

Library Management System

Ebrahim Golriz

December 2023

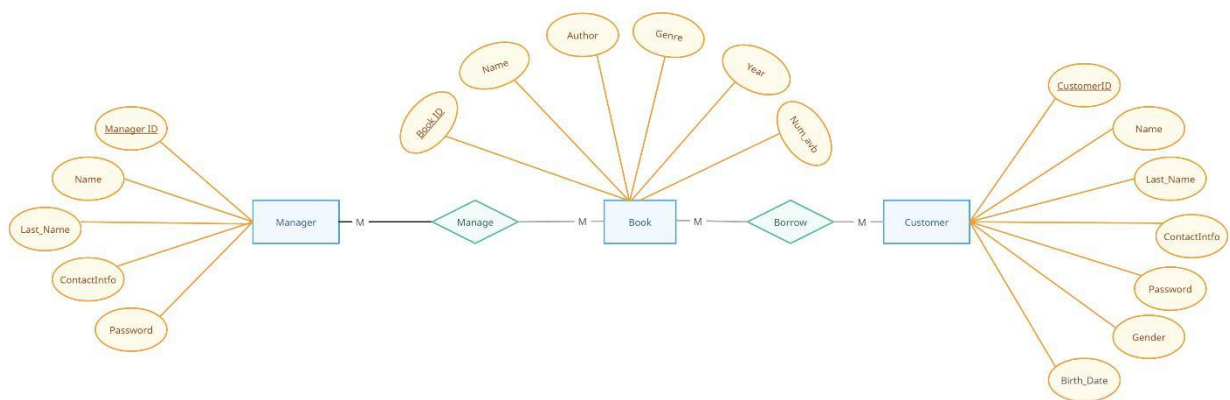
To implement a library system, we need some basic entities:

- **Book:** Information about each book and the number of each book in the library. There must be at least 1 copy of each book in the library.
- **Customer:** Contains personal information about the customer and the books he has borrowed. The customer can search through the library books, select the book he wants and borrow it. He also has access to the list of books he has borrowed and can return them to the library.
- **Manager:** Contains the personal information of the administrator; the administrator has access to the list of books and customers. He can see the books that a particular customer is currently borrowing.

In a hypothetical scenario,

The customer must register and log in with a password and Log in to the program, then he can view the available books and search based on the ID, Filter their various features. Then he can borrow the book he wants.

The manager can change book information or add a new book.



ER Chart Intended for this syste

Required tables created in *SQL Server*:

```
CreateTables.sql
1  CREATE TABLE books (
2      b_id INT IDENTITY(1,1) PRIMARY KEY,
3      b_name VARCHAR(100),
4      author VARCHAR(100),
5      genre VARCHAR(50),
6      p_year INT,
7      num_avb INT
8  );
9
10 CREATE TABLE managers (
11     m_id INT IDENTITY(1,1) PRIMARY KEY,
12     m_name VARCHAR(100),
13     m_lastname VARCHAR(100),
14     m_contactinfo VARCHAR(100),
15     m_password VARCHAR(50)
16 );
17
18
19 CREATE TABLE customers (
20     c_id INT IDENTITY(1,1) PRIMARY KEY,
21     c_name VARCHAR(100),
22     c_lastname VARCHAR(100),
23     c_contactinfo VARCHAR(100),
24     c_password VARCHAR(50),
25     c_gender VARCHAR(10),
26     c_birthdate DATE
27 );
28
29
30 CREATE TABLE customers_books (
31     c_id INT,
32     b_id INT,
33     Borrow_Date DATE,
34     FOREIGN KEY (c_id) REFERENCES customers(c_id),
35     FOREIGN KEY (b_id) REFERENCES books(b_id),
36     PRIMARY KEY (c_id, b_id)
37 );
38
39 CREATE TABLE customers_books_history (
40     c_id INT,
41     b_id INT,
42     date_borrowed DATE,
43     date_returned DATE default '9999-12-31',
44     PRIMARY KEY (c_id, b_id, date_borrowed),
45     FOREIGN KEY (c_id) REFERENCES customers(c_id),
46     FOREIGN KEY (b_id) REFERENCES books(b_id)
47 );
```

Using sqlalchemy ,We add the database and its tables to the program in the Python environment.

```
18 # Define the database connection string
19 db_connection_string = 'mssql+pyodbc://DESKTOP-3KLDFB9/LibraryDB?driver=ODBC+Driver+17+for+SQL+Server'
20
21 # Create the SQLAlchemy engine using PyODBC
22 db_engine = create_engine(db_connection_string)
23
24 # Create metadata
25 metadata = MetaData()
26
27 managers_table = Table('managers', metadata, autoload_with=db_engine)
28 customers_table = Table('customers', metadata, autoload_with=db_engine)
29 books_table = Table('books', metadata, autoload_with=db_engine)
30 customers_books_table = Table('customers_books', metadata, autoload_with=db_engine)
31 customers_books_history_table = Table('customers_books_history', metadata, autoload_with=db_engine)
32
```

