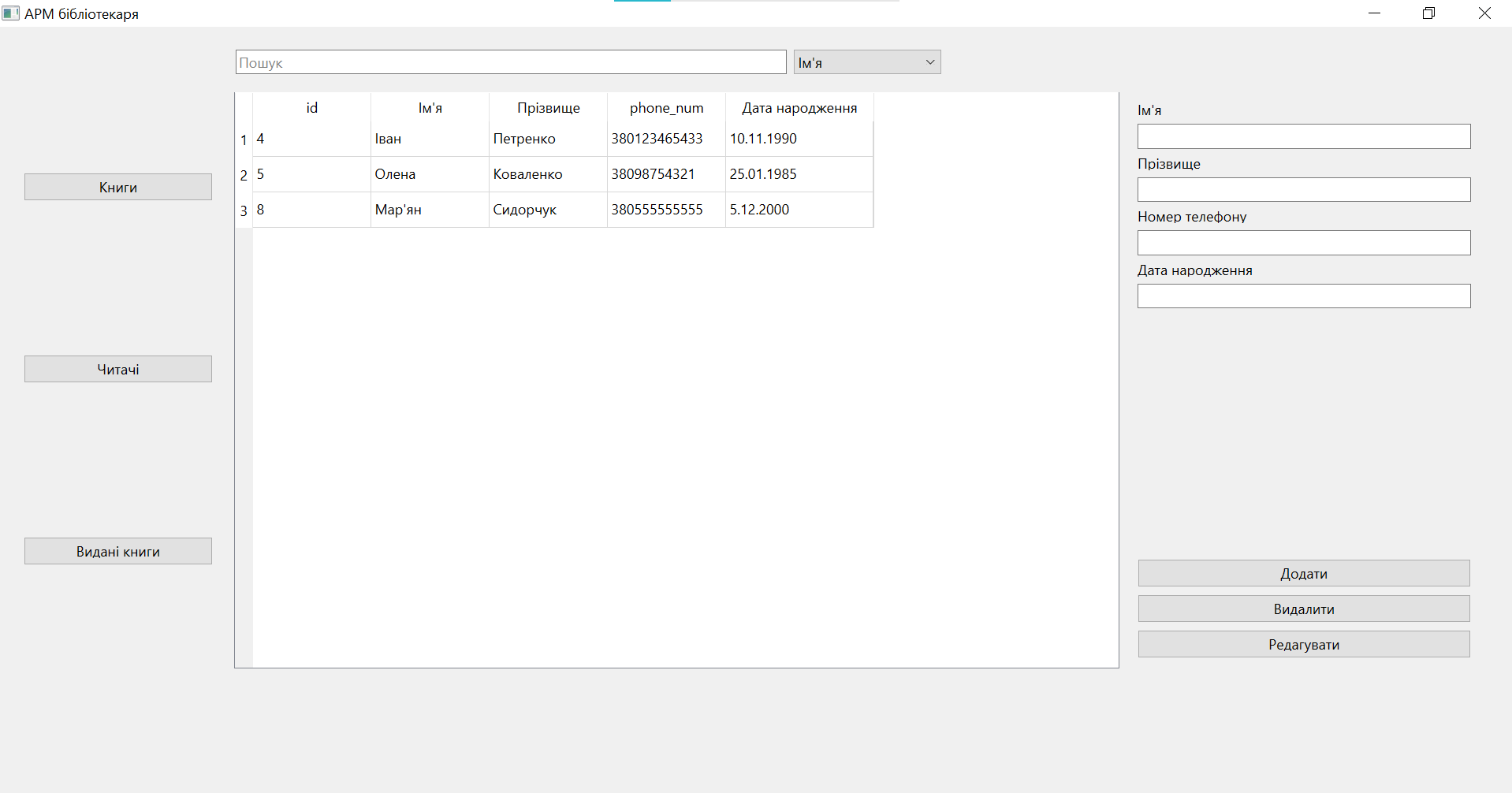


Рис. 4.5 Меню для роботи з читачами

Тут теж є таблиця і поле для пошуку (Див. Рис. 4.6).

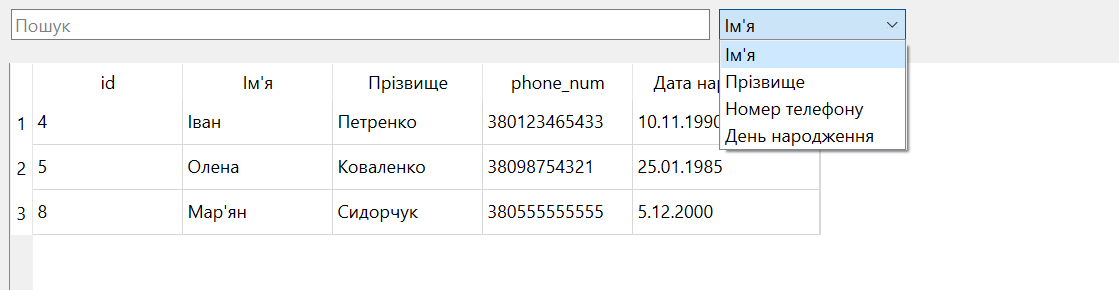


Рис. 4.6 Пошук читачів

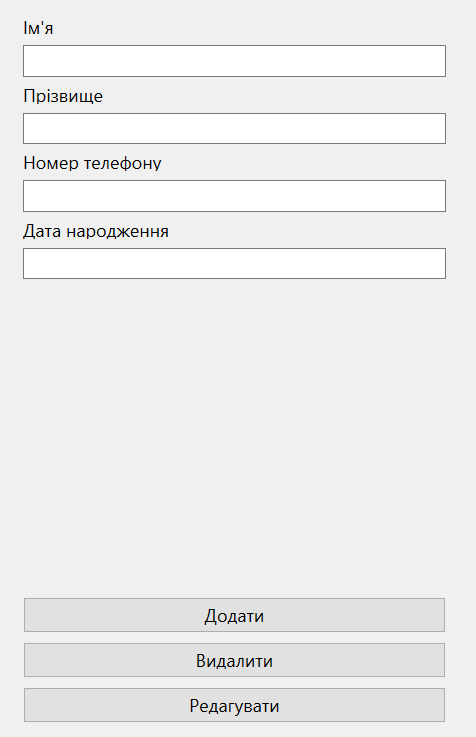
А також поля для введення і кнопки для додавання, видалення і редагування (Див. Рис. 4.7).

Рис. 4.7 Робота з читачами

При переході до меню з виданими книгами користувач бачить таблицю для перегляду виданих книг, поле для пошуку і кнопку для переходу до вікна з видачею і поверненням книг(Див. Рис. 4.8).