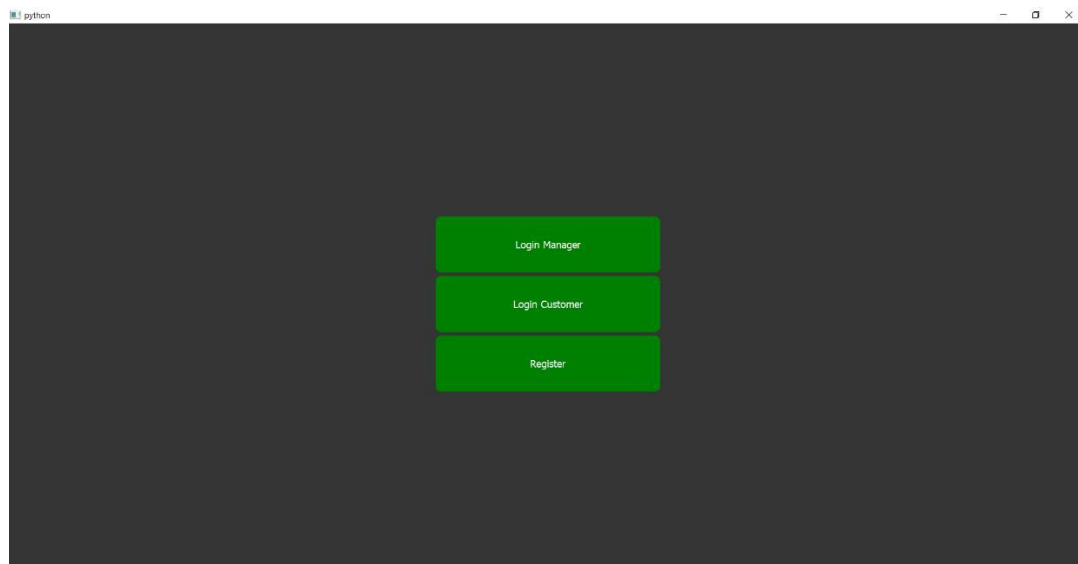
This section also includes the necessary preparations for creating a user interface using PyQt:

```
1876 def main():
1877     app = QApplication(sys.argv)
1878
1879
1880
1881     app.setStyle('Fusion')
1882
1883     palette = QPalette()
1884     palette.setColor(QPalette.Window, QColor(53, 53, 53))
1885     palette.setColor(QPalette.WindowText, Qt.white)
1886     palette.setColor(QPalette.Base, QColor(25, 25, 25))
1887     palette.setColor(QPalette.AlternateBase, QColor(53, 53, 53))
1888     palette.setColor(QPalette.ToolTipBase, Qt.white)
1889     palette.setColor(QPalette.ToolTipText, Qt.white)
1890     palette.setColor(QPalette.Text, Qt.white)
1891     palette.setColor(QPalette.Button, QColor(53, 53, 53))
1892     palette.setColor(QPalette.ButtonText, Qt.white)
1893     palette.setColor(QPalette.BrightText, Qt.red)
1894     palette.setColor(QPalette.Link, QColor(42, 130, 218))
1895     palette.setColor(QPalette.Highlight, QColor(116, 215, 112))
1896     palette.setColor(QPalette.HighlightedText, Qt.black)
1897
1898     app.setPalette(palette)
1899
1900     page_controller = PageController()
1901     page_controller.get_widget().show()
1902     sys.exit(app.exec_())
1903
1904 if __name__ == '__main__':
1905     main()
1906
```

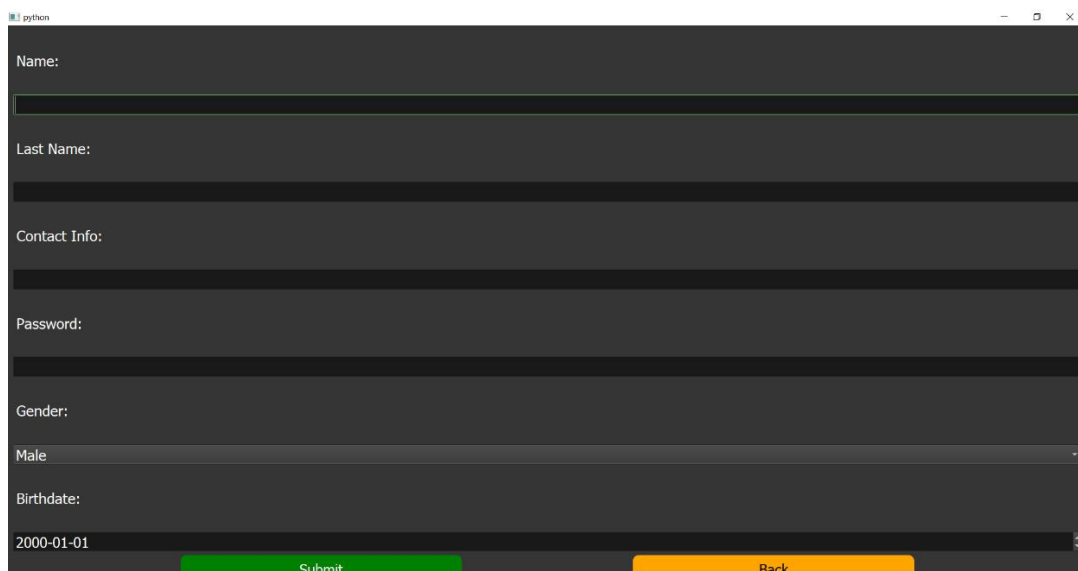
Now, by running a simple scenario, we will examine how the program works and how it interacts with the created database.

Customer

At the beginning of the program, we are presented with the following page:



By pressing the button Register, We enter the following page:

A screenshot of the same Python application window, now displaying a registration form. The form is on a dark gray background. It contains several input fields and labels: 'Name:' followed by a text input field; 'Last Name:' followed by a text input field; 'Contact Info:' followed by a text input field; 'Password:' followed by a text input field; 'Gender:' followed by a dropdown menu showing 'Male'; and 'Birthdate:' followed by a date input field showing '2000-01-01'. At the bottom of the form, there are two buttons: a green 'Submit' button on the left and an orange 'Back' button on the right.