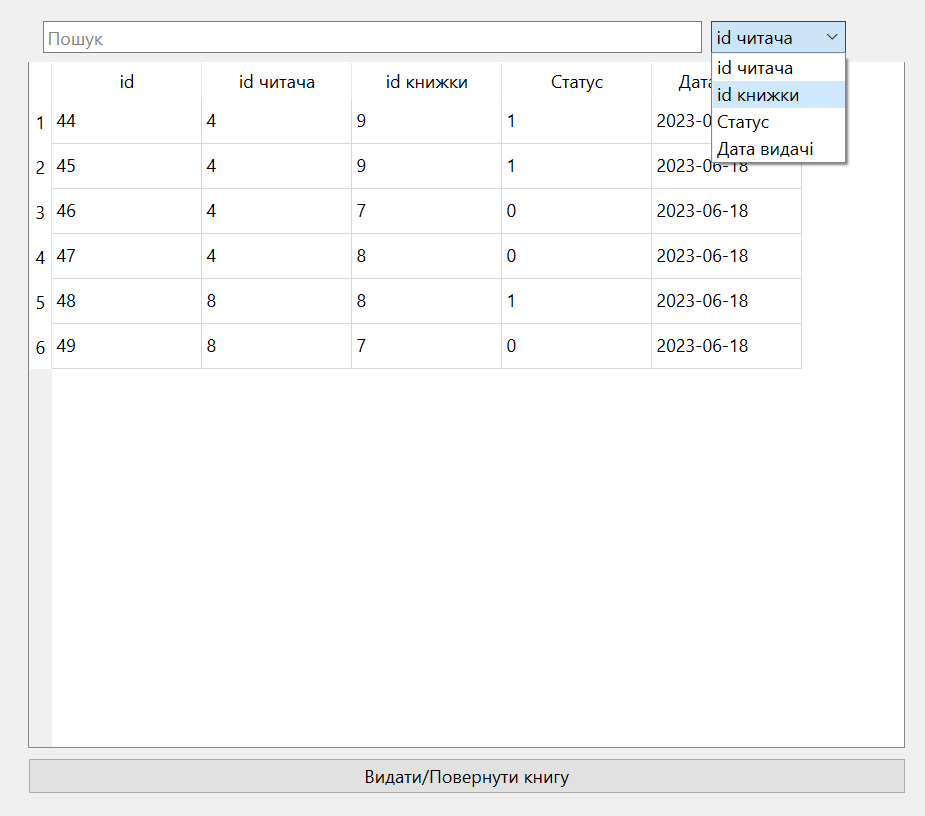


Рис. 4.8 Меню з виданими книгами

При відкритті вікна для видачі або повернення книг спочатку треба обрати читача(Див. Рис. 4.9).

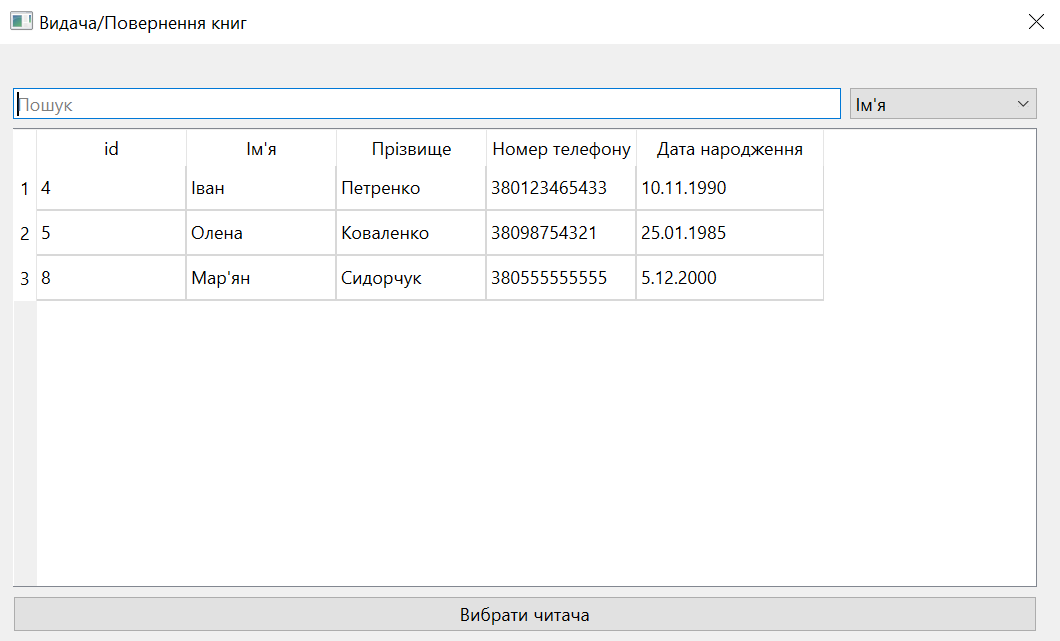


Рис. 4.9 Вибір читача

Далі з’явиться меню де можна вибрати пункт: видати книгу, повернути книгу і завершити(Див. Рис. 4.10).

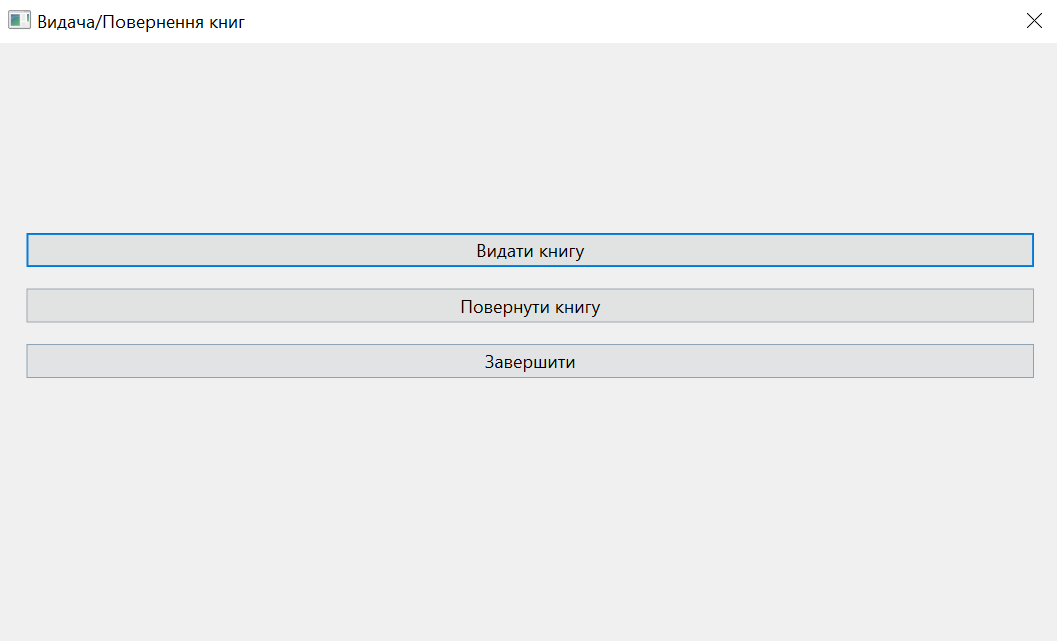


Рис. 4.10 Меню видачі/повернення книг

При виборі кнопки “Видати книгу” відкриється меню для видачі книг де необхідно вибрати книгу і натиснути відповідну кнопку. При завершенні видачі книг треба натиснути кнопку “Назад” яка поверне вас до попереднього меню(Див. Рис. 4.11).

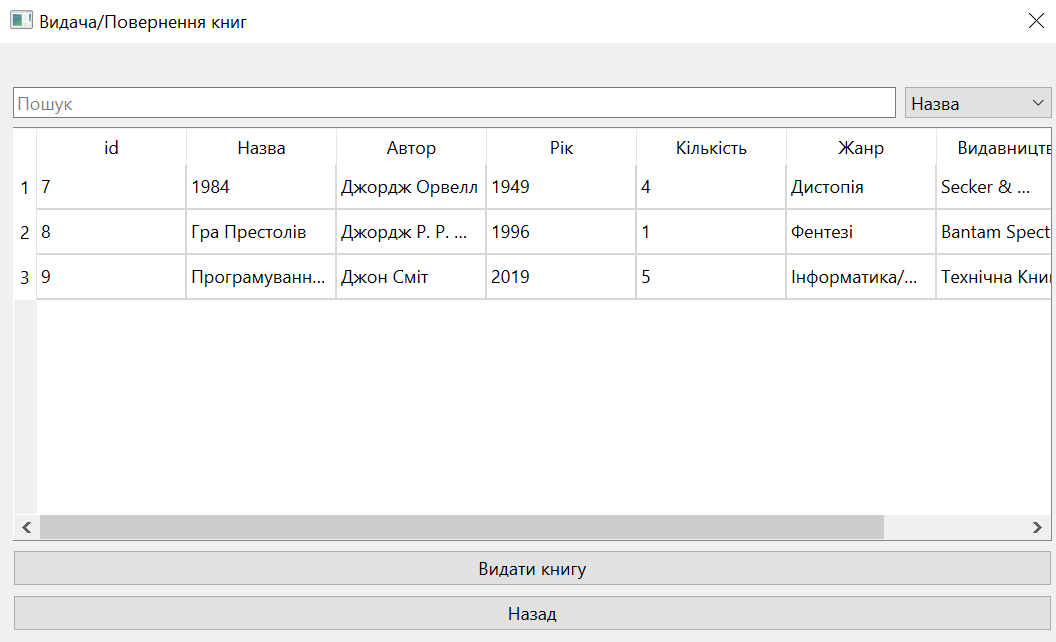


Рис. 4.11 Видача книг

При виборі кнопки “Повернути книгу” відкриється меню для повернення книг де необхідно вибрати одну або кілька книг і натиснути відповідну кнопку. При завершенні повернення книг треба натиснути кнопку “Назад” яка поверне вас до попереднього меню(Див. Рис. 4.12).