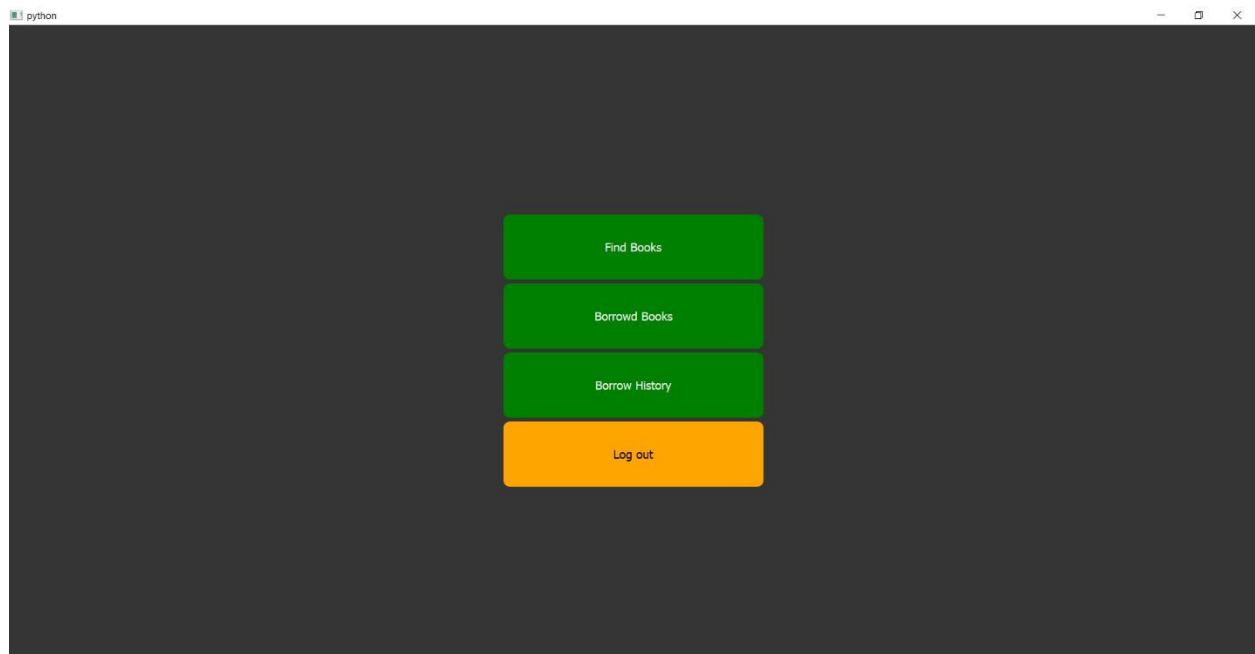
On this page, the user enters their information and presses the button **Submit**. The entered information, by creating a query and executing it, is converted into a new row or instance of the table **customers**.

```
1839 # Get the input values
1840 name = self.name_input.text()
1841 lastname = self.lastname_input.text()
1842 contactinfo = self.contactinfo_input.text()
1843 password = self.password_input.text()
1844 gender = self.gender_input.currentText()
1845 birthdate = self.birthdate_input.date()
1846
1847
1848 if not name or not lastname or not contactinfo or not password or not gender:
1849     QMessageBox.warning(self, 'Input Error', 'All fields must be filled.')
1850     return
1851 if birthdate.year() >= 2004:
1852     QMessageBox.warning(self, 'Input Error', 'Birthdate must be before 2004.')
1853     return
1854 birthdate = self.birthdate_input.date().toString("yyyy-MM-dd")
1855
1856 query = customers_table.insert().values(
1857     c_name=name,
1858     c_lastname=lastname,
1859     c_contactinfo=contactinfo,
1860     c_password=password,
1861     c_gender=gender,
1862     c_birthdate=birthdate
1863 )
1864
1865 with db_engine.connect() as connection:
1866     result = connection.execute(query)
1867     connection.commit()
```

After the information is confirmed, during a message, the user is given an ID that they can use to log in to the system.



The user will now be redirected to their home page:



By pressing the button **Find Books**, The following page will open where the user can apply the filters they need to the list of books in the table **books**.

A screenshot of a web application search filter form. The form is on a dark gray background. It contains several input fields and a dropdown menu. The fields are labeled: 'Book Name:', 'Genre:', 'Before Year:', 'After Year:', 'Author:', and 'Availability:'. The 'Genre' field contains the text 'fantasy'. The 'Before Year' field contains '2024'. The 'After Year' field contains '1'. The 'Availability' dropdown menu is open, showing the option 'Available to borrow'. At the bottom of the form, there are two buttons: a green 'Search' button and an orange 'Back' button. The window title bar at the top shows 'python' and standard window controls.

Available to borrow option Only shows books that the library have more than 1 of.