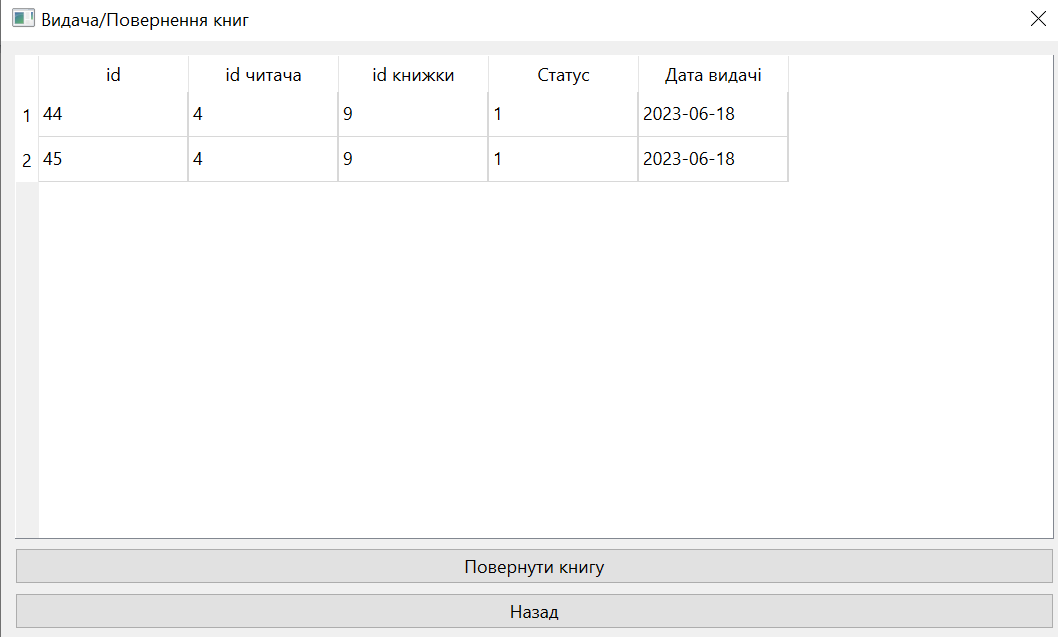


Рис. 4.12 Повернення книг

**ВИСНОВОК**

В ході розробки курсової роботи була створена АРМ бібліотекаря, яка дозволяє здійснювати керування книгами та читачами в бібліотеці. Для реалізації функціональності було використано базу даних і фреймворк Qt.

База даних була створена за допомогою SQLite, яка забезпечує зберігання і організацію даних про книги та читачів. В базі даних були визначені таблиці, такі як “books”, “readers” і “taken\_books” , які містять відповідну інформацію про книги, читачів і видані книги. Для доступу до бази даних і виконання запитів було використано клас «QSqlDatabase» та «QSqlQuery».

У програмі були реалізовані різні функції, такі як додавання нових книг та читачів, видалення записів, редагування інформації, пошук книг за різними критеріями та інше. Для відображення та взаємодії з даними були використані відповідні елементи у користувацькому інтерфейсі, такі як текстові поля («QLineEdit»), мітки («QLabel»), кнопки («QPushButton»), таблиці («QTableView»), комбіновані списки («QComboBox») та інші.

Користувач має змогу зручно керувати книгами та читачами, швидко знаходити необхідну інформацію та виконувати різні операції з даними. Програма забезпечує зручну та ефективну роботу бібліотекаря і сприяє покращенню управління бібліотекою.

**ЛІТЕРАТУРА**

1. Буда М.В. «Програмування на мові C++ з використанням фреймворку Qt.» - К.: Либідь, 2018.
2. Blanchette J., Summerfield M. «C++ GUI Programming with Qt 4.» - Prentice Hall, 2008.
3. Jasmin B., Sommerlad P. «Modern C++ Programming with Test-Driven Development: Code Better, Sleep Better.» - Pragmatic Bookshelf, 2013.

Посилання:

1. Офіційна документація Qt Framework. <https://doc.qt.io/>
2. Офіційна документація SQLite. <https://www.sqlite.org/docs.html>

**ДОДАТОК А**

**L.pro:**

QT += core gui sql

greaterThan(QT\_MAJOR\_VERSION, 4): QT += widgets

CONFIG += c++17

SOURCES += \

givebook.cpp \

main.cpp \

mainwindow.cpp

HEADERS += \

givebook.h \

mainwindow.h

FORMS += \

givebook.ui \

mainwindow.ui

# Default rules for deployment.

qnx: target.path = /tmp/$${TARGET}/bin

else: unix:!android: target.path = /opt/$${TARGET}/bin

!isEmpty(target.path): INSTALLS += target

**main.cpp:**

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.setWindowTitle("АРМ бібліотекаря");

w.show();

return a.exec();

}

**mainwindow.cpp:**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "givebook.h"

int id;

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

showMaximized();

db=QSqlDatabase::addDatabase("QSQLITE");

db.setDatabaseName("./Ldb.db");

db.open();

model = new QSqlTableModel(this,db);

model->setTable("books");

model->select();

ui->readersWidget->hide();

ui->TakenBooks->hide();

ui->tableView->setModel(model);

model2 = new QSqlTableModel(this,db);

model2->setTable("readers");

model2->select();

ui->tableView\_2->setModel(model2);

model3 = new QSqlTableModel(this,db);

model3->setTable("taken\_books");

model3->select();

ui->tableView\_3->setModel(model3);

ui->pushButton\_8->hide();

ui->pushButton\_9->hide();

ui->pushButton\_10->hide();

ui->AcceptResuctReader->hide();

model->setHeaderData(1, Qt::Horizontal, "Назва");

model->setHeaderData(2, Qt::Horizontal, "Автор");

model->setHeaderData(3, Qt::Horizontal, "Рік");

model->setHeaderData(4, Qt::Horizontal, "Кількість");

model->setHeaderData(5, Qt::Horizontal, "Жанр");

model->setHeaderData(6, Qt::Horizontal, "Видавництво");

model2->setHeaderData(1, Qt::Horizontal, "Ім'я");

model2->setHeaderData(2, Qt::Horizontal, "Прізвище");

//model2->setHeaderData(3, Qt::Horizontal, "Номер телефону");

model2->setHeaderData(4, Qt::Horizontal, "Дата народження");

ui->tableView\_2->setColumnWidth(4, 125);

model3->setHeaderData(1, Qt::Horizontal, "id читача");

model3->setHeaderData(2, Qt::Horizontal, "id книжки");

model3->setHeaderData(3, Qt::Horizontal, "Статус");

model3->setHeaderData(4, Qt::Horizontal, "Дата видачі");

QString pattern = "^(0?[1-9]|[12][0-9]|3[01])\\.(0?[1-9]|1[012])\\.\\d{4}$";

QRegularExpressionValidator\* validator = new QRegularExpressionValidator(QRegularExpression(pattern), this);

ui->birthday->setValidator(validator);

pattern = "^\\+?\\d{1,3}?[-.\\s]?\\(?\\d{3}\\)?[-.\\s]?\\d{3}[-.\\s]?\\d{3}$";

QRegularExpressionValidator\* validator2 = new QRegularExpressionValidator(QRegularExpression(pattern), this);

ui->phone\_num->setValidator(validator2);

QIntValidator\* validatorYear = new QIntValidator(1900, 2100, this);

ui->lineEdit\_2->setValidator(validatorYear);

QIntValidator\* validatorInt = new QIntValidator(1, std::numeric\_limits<int>::max(), this);

ui->lineEdit\_3->setValidator(validatorInt);

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::on\_pushButton\_4\_clicked()

{

QString name1 = ui->lineEdit->text();

int year = (ui->lineEdit\_2->text()).toInt();

QString author=ui->lineEdit\_8->text();

int existence = ui->lineEdit\_3->text().toInt();

QString genre = ui->lineEdit\_4->text();

QString edition = ui->lineEdit\_5->text();

QSqlQuery q1;

q1.prepare("INSERT INTO books (name,author, year, existence, genre, edition) VALUES (:name,:author, :year, :existence, :genre, :edition)");

q1.bindValue(":name", name1);

q1.bindValue(":year", year);

q1.bindValue(":author",author);

q1.bindValue(":existence", existence);

q1.bindValue(":genre", genre);

q1.bindValue(":edition", edition);

q1.exec();

model->select();

}

void MainWindow::on\_pushButton\_7\_clicked()

{

QItemSelectionModel\* selectionModel = ui->tableView->selectionModel();

QModelIndexList selectedIndexes = selectionModel->selectedIndexes();

QList<int> selectedIds;

foreach(const QModelIndex& index, selectedIndexes) {

int id = index.sibling(index.row(), 0).data().toInt();

selectedIds.append(id);

}

foreach(int id, selectedIds) {

QSqlQuery query;

query.prepare("DELETE FROM books WHERE id = :id");

query.bindValue(":id", id);

query.exec();

}

model->select();

}

void MainWindow::on\_pushButton\_6\_clicked()

{

QItemSelectionModel\* selectionModel = ui->tableView->selectionModel();

QModelIndexList selectedIndexes = selectionModel->selectedIndexes();

foreach(const QModelIndex& index, selectedIndexes) {

id = index.sibling(index.row(), 0).data().toInt();

}

QSqlQuery query;

query.prepare("SELECT \* FROM books WHERE id = :id");

query.bindValue(":id", id);

query.exec();

if (query.next()) {

ui->lineEdit->setText(query.value("name").toString());

ui->lineEdit\_2->setText(query.value("year").toString());

ui->lineEdit\_8->setText(query.value("author").toString());

ui->lineEdit\_3->setText(query.value("existence").toString());

ui->lineEdit\_4->setText(query.value("genre").toString());

ui->lineEdit\_5->setText(query.value("edition").toString());

ui->pushButton\_8->show();

ui->pushButton\_9->show();

ui->pushButton\_4->hide();

ui->pushButton\_6->hide();

ui->pushButton\_7->hide();

}

}

void MainWindow::on\_pushButton\_8\_clicked()

{

QSqlQuery updateQuery;

updateQuery.prepare("UPDATE books SET name = :name,author=:author,year=:year,existence=:existence,genre=:genre,edition=:edition WHERE ID = :id");

updateQuery.bindValue(":id", id);

updateQuery.bindValue(":name", ui->lineEdit->text());

updateQuery.bindValue(":author", ui->lineEdit\_8->text());

updateQuery.bindValue(":year", ui->lineEdit\_2->text());

updateQuery.bindValue(":existence", ui->lineEdit\_3->text());

updateQuery.bindValue(":genre", ui->lineEdit\_4->text());

updateQuery.bindValue(":edition", ui->lineEdit\_5->text());

updateQuery.exec();

model->select();

ui->pushButton\_8->hide();

ui->pushButton\_9->hide();

ui->pushButton\_4->show();

ui->pushButton\_6->show();

ui->pushButton\_7->show();

ui->lineEdit->setText("");

ui->lineEdit\_2->setText("");

ui->lineEdit\_8->setText("");

ui->lineEdit\_3->setText("");

ui->lineEdit\_4->setText("");

ui->lineEdit\_5->setText("");

id=-1;

}

void MainWindow::on\_pushButton\_9\_clicked()

{

id=-1;

ui->pushButton\_8->hide();

ui->pushButton\_9->hide();

ui->lineEdit->setText("");

ui->lineEdit\_2->setText("");

ui->lineEdit\_8->setText("");

ui->lineEdit\_3->setText("");

ui->lineEdit\_4->setText("");

ui->lineEdit\_5->setText("");

ui->pushButton\_4->show();

ui->pushButton\_6->show();

ui->pushButton\_7->show();

model->select();

}

void MainWindow::on\_pushButton\_2\_clicked()

{

ui->readersWidget->show();

ui->booksWidget->hide();

ui->TakenBooks->hide();

}

void MainWindow::on\_pushButton\_clicked()

{

ui->readersWidget->hide();

ui->booksWidget->show();

ui->TakenBooks->hide();

}

void MainWindow::on\_pushButton\_3\_clicked()

{

ui->TakenBooks->show();

ui->readersWidget->hide();

ui->booksWidget->hide();

}

void MainWindow::on\_pushButton\_5\_clicked()

{

QString first\_name = ui->first\_name->text();

QString last\_name = ui->last\_name->text();

QString phone\_num = (ui->phone\_num->text());

QString birthday = ui->birthday->text();

QSqlQuery q1;

q1.prepare("INSERT INTO readers VALUES (NULL,:first\_name, :last\_name,:phone\_num,:birthday)");

q1.bindValue(":first\_name", first\_name);

q1.bindValue(":last\_name", last\_name);

q1.bindValue(":phone\_num", phone\_num);

q1.bindValue(":birthday", birthday);

q1.exec();

model2->select();

}

void MainWindow::on\_lineEdit\_6\_textEdited(const QString &arg1)

{

QString columnName;

switch (ui->comboBox\_findBook->currentIndex()) {

case 0:

columnName="name";

break;

case 1:

columnName="author";

break;

case 2:

columnName="year";

break;

case 3:

columnName="existence";

break;

case 4:

columnName="genre";

break;

case 5:

columnName="edition";

break;

default:

break;

}

model->setFilter(""+columnName+" LIKE '%"+ui->lineEdit\_6->text()+"%'");

}

void MainWindow::on\_deleteReader\_clicked()

{

QItemSelectionModel\* selectionModel = ui->tableView\_2->selectionModel();

QModelIndexList selectedIndexes = selectionModel->selectedIndexes();

QList<int> selectedIds;

foreach(const QModelIndex& index, selectedIndexes) {

int id = index.sibling(index.row(), 0).data().toInt();

selectedIds.append(id);

}

foreach(int id, selectedIds) {

QSqlQuery query;

query.prepare("DELETE FROM readers WHERE id = :id");

query.bindValue(":id", id);

query.exec();

}

model2->select();

}

void MainWindow::on\_reductReader\_clicked()

{

QItemSelectionModel\* selectionModel = ui->tableView\_2->selectionModel();

QModelIndexList selectedIndexes = selectionModel->selectedIndexes();

foreach(const QModelIndex& index, selectedIndexes) {

id = index.sibling(index.row(), 0).data().toInt();