Search command with filters applied by a query is executed, the result is the rows fetched from the table and If the size value is zero, it means that there was no book with the applied filters in the books table:

```
def submit_search(self):
    # Get the input values
    b_name = self.b_name_input.text()
    genre = self.genre_input.text()
    before_year = self.before_year_input.text()
    if (not before_year):
        before_year = 2024
    after_year = self.after_year_input.text()
    if (not after_year):
        after_year = 1
    author = self.author_input.text()
    availability = self.availability_input.currentText()
    query = select(books_table).where([
        (books_table.c.b_name.like(f'%{b_name}%')) &
        (books_table.c.genre.like(f'%{genre}%')) &
        (books_table.c.p_year <= int(before_year)) &
        (books_table.c.p_year >= int(after_year)) &
        (books_table.c.author.like(f'%{author}%')) &
        (
            (books_table.c.num_avb > 1)
            if availability == 'Available to borrow'
            else True
        )
    ])
    with db_engine.connect() as connection:
        result = connection.execute(query)
        rows = result.fetchall()
        connection.commit()

    # Check if any books were found
    if len(rows) == 0:
        QMessageBox.warning(self, 'No Books Found', 'No books were found that match your search criteria.')
    else:
```

	ID	Name	Author	Genre	Year	Availability	Action
1	8	The Lost Kingdom	Victoria Kingsley	Fantasy	2019	2	<button>Borrow</button>
2	18	The Forgotten Realm	Serena Nightingale	Fantasy	2020	5	<button>Borrow</button>
3	24	Lord of the Rings	J.R.R. Tolkien	Fantasy	1954	7	<button>Borrow</button>
4	25	The Hobbit	J.R.R. Tolkien	Fantasy	1937	5	<button>Borrow</button>
5	26	The Hidden Oasis	Isabella Mirage	Fantasy	2020	2	<button>Borrow</button>
6	32	Kingdom of Dreams	William Kingsley	Fantasy	2017	7	<button>Borrow</button>

List of books shown to the user

Now by pressing the button **Borrow** For each book, it is first checked whether the user has permission to borrow that book; The conditions are:

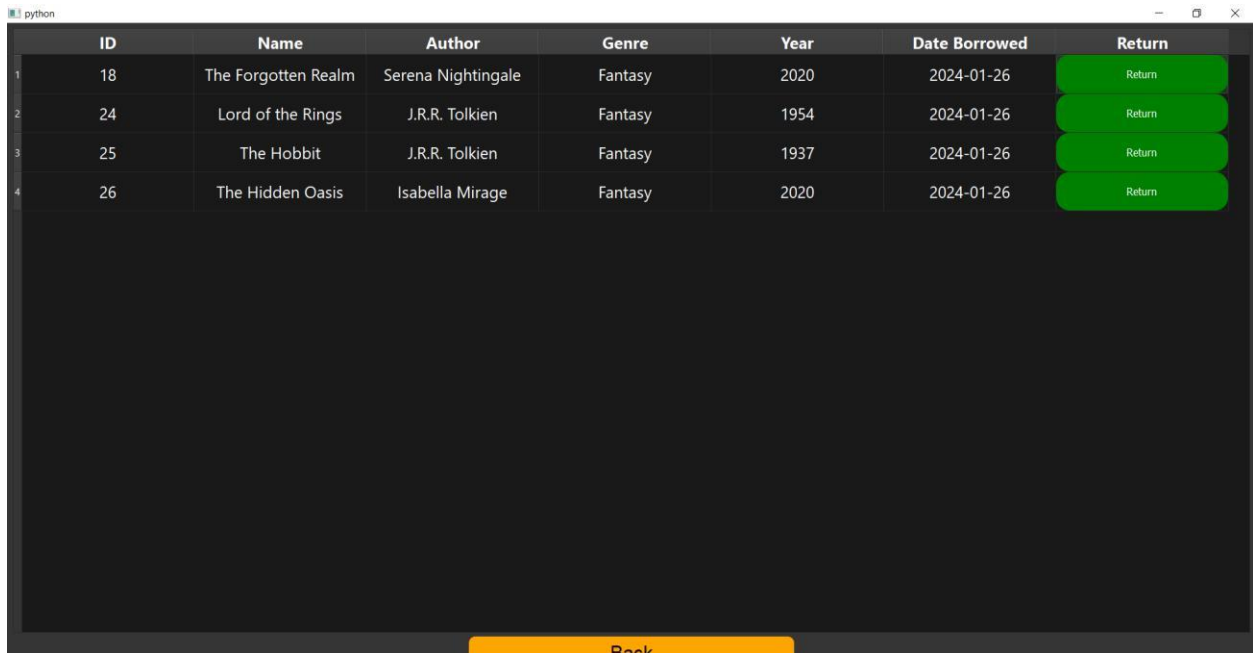
1. The stock level of the desired book must be at least 2.
2. The user has not currently borrowed that book.
3. The user cannot borrow and return a book on the same day and then want to borrow that book again on the same day.

```
1445
1446 with db_engine.connect() as connection:
1447
1448     # Check the availability of the book
1449     query = select(books_table.c.num_avb).where(books_table.c.b_id == book_id)
1450
1451     num_avb_item = self.table.item(row, 5)
1452
1453     if num_avb <= 1:
1454         QMessageBox.warning(self, 'Book Unavailable', f'This is the last available stock of book {book_id}, "{book_name}". Can not be borrowed.')
1455         return
1456     query = select(customers_books_table.c).where(
1457         (customers_books_table.c.c_id == self.customer_id) &
1458         (customers_books_table.c.b_id == book_id)
1459     )
1460     result = connection.execute(query)
1461     row = result.fetchone()
1462     if row is not None:
1463         QMessageBox.warning(self, 'Book Already Borrowed', f'You have already borrowed one copy of book {book_id}, "{book_name}".')
1464         return
1465
1466     # Check if the book has already been borrowed by the customer on the same day
1467     query = select(customers_books_history_table.c).where(
1468         (customers_books_history_table.c.c_id == self.customer_id) &
1469         (customers_books_history_table.c.b_id == book_id) &
1470         (customers_books_history_table.c.date_borrowed == date.today())
1471     )
1472     result = connection.execute(query)
1473     row = result.fetchone()
1474     if row is not None:
1475         QMessageBox.warning(self, 'Book Already Borrowed Today and then returned', f'You have already borrowed book {book_id}, "{book_name}" today and returned it.')
1476         return
1477
```

Next, if the above conditions are met, first Insert book and customer ID in customers_books table along with the date of the day. Then the same values are also entered in customers_books_history table. After that, the inventory level of that book decreases by one unit:

```
1481
1482 # Insert the customer_id and book_id into the customers_books table
1483 query = customers_books_table.insert().values(
1484     c_id=self.customer_id,
1485     b_id=book_id,
1486     Borrow_Date=date.today()
1487 )
1488 connection.execute(query)
1489
1490 # Insert the customer_id and book_id into the customers_books_history table
1491 query = customers_books_history_table.insert().values(
1492     c_id=self.customer_id,
1493     b_id=book_id,
1494     date_borrowed = date.today()
1495 )
1496 connection.execute(query)
1497 # update the book's availability
1498 query = (
1499     update(books_table).
1500     where(books_table.c.b_id == book_id).
1501     values(num_avb=books_table.c.num_avb - 1)
1502 )
1503 connection.execute(query)
1504 connection.commit()
1505
1506 self.find_books_page.submit_search()
1507 QMessageBox.information(self, 'Book Borrowed', f'Customer {self.customer_id} borrowed book {book_id}, "{book_name}".')
```


By pressing the button **Borrowed Books** On the user's home page, we are presented with the following page, which is a list of books that the customer has purchased. Currently borrowed, along with book details and borrowing date:



ID	Name	Author	Genre	Year	Date Borrowed	Return
18	The Forgotten Realm	Serena Nightingale	Fantasy	2020	2024-01-26	Return
24	Lord of the Rings	J.R.R. Tolkien	Fantasy	1954	2024-01-26	Return
25	The Hobbit	J.R.R. Tolkien	Fantasy	1937	2024-01-26	Return
26	The Hidden Oasis	Isabella Mirage	Fantasy	2020	2024-01-26	Return

This list is created by creating a select query in which the tables `customers_books_table` and `books_table` with the same ID are joined together, the rows where the user ID is the same as the ID in `customers_books` are selected, and the desired columns are picked:

```
908 def find borrowed books(self):
909     # Join the tables and select the columns
910     query = [
911         select(
912             books_table.c.b_id,
913             books_table.c.b_name,
914             books_table.c.author,
915             books_table.c.genre,
916             books_table.c.p_year,
917             customers_books_table.c.Borrow_Date
918         )
919         .join_from(customers_books_table, books_table, customers_books_table.c.b_id == books_table.c.b_id)
920         .where(customers_books_table.c.c_id == self.customer_id)
921     ]
922
923     with db_engine.connect() as connection:
924         result = connection.execute(query)
925         rows = result.fetchall()
926
927     if len(rows) == 0:
928         QMessageBox.warning(self, 'No Books Found', 'You do not currently have a borrowed book')
929         self.page_controller.show_customer_main_page()
```

If there are no rows as a result of this fetch, it means the user has not currently borrowed a book.

By pressing the **Return** button in the list of books borrowed by the user, the **customers_books_history** table is first updated, and for the desired row (given the customer ID and book and the return date, which is equivalent to the default date), the value of the **date_returned** column is changed to the current date; then the inventory of that book is increased by one unit, and that desired row is deleted from the **customers_books** table, given the customer ID and book.

```
with db_engine.connect() as connection:

    # update customers_books_history table
    query = update(customers_books_history_table).where(
        customers_books_history_table.c.b_id == book_id,
        customers_books_history_table.c.c_id == self.customer_id,
        customers_books_history_table.c.date_returned == '9999-12-31'
    ).values(
        date_returned = date.today()
    )
    connection.execute(query)

    # update the book's availability
    query = update(books_table).where(books_table.c.b_id == book_id).values(num_avb = books_table.c.num_avb + 1)
    connection.execute(query)

    # remove item from customers_books table
    query = delete(customers_books_table).where(
        customers_books_table.c.b_id == book_id,
        customers_books_table.c.c_id == self.customer_id
    )

    connection.execute(query)

    connection.commit()
```