}

QSqlQuery query;

query.prepare("SELECT \* FROM readers WHERE id = :id");

query.bindValue(":id", id);

query.exec();

if (query.next()) {

ui->first\_name->setText(query.value("first\_name").toString());

ui->last\_name->setText(query.value("last\_name").toString());

ui->birthday->setText(query.value("birthday").toString());

ui->phone\_num->setText(query.value("phone\_num").toString());

ui->AcceptResuctReader->show();

ui->pushButton\_10->show();

ui->pushButton\_5->hide();

ui->deleteReader->hide();

ui->reductReader->hide();

}

}

void MainWindow::on\_AcceptResuctReader\_clicked()

{

QSqlQuery updateQuery;

updateQuery.prepare("UPDATE readers SET first\_name = :first\_name,last\_name=:last\_name,birthday=:birthday,phone\_num=:phone\_num WHERE id = :id");

updateQuery.bindValue(":id", id);

updateQuery.bindValue(":first\_name", ui->first\_name->text());

updateQuery.bindValue(":last\_name", ui->last\_name->text());

updateQuery.bindValue(":birthday", ui->birthday->text());

updateQuery.bindValue(":phone\_num", ui->phone\_num->text());

updateQuery.exec();

model2->select();

ui->AcceptResuctReader->hide();

ui->pushButton\_10->hide();

ui->pushButton\_5->show();

ui->deleteReader->show();

ui->reductReader->show();

ui->first\_name->setText("");

ui->last\_name->setText("");

ui->birthday->setText("");

ui->phone\_num->setText("");

id=-1;

}

void MainWindow::on\_pushButton\_10\_clicked()

{

id=-1;

model2->select();

ui->AcceptResuctReader->hide();

ui->pushButton\_10->hide();

ui->pushButton\_5->show();

ui->deleteReader->show();

ui->reductReader->show();

ui->first\_name->setText("");

ui->last\_name->setText("");

ui->birthday->setText("");

ui->phone\_num->setText("");

}

void MainWindow::on\_lineEdit\_7\_textEdited(const QString &arg1)

{

QString columnName;

switch (ui->comboBox\_2->currentIndex()) {

case 0:

columnName="first\_name";

break;

case 1:

columnName="last\_name";

break;

case 2:

columnName="phone\_num";

break;

case 3:

columnName="birthday";

break;

default:

break;

}

model2->setFilter(""+columnName+" LIKE '%"+ui->lineEdit\_7->text()+"%'");

}

void MainWindow::on\_giveBook\_clicked()

{

GiveBook w;

w.setModal(true);

w.setWindowTitle("Видача/Повернення книг");

w.exec();

model->select();

model3->select();

}

void MainWindow::on\_lineEdit\_10\_textEdited(const QString &arg1)

{

QString columnName;

switch (ui->comboBox\_4->currentIndex()) {

case 0:

columnName="reader\_id";

break;

case 1:

columnName="book\_id";

break;

case 2:

columnName="status";

break;

case 3:

columnName="date";

break;

default:

break;

}

model3->setFilter(""+columnName+" LIKE '%"+ui->lineEdit\_10->text()+"%'");

}

**mainwindow.h:**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QSqlDatabase>

#include <QDebug>

#include <QSqlQuery>

#include <QSqlTableModel>

#include <QItemDelegate>

#include <QStyledItemDelegate>

#include <QValidator>

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private slots:

void on\_pushButton\_4\_clicked();

void on\_pushButton\_2\_clicked();

void on\_pushButton\_clicked();

void on\_pushButton\_5\_clicked();

void on\_pushButton\_7\_clicked();

void on\_pushButton\_6\_clicked();

void on\_pushButton\_8\_clicked();

void on\_lineEdit\_6\_textEdited(const QString &arg1);

void on\_deleteReader\_clicked();

void on\_reductReader\_clicked();

void on\_AcceptResuctReader\_clicked();

void on\_lineEdit\_7\_textEdited(const QString &arg1);

void on\_pushButton\_3\_clicked();

void on\_giveBook\_clicked();

void on\_pushButton\_9\_clicked();

void on\_pushButton\_10\_clicked();

void on\_lineEdit\_10\_textEdited(const QString &arg1);

private:

Ui::MainWindow \*ui;

QSqlDatabase db;

QSqlQuery \*query;

QSqlTableModel \*model;

QSqlTableModel \*model2;

QSqlTableModel \*model3;

};

#endif // MAINWINDOW\_H

**mainwindow.ui:**

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>MainWindow</class>

<widget class="QMainWindow" name="MainWindow">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>1277</width>

<height>600</height>

</rect>

</property>

<property name="sizePolicy">

<sizepolicy hsizetype="Maximum" vsizetype="Maximum">

<horstretch>0</horstretch>

<verstretch>0</verstretch>

</sizepolicy>

</property>

<property name="windowTitle">

<string>MainWindow</string>

</property>

<widget class="QWidget" name="centralwidget">

<widget class="QWidget" name="verticalLayoutWidget">

<property name="geometry">

<rect>

<x>20</x>

<y>-1</y>

<width>160</width>

<height>581</height>

</rect>

</property>

<layout class="QVBoxLayout" name="verticalLayout">

<item>

<widget class="QPushButton" name="pushButton">

<property name="text">

<string>Книги</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="pushButton\_2">

<property name="text">

<string>Читачі</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="pushButton\_3">

<property name="text">

<string>Видані книги</string>

</property>

</widget>

</item>

</layout>

</widget>

<widget class="QWidget" name="booksWidget" native="true">

<property name="enabled">

<bool>true</bool>

</property>

<property name="geometry">

<rect>

<x>180</x>

<y>10</y>

<width>1091</width>

<height>551</height>

</rect>

</property>

<layout class="QVBoxLayout" name="verticalLayout\_5">

<item>

<layout class="QHBoxLayout" name="horizontalLayout\_findBook">

<item>

<spacer name="horizontalSpacer\_2">

<property name="orientation">

<enum>Qt::Horizontal</enum>

</property>

<property name="sizeType">

<enum>QSizePolicy::Minimum</enum>

</property>

<property name="sizeHint" stdset="0">

<size>

<width>10</width>

<height>20</height>

</size>

</property>

</spacer>

</item>

<item>

<widget class="QLineEdit" name="lineEdit\_6">

<property name="autoFillBackground">

<bool>false</bool>

</property>

<property name="text">

<string/>

</property>

<property name="placeholderText">

<string>Пошук</string>

</property>

</widget>

</item>

<item>

<widget class="QComboBox" name="comboBox\_findBook">

<item>

<property name="text">

<string>Назва</string>

</property>

</item>

<item>

<property name="text">

<string>Автор</string>

</property>

</item>

<item>

<property name="text">

<string>Рік</string>

</property>

</item>

<item>

<property name="text">

<string>Наявність</string>

</property>

</item>

<item>

<property name="text">

<string>Жанр</string>

</property>

</item>

<item>

<property name="text">

<string>Видавництво</string>

</property>

</item>

</widget>

</item>

<item>

<spacer name="horizontalSpacer">

<property name="orientation">

<enum>Qt::Horizontal</enum>

</property>

<property name="sizeType">

<enum>QSizePolicy::Expanding</enum>

</property>

<property name="sizeHint" stdset="0">

<size>

<width>40</width>

<height>20</height>

</size>

</property>

</spacer>

</item>

</layout>

</item>

<item>

<widget class="QWidget" name="horizontalWidget" native="true">

<property name="enabled">

<bool>true</bool>

</property>

<property name="sizePolicy">

<sizepolicy hsizetype="Minimum" vsizetype="MinimumExpanding">

<horstretch>0</horstretch>

<verstretch>0</verstretch>

</sizepolicy>

</property>

<layout class="QHBoxLayout" name="horizontalLayout">

<item>

<layout class="QVBoxLayout" name="verticalLayout\_4"/>

</item>

<item>

<widget class="QTableView" name="tableView">

<property name="enabled">

<bool>true</bool>

</property>

<property name="sizePolicy">

<sizepolicy hsizetype="Minimum" vsizetype="Preferred">

<horstretch>5</horstretch>

<verstretch>0</verstretch>

</sizepolicy>

</property>

</widget>

</item>

<item alignment="Qt::AlignTop">

<widget class="QWidget" name="inputB" native="true">

<property name="sizePolicy">

<sizepolicy hsizetype="MinimumExpanding" vsizetype="MinimumExpanding">

<horstretch>2</horstretch>

<verstretch>0</verstretch>

</sizepolicy>

</property>

<layout class="QVBoxLayout" name="verticalLayout\_2">

<item>

<widget class="QLabel" name="label">

<property name="maximumSize">

<size>

<width>16777215</width>

<height>12</height>

</size>

</property>

<property name="text">

<string>Назва</string>

</property>

</widget>

</item>

<item>

<widget class="QLineEdit" name="lineEdit"/>

</item>

<item>

<widget class="QLabel" name="label\_10">

<property name="maximumSize">

<size>

<width>16777215</width>

<height>12</height>

</size>

</property>

<property name="text">

<string>Автор</string>

</property>

</widget>

</item>

<item>

<widget class="QLineEdit" name="lineEdit\_8"/>

</item>

<item>

<widget class="QLabel" name="label\_2">

<property name="maximumSize">

<size>

<width>16777215</width>

<height>12</height>

</size>

</property>

<property name="text">

<string>Рік</string>

</property>

</widget>

</item>

<item alignment="Qt::AlignTop">

<widget class="QLineEdit" name="lineEdit\_2">

<property name="inputMethodHints">

<set>Qt::ImhDigitsOnly|Qt::ImhFormattedNumbersOnly</set>

</property>

</widget>

</item>

<item>

<widget class="QLabel" name="label\_3">

<property name="maximumSize">

<size>

<width>16777215</width>

<height>12</height>

</size>

</property>

<property name="text">

<string>Наявність</string>

</property>

</widget>

</item>

<item>

<widget class="QLineEdit" name="lineEdit\_3"/>

</item>

<item>

<widget class="QLabel" name="label\_4">

<property name="maximumSize">

<size>

<width>16777215</width>

<height>12</height>

</size>

</property>

<property name="text">

<string>Жанр</string>

</property>

</widget>

</item>

<item>

<widget class="QLineEdit" name="lineEdit\_4"/>

</item>

<item>

<widget class="QLabel" name="label\_5">

<property name="maximumSize">

<size>

<width>16777215</width>

<height>12</height>

</size>

</property>

<property name="text">

<string>Видавництво</string>

</property>

</widget>

</item>

<item>

<widget class="QLineEdit" name="lineEdit\_5"/>

</item>

<item>

<spacer name="verticalSpacer">

<property name="orientation">

<enum>Qt::Vertical</enum>

</property>

<property name="sizeHint" stdset="0">

<size>

<width>20</width>

<height>200</height>

</size>

</property>

</spacer>

</item>

<item>

<widget class="QPushButton" name="pushButton\_4">

<property name="text">

<string>Додати</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="pushButton\_7">

<property name="text">

<string>Видалити</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="pushButton\_6">

<property name="text">

<string>Редагувати</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="pushButton\_8">

<property name="text">

<string>Підтвердити</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="pushButton\_9">

<property name="text">

<string>Скасувати</string>

</property>

</widget>

</item>

</layout>

</widget>

</item>

</layout>

<zorder>inputB</zorder>

<zorder>tableView</zorder>

</widget>

</item>

</layout>

</widget>

<widget class="QWidget" name="readersWidget" native="true">

<property name="enabled">

<bool>true</bool>

</property>

<property name="geometry">

<rect>

<x>180</x>

<y>10</y>

<width>1091</width>

<height>551</height>

</rect>

</property>

<layout class="QVBoxLayout" name="verticalLayout\_6">

<item>

<layout class="QHBoxLayout" name="horizontalLayout\_4">

<item>

<spacer name="horizontalSpacer\_3">

<property name="orientation">

<enum>Qt::Horizontal</enum>

</property>

<property name="sizeType">

<enum>QSizePolicy::Minimum</enum>

</property>

<property name="sizeHint" stdset="0">

<size>

<width>10</width>

<height>20</height>

</size>

</property>

</spacer>

</item>

<item>

<widget class="QLineEdit" name="lineEdit\_7">

<property name="placeholderText">

<string>Пошук</string>

</property>

</widget>

</item>

<item>

<widget class="QComboBox" name="comboBox\_2">

<item>

<property name="text">

<string>Ім'я</string>

</property>

</item>

<item>

<property name="text">

<string>Прізвище</string>

</property>

</item>

<item>

<property name="text">

<string>Номер телефону</string>

</property>

</item>

<item>

<property name="text">

<string>День народження</string>

</property>

</item>

</widget>

</item>

<item>

<spacer name="horizontalSpacer\_4">

<property name="orientation">

<enum>Qt::Horizontal</enum>

</property>

<property name="sizeHint" stdset="0">

<size>

<width>40</width>

<height>20</height>

</size>

</property>

</spacer>

</item>

</layout>

</item>

<item>

<widget class="QWidget" name="horizontalWidget\_2" native="true">

<property name="enabled">

<bool>true</bool>

</property>

<property name="sizePolicy">

<sizepolicy hsizetype="Minimum" vsizetype="MinimumExpanding">

<horstretch>0</horstretch>

<verstretch>0</verstretch>

</sizepolicy>

</property>

<layout class="QHBoxLayout" name="horizontalLayout\_2">

<item>

<widget class="QTableView" name="tableView\_2">

<property name="enabled">

<bool>true</bool>

</property>

<property name="sizePolicy">

<sizepolicy hsizetype="Minimum" vsizetype="Minimum">

<horstretch>5</horstretch>

<verstretch>0</verstretch>

</sizepolicy>

</property>

</widget>

</item>

<item alignment="Qt::AlignTop">

<widget class="QWidget" name="inputB\_2" native="true">

<property name="sizePolicy">

<sizepolicy hsizetype="MinimumExpanding" vsizetype="MinimumExpanding">

<horstretch>2</horstretch>

<verstretch>0</verstretch>

</sizepolicy>

</property>

<layout class="QVBoxLayout" name="verticalLayout\_3">

<item>

<widget class="QLabel" name="label\_6">

<property name="maximumSize">

<size>

<width>16777215</width>

<height>12</height>

</size>

</property>

<property name="text">

<string>Ім'я</string>

</property>

</widget>

</item>

<item>

<widget class="QLineEdit" name="first\_name"/>

</item>

<item>

<widget class="QLabel" name="label\_7">

<property name="maximumSize">

<size>

<width>16777215</width>

<height>12</height>

</size>

</property>

<property name="text">

<string>Прізвище</string>

</property>

</widget>

</item>

<item>

<widget class="QLineEdit" name="last\_name"/>

</item>

<item>

<widget class="QLabel" name="label\_8">

<property name="maximumSize">

<size>

<width>16777215</width>

<height>12</height>

</size>

</property>

<property name="text">

<string>Номер телефону</string>

</property>

</widget>

</item>

<item>

<widget class="QLineEdit" name="phone\_num"/>

</item>

<item>

<widget class="QLabel" name="label\_9">

<property name="maximumSize">

<size>

<width>16777215</width>

<height>12</height>

</size>

</property>

<property name="text">

<string>Дата народження</string>

</property>

</widget>

</item>

<item>

<widget class="QLineEdit" name="birthday"/>

</item>

<item>

<spacer name="verticalSpacer\_2">

<property name="orientation">

<enum>Qt::Vertical</enum>

</property>

<property name="sizeHint" stdset="0">

<size>

<width>20</width>

<height>250</height>

</size>

</property>

</spacer>

</item>

<item>

<widget class="QPushButton" name="pushButton\_5">

<property name="text">

<string>Додати</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="deleteReader">