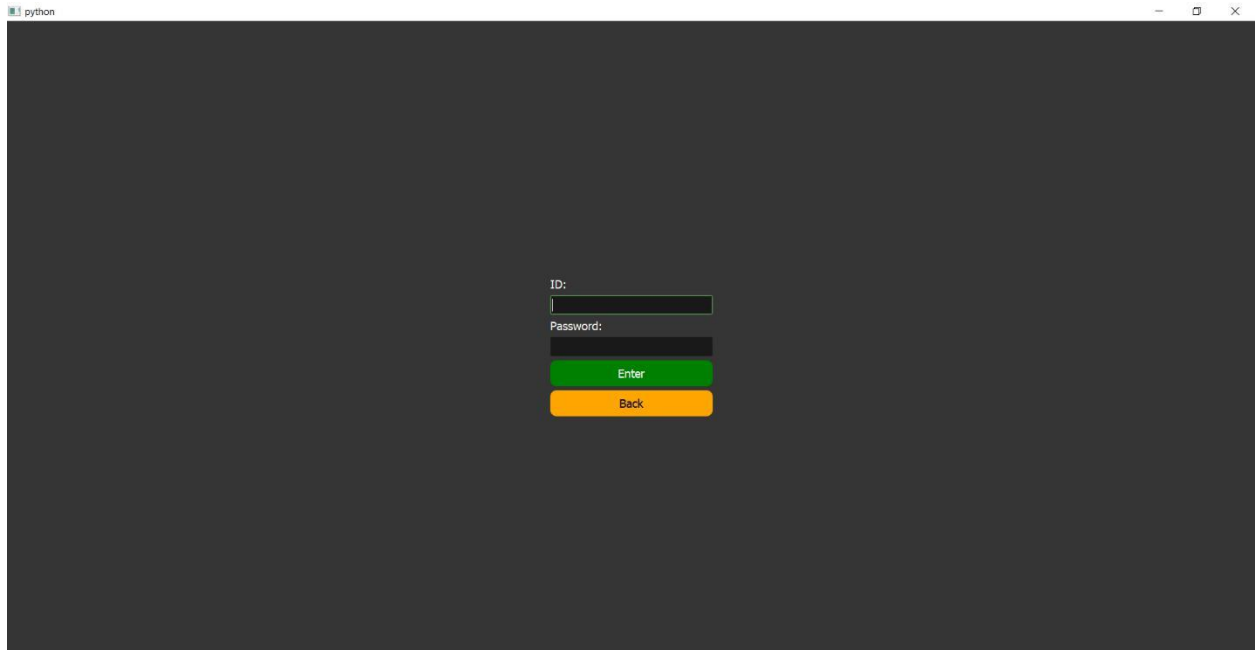
By pressing the **Borrow History** button on the user's home page, the rows related to the user are fetched from the **customers_books_history** table according to the select statement explained in the previous section and displayed to the user:

	Book ID	Book Name	Author	Genre	Year	Borrow Date	Return Date
1	24	Lord of the Rings	J.R.R. Tolkien	Fantasy	1954	2024-01-26	Not yet returned
2	26	The Hidden Oasis	Isabella Mirage	Fantasy	2020	2024-01-26	Not yet returned
3	18	The Forgotten Realm	Serena Nightingale	Fantasy	2020	2024-01-26	2024-01-26
4	25	The Hobbit	J.R.R. Tolkien	Fantasy	1937	2024-01-26	2024-01-26

Back

Manager

The manager must first log in to their account (an instance of the administrator in the administrators table is already entered into the database)



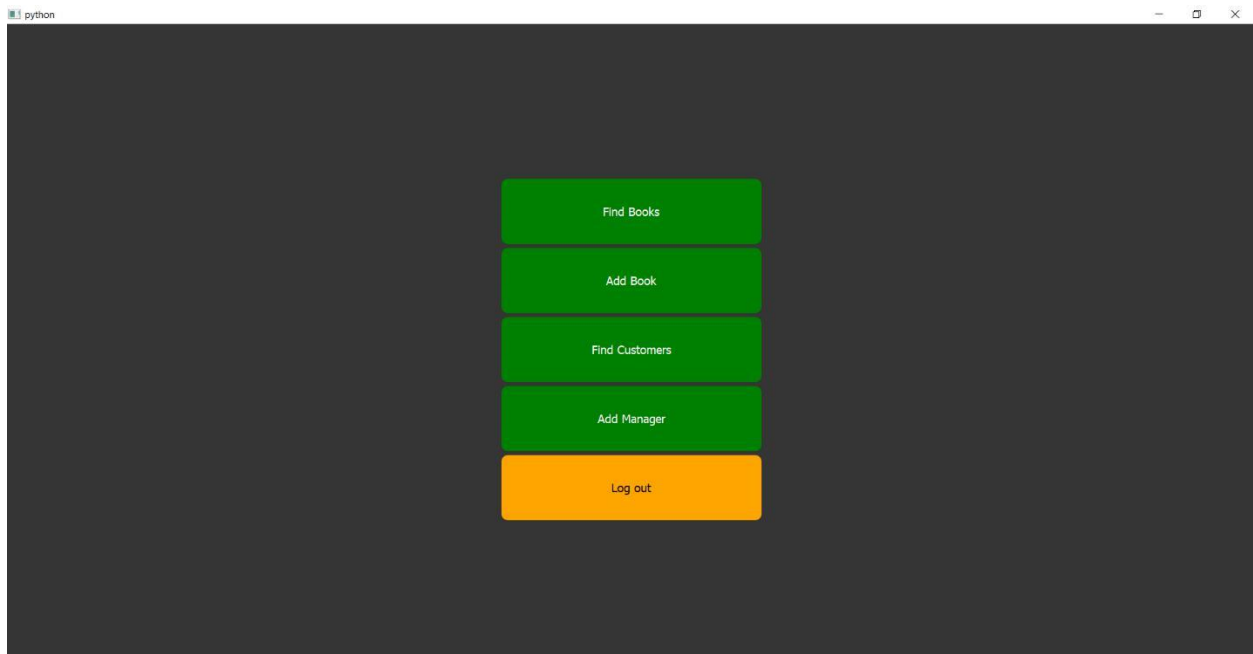
By entering the information, the ID and Password entered in the managers table are selected and if such a row exists, the manager is taken to the main management page:

```
with db_engine.connect() as connection:
    query = select(managers_table).where(
        (managers_table.c.m_id == entered_id) & (managers_table.c.m_password == entered_password)
    )
    result = connection.execute(query).fetchall()

    if len(result) > 0:
        QMessageBox.information(self, 'Login Status', 'Welcome!')
        self.page_controller.managermainpage.manager_id = entered_id
        self.page_controller.show_manager_main_page()
    else:
        QMessageBox.warning(self, 'Login Status', 'User not found.')
        self.line_edit_id.clear()
        self.line_edit_password.clear()
```

User login is done in the same way.