<property name="text">

<string>Видалити</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="reductReader">

<property name="text">

<string>Редагувати</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="AcceptResuctReader">

<property name="text">

<string>Підтвердити</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="pushButton\_10">

<property name="text">

<string>Скасувати</string>

</property>

</widget>

</item>

</layout>

</widget>

</item>

</layout>

</widget>

</item>

</layout>

</widget>

<widget class="QWidget" name="TakenBooks" native="true">

<property name="enabled">

<bool>true</bool>

</property>

<property name="geometry">

<rect>

<x>180</x>

<y>10</y>

<width>621</width>

<height>551</height>

</rect>

</property>

<layout class="QVBoxLayout" name="verticalLayout\_7">

<item>

<widget class="QWidget" name="horizontalWidget\_3" native="true">

<property name="enabled">

<bool>true</bool>

</property>

<property name="sizePolicy">

<sizepolicy hsizetype="Minimum" vsizetype="MinimumExpanding">

<horstretch>0</horstretch>

<verstretch>0</verstretch>

</sizepolicy>

</property>

<layout class="QVBoxLayout" name="verticalLayout\_8">

<item>

<layout class="QHBoxLayout" name="horizontalLayout\_6">

<item>

<spacer name="horizontalSpacer\_7">

<property name="orientation">

<enum>Qt::Horizontal</enum>

</property>

<property name="sizeType">

<enum>QSizePolicy::Minimum</enum>

</property>

<property name="sizeHint" stdset="0">

<size>

<width>10</width>

<height>20</height>

</size>

</property>

</spacer>

</item>

<item>

<widget class="QLineEdit" name="lineEdit\_10">

<property name="autoFillBackground">

<bool>false</bool>

</property>

<property name="text">

<string/>

</property>

<property name="placeholderText">

<string>Пошук</string>

</property>

</widget>

</item>

<item>

<widget class="QComboBox" name="comboBox\_4">

<item>

<property name="text">

<string>id читача</string>

</property>

</item>

<item>

<property name="text">

<string>id книжки</string>

</property>

</item>

<item>

<property name="text">

<string>Статус</string>

</property>

</item>

<item>

<property name="text">

<string>Дата видачі</string>

</property>

</item>

</widget>

</item>

<item>

<spacer name="horizontalSpacer\_8">

<property name="orientation">

<enum>Qt::Horizontal</enum>

</property>

<property name="sizeType">

<enum>QSizePolicy::Fixed</enum>

</property>

<property name="sizeHint" stdset="0">

<size>

<width>40</width>

<height>20</height>

</size>

</property>

</spacer>

</item>

</layout>

</item>

<item>

<widget class="QTableView" name="tableView\_3">

<property name="enabled">

<bool>true</bool>

</property>

<property name="sizePolicy">

<sizepolicy hsizetype="Minimum" vsizetype="Minimum">

<horstretch>2</horstretch>

<verstretch>0</verstretch>

</sizepolicy>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="giveBook">

<property name="sizePolicy">

<sizepolicy hsizetype="Minimum" vsizetype="Fixed">

<horstretch>1</horstretch>

<verstretch>0</verstretch>

</sizepolicy>

</property>

<property name="text">

<string>Видати/Повернути книгу</string>

</property>

</widget>

</item>

</layout>

</widget>

</item>

</layout>

</widget>

<zorder>verticalLayoutWidget</zorder>

<zorder>readersWidget</zorder>

<zorder>booksWidget</zorder>

<zorder>TakenBooks</zorder>

</widget>

<widget class="QMenuBar" name="menubar">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>1277</width>

<height>17</height>

</rect>

</property>

</widget>

<widget class="QStatusBar" name="statusbar"/>

</widget>

<resources/>

<connections/>

</ui>

**givebook.cpp:**

#include "givebook.h"

#include "ui\_givebook.h"

int idReader;

GiveBook::GiveBook(QWidget \*parent) :

QDialog(parent),

ui(new Ui::GiveBook)

{

ui->setupUi(this);

db=QSqlDatabase::addDatabase("QSQLITE");

db.setDatabaseName("./Ldb.db");

db.open();

query = new QSqlQuery(db);

model = new QSqlTableModel(this,db);

model->setTable("readers");

model->select();

ui->tableView->setModel(model);

model2 = new QSqlTableModel(this,db);

model2->setTable("books");

model2->setFilter(" existence>0");

model2->select();

ui->tableView\_2->setModel(model2);

model3 = new QSqlTableModel(this,db);

model3->setTable("taken\_books");

model3->select();

ui->tableView\_3->setModel(model3);

ui->books->hide();

ui->choose->hide();

ui->returnBook->hide();

model2->setHeaderData(1, Qt::Horizontal, "Назва");

model2->setHeaderData(2, Qt::Horizontal, "Автор");

model2->setHeaderData(3, Qt::Horizontal, "Рік");

model2->setHeaderData(4, Qt::Horizontal, "Кількість");

model2->setHeaderData(5, Qt::Horizontal, "Жанр");

model2->setHeaderData(6, Qt::Horizontal, "Видавництво");

model->setHeaderData(1, Qt::Horizontal, "Ім'я");

model->setHeaderData(2, Qt::Horizontal, "Прізвище");

model->setHeaderData(3, Qt::Horizontal, "Номер телефону");

model->setHeaderData(4, Qt::Horizontal, "Дата народження");

ui->tableView->setColumnWidth(4, 125);

model3->setHeaderData(1, Qt::Horizontal, "id читача");

model3->setHeaderData(2, Qt::Horizontal, "id книжки");

model3->setHeaderData(3, Qt::Horizontal, "Статус");

model3->setHeaderData(4, Qt::Horizontal, "Дата видачі");

}

GiveBook::~GiveBook()

{

delete ui;

}

void GiveBook::on\_pushButton\_clicked()

{

QItemSelectionModel\* selectionModel = ui->tableView->selectionModel();

QModelIndexList selectedIndexes = selectionModel->selectedIndexes();

int id;

foreach(const QModelIndex& index, selectedIndexes) {

id = index.sibling(index.row(), 0).data().toInt();

}

QSqlQuery query;

query.prepare("Select \* FROM readers WHERE id = :id");

query.bindValue(":id", id);

query.exec();

ui->readers->hide();

ui->choose->show();

idReader=id;

model3->setFilter(" reader\_id="+QString::number(idReader)+" AND status=1");

model3->select();

}

void GiveBook::on\_pushButton\_2\_clicked()

{

QItemSelectionModel\* selectionModel = ui->tableView\_2->selectionModel();

QModelIndexList selectedIndexes = selectionModel->selectedIndexes();

int id;

foreach(const QModelIndex& index, selectedIndexes) {

id = index.sibling(index.row(), 0).data().toInt();

}

QSqlQuery q1,q2;

QDate currentDate = QDate::currentDate();

q1.prepare("INSERT INTO taken\_books VALUES (NULL,:reader\_id,:book\_id,1,:date)");

q1.bindValue(":reader\_id", idReader);

q1.bindValue(":book\_id", id);

q1.bindValue(":date", currentDate);

q1.exec();

q2.prepare("UPDATE books SET existence=existence-1 WHERE id=:book\_id");

q2.bindValue(":book\_id", id);

q2.exec();

model2->select();

model3->select();

}

void GiveBook::on\_pushButton\_5\_clicked()

{

accept();

}

void GiveBook::on\_pushButton\_3\_clicked()

{

ui->choose->hide();

ui->books->show();