The admin home page looks like this:



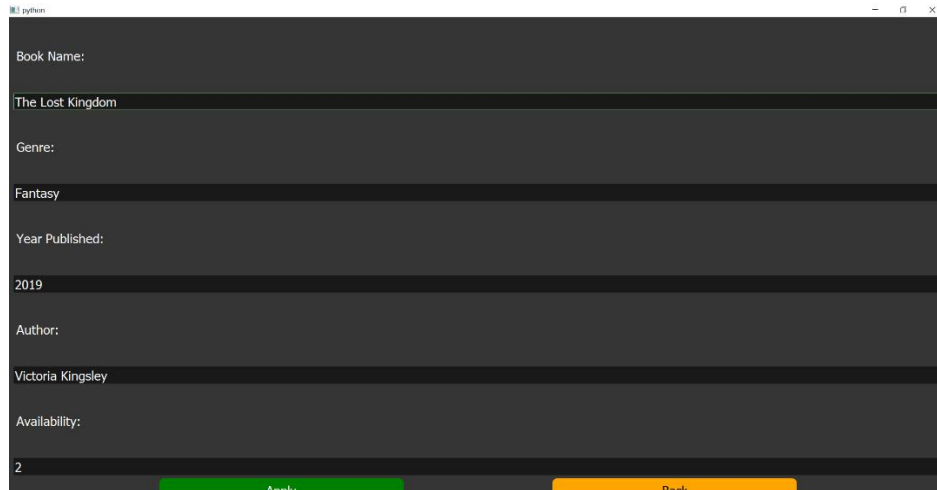
The manager's home page looks like this: By pressing the **Find Books** button, we are redirected to the same page that was shown to the user to filter the books, and after applying the necessary filters, the list of books is shown to the administrator:

A screenshot of a web application window titled 'python'. It displays a table with 8 columns: ID, Name, Author, Genre, Year, Availability, Edit, and Current Borrowers. The table contains 6 rows of data. Below the table, there is a large dark gray area and a yellow 'Back' button at the bottom center.

	ID	Name	Author	Genre	Year	Availability	Edit	Current Borrowers
1	8	The Lost Kingdom	Victoria Kingsley	Fantasy	2019	2	Edit	Current Borrowers
2	18	The Forgotten ...	Serena Nightingale	Fantasy	2020	5	Edit	Current Borrowers
3	24	Lord of the Rings	J.R.R. Tolkien	Fantasy	1954	6	Edit	Current Borrowers
4	25	The Hobbit	J.R.R. Tolkien	Fantasy	1937	5	Edit	Current Borrowers
5	26	The Hidden Oasis	Isabella Mirage	Fantasy	2020	1	Edit	Current Borrowers
6	32	Kingdom of ...	William Kingsley	Fantasy	2017	7	Edit	Current Borrowers

As you can see, for each book, there are two buttons for the manager, **Edit** and List of people who have currently **borrowed** that book.

By pressing the button **Edit** The following page is shown to the administrator where he can change the book information:



The screenshot shows a Python application window with a dark theme. It contains a form for editing book information. The fields are as follows:

- Book Name: The Lost Kingdom
- Genre: Fantasy
- Year Published: 2019
- Author: Victoria Kingsley
- Availability: 2

At the bottom of the form, there are two buttons: a green "Apply" button and an orange "Back" button.

By confirming the entered information, the values of the properties of that book are replaced with the entered values, and the table **books** will be updated.

```
def applychanges(self):
    b_name = self.b_name_input.text()
    genre = self.genre_input.text()
    author = self.author_input.text()

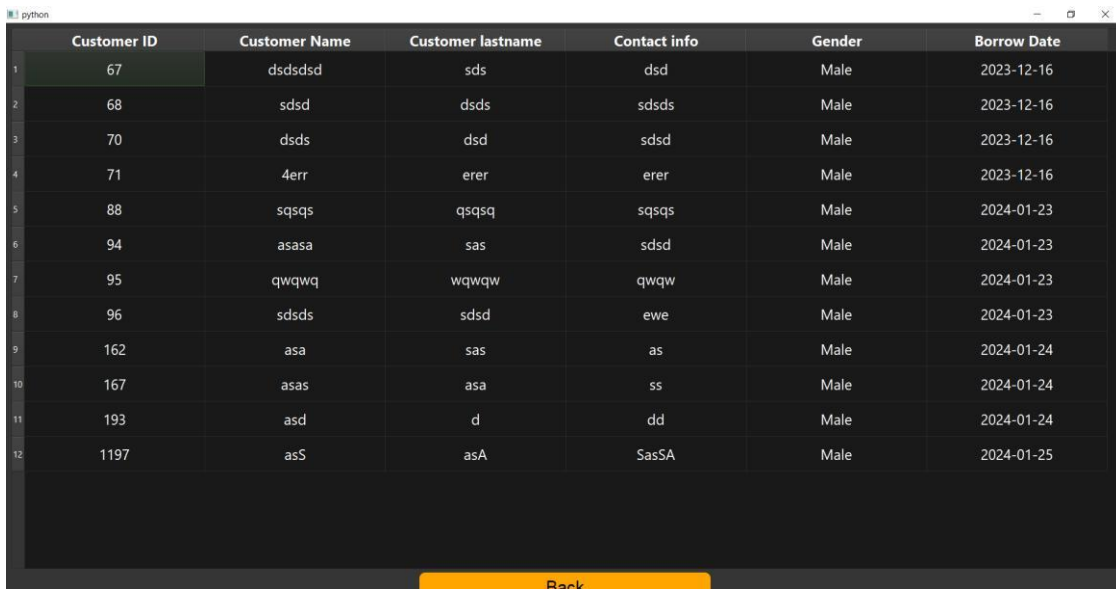
    try:
        p_year = int(self.year_published_input.text())
        num_avb = int(self.availability_input.text())
    except ValueError:
        QMessageBox.warning(self, 'Input Error', 'Year and Availability must be numbers.')
        return

    if not b_name or not genre or not p_year or not author or not num_avb:
        QMessageBox.warning(self, 'Input Error', 'All fields must be filled.')
        return

    update_stmt = update(books_table).where(books_table.c.b_id == self.book_id).values(
        b_name=b_name,
        genre=genre,
        p_year=p_year,
        author=author,
        num_avb=num_avb
    )

    with db_engine.connect() as connection:
        result = connection.execute(update_stmt)
        connection.commit()
```