}

void GiveBook::on\_pushButton\_4\_clicked()

{

ui->choose->hide();

ui->returnBook->show();

}

void GiveBook::on\_lineEdit\_7\_textEdited(const QString &arg1)

{

QString columnName;

switch (ui->comboBox\_3->currentIndex()) {

case 0:

columnName="first\_name";

break;

case 1:

columnName="last\_name";

break;

case 2:

columnName="phone\_num";

break;

case 3:

columnName="birthday";

break;

default:

break;

}

model->setFilter(""+columnName+" LIKE '%"+ui->lineEdit\_7->text()+"%'");

}

void GiveBook::on\_lineEdit\_8\_textEdited(const QString &arg1)

{

QString columnName;

switch (ui->comboBox\_2->currentIndex()) {

case 0:

columnName="name";

break;

case 1:

columnName="author";

break;

case 2:

columnName="year";

break;

case 3:

columnName="existence";

break;

case 4:

columnName="genre";

break;

case 5:

columnName="edition";

break;

default:

break;

}

model2->setFilter(""+columnName+" LIKE '%"+ui->lineEdit\_8->text()+"%' AND existence>0");

}

void GiveBook::on\_pushButton\_6\_clicked()

{

QItemSelectionModel\* selectionModel = ui->tableView\_3->selectionModel();

QModelIndexList selectedIndexes = selectionModel->selectedIndexes();

QList<int> selectedIds;

foreach(const QModelIndex& index, selectedIndexes) {

int id = index.sibling(index.row(), 0).data().toInt();

selectedIds.append(id);

}

foreach(int id, selectedIds) {

QSqlQuery query;

query.prepare("UPDATE taken\_books SET status=0 WHERE id = :id");

query.bindValue(":id", id);

query.exec();

}

foreach (QModelIndex i, selectedIndexes) {

QVariant value = model3->data(model3->index(i.row(), 2));

QString id = value.toString();

QSqlQuery query;

query.prepare("UPDATE books SET existence=existence+1 WHERE id=:book\_id");

query.bindValue(":book\_id", id);

query.exec();

}

model3->select();

model2->select();

}

void GiveBook::on\_back2\_clicked()

{

ui->returnBook->hide();

ui->choose->show();

}

void GiveBook::on\_back\_clicked()

{

ui->books->hide();

ui->choose->show();

}

**givebook.h:**

#ifndef GIVEBOOK\_H

#define GIVEBOOK\_H

#include <QDialog>

#include <QSqlDatabase>

#include <QDebug>

#include <QSqlQuery>

#include <QSqlTableModel>

#include <QDateTime>

namespace Ui {

class GiveBook;

}

class GiveBook : public QDialog

{

Q\_OBJECT

public:

explicit GiveBook(QWidget \*parent = nullptr);

~GiveBook();

private slots:

void on\_pushButton\_clicked();

void on\_pushButton\_2\_clicked();

void on\_pushButton\_5\_clicked();

void on\_pushButton\_3\_clicked();

void on\_lineEdit\_7\_textEdited(const QString &arg1);

void on\_lineEdit\_8\_textEdited(const QString &arg1);

void on\_pushButton\_4\_clicked();

void on\_pushButton\_6\_clicked();

void on\_back2\_clicked();

void on\_back\_clicked();

private:

Ui::GiveBook \*ui;

QSqlDatabase db;

QSqlQuery \*query;

QSqlTableModel \*model;

QSqlTableModel \*model2;

QSqlTableModel \*model3;

};

#endif // GIVEBOOK\_H

**givebook.ui:**

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>GiveBook</class>

<widget class="QDialog" name="GiveBook">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>708</width>

<height>402</height>

</rect>

</property>

<property name="windowTitle">

<string>Dialog</string>

</property>

<widget class="QWidget" name="books" native="true">

<property name="enabled">

<bool>true</bool>

</property>

<property name="geometry">

<rect>

<x>0</x>

<y>20</y>

<width>711</width>

<height>381</height>

</rect>

</property>

<layout class="QVBoxLayout" name="verticalLayout\_2">

<item>

<layout class="QHBoxLayout" name="horizontalLayout\_5">

<item>

<widget class="QLineEdit" name="lineEdit\_8">

<property name="autoFillBackground">

<bool>false</bool>

</property>

<property name="text">

<string/>

</property>

<property name="placeholderText">

<string>Пошук</string>

</property>

</widget>

</item>

<item>

<widget class="QComboBox" name="comboBox\_2">

<item>

<property name="text">

<string>Назва</string>

</property>

</item>

<item>

<property name="text">

<string>Автор</string>

</property>

</item>

<item>

<property name="text">

<string>Рік</string>

</property>

</item>

<item>

<property name="text">

<string>Наявність</string>

</property>

</item>

<item>

<property name="text">

<string>Жанр</string>

</property>

</item>

<item>

<property name="text">

<string>Видавництво</string>

</property>

</item>

</widget>

</item>

</layout>

</item>

<item>

<widget class="QTableView" name="tableView\_2"/>

</item>

<item>

<widget class="QPushButton" name="pushButton\_2">

<property name="text">

<string>Видати книгу</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="back">

<property name="text">

<string>Назад</string>

</property>

</widget>

</item>

</layout>

</widget>

<widget class="QWidget" name="readers" native="true">

<property name="geometry">

<rect>

<x>0</x>

<y>20</y>

<width>701</width>

<height>381</height>

</rect>

</property>

<layout class="QVBoxLayout" name="verticalLayout">

<item>

<layout class="QHBoxLayout" name="horizontalLayout\_4">

<item>

<widget class="QLineEdit" name="lineEdit\_7">

<property name="autoFillBackground">

<bool>false</bool>

</property>

<property name="text">

<string/>

</property>

<property name="placeholderText">

<string>Пошук</string>

</property>

</widget>

</item>

<item>

<widget class="QComboBox" name="comboBox\_3">

<item>

<property name="text">

<string>Ім'я</string>

</property>

</item>

<item>

<property name="text">

<string>Прізвище</string>

</property>

</item>

<item>

<property name="text">

<string>Номер телефону</string>

</property>

</item>

<item>

<property name="text">

<string>День народження</string>

</property>

</item>

</widget>

</item>

</layout>

</item>

<item>

<widget class="QTableView" name="tableView"/>

</item>

<item>

<widget class="QPushButton" name="pushButton">

<property name="text">

<string>Вибрати читача</string>

</property>

</widget>

</item>

</layout>

</widget>

<widget class="QWidget" name="returnBook" native="true">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>711</width>

<height>401</height>

</rect>

</property>

<layout class="QVBoxLayout" name="verticalLayout\_4">

<item>

<widget class="QTableView" name="tableView\_3"/>

</item>

<item>

<widget class="QPushButton" name="pushButton\_6">

<property name="text">

<string>Повернути книгу</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="back2">

<property name="text">

<string>Назад</string>

</property>

</widget>

</item>

</layout>

<zorder>pushButton\_6</zorder>

<zorder>back2</zorder>

<zorder>tableView\_3</zorder>

</widget>

<widget class="QWidget" name="choose" native="true">

<property name="geometry">

<rect>

<x>10</x>

<y>110</y>

<width>691</width>

<height>131</height>

</rect>

</property>

<layout class="QVBoxLayout" name="verticalLayout\_3">

<item>

<widget class="QPushButton" name="pushButton\_3">

<property name="text">

<string>Видати книгу</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="pushButton\_4">

<property name="text">

<string>Повернути книгу</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="pushButton\_5">

<property name="text">

<string>Завершити</string>

</property>

</widget>

</item>

</layout>

</widget>

<zorder>returnBook</zorder>

<zorder>readers</zorder>

<zorder>choose</zorder>

<zorder>books</zorder>

</widget>

<resources/>

<connections/>

</ui>