By pressing the button **Current Borrowers**, a list of customers who have currently borrowed that book will be displayed:



	Customer ID	Customer Name	Customer lastname	Contact info	Gender	Borrow Date
1	67	dsdsdsd	sds	dsd	Male	2023-12-16
2	68	sdsd	dsds	sdsds	Male	2023-12-16
3	70	dsds	dsd	sdsd	Male	2023-12-16
4	71	4err	erer	erer	Male	2023-12-16
5	88	sqsqs	qsqsq	sqsqs	Male	2024-01-23
6	94	asasa	sas	sdsd	Male	2024-01-23
7	95	qwqwq	wqwqw	qwqw	Male	2024-01-23
8	96	sdsds	sdsd	ewe	Male	2024-01-23
9	162	asa	sas	as	Male	2024-01-24
10	167	asas	asa	ss	Male	2024-01-24
11	193	asd	d	dd	Male	2024-01-24
12	1197	asS	asA	SasSA	Male	2024-01-25

Back

This operation is performed by joining the two tables **customers_books** and **customers**, and selecting those rows where the ID of the desired book is equal to the same ID in **customers_books**.

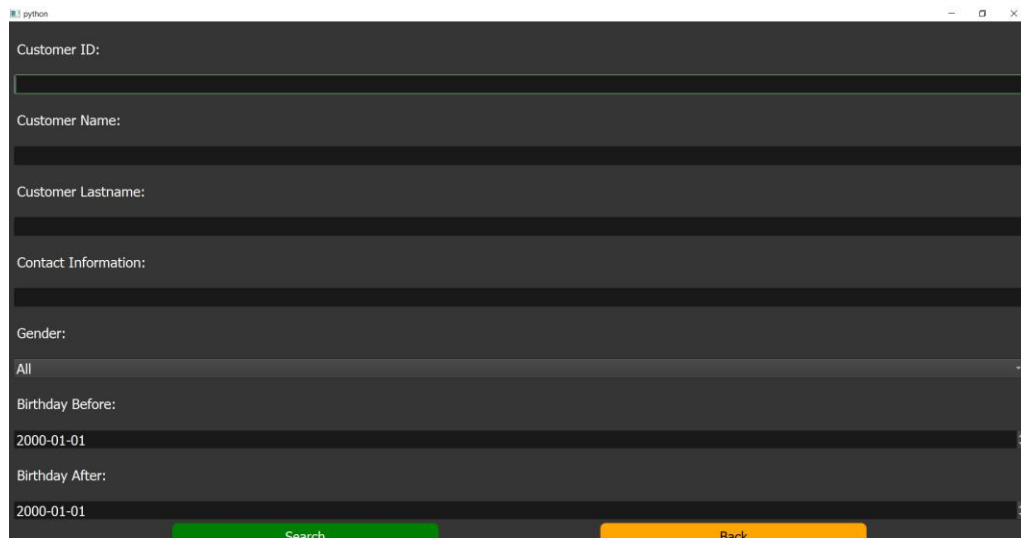
```
def load_data(self):
    self.table.setSortingEnabled(False)
    self.table.clear()

    query = (
        select(
            customers_table.c.c_id,
            customers_table.c.c_name,
            customers_table.c.c_lastname,
            customers_table.c.c_contactinfo,
            customers_table.c.c_gender,
            customers_books_table.c.Borrow_Date,
        )
        .join_from(customers_books_table, customers_table, customers_books_table.c.c_id == customers_table.c.c_id)
        .where(customers_books_table.c.b_id == self.book_id)
    )

    with db_engine.connect() as connection:
        result = connection.execute(query)
        rows = result.fetchall()

    if len(rows) == 0:
        QMessageBox.warning(self, 'No current borrowers', 'This books has no current borrowers')
        self.page_controller.show_booklist_page()
```

By pressing the button **Find Customers**, The following page is displayed, giving the administrator the ability to filter customers based on their characteristics:



The screenshot shows a search form with the following fields and controls:

- Customer ID:
- Customer Name:
- Customer Lastname:
- Contact Information:
- Gender:
- Birthday Before:
- Birthday After:
- Search:
- Back:

This is done by executing the select command with the condition entered by the administrator:

```
def submit_search(self):
    # Get the input values
    c_id_text = self.c_id_input.text()
    c_name = self.c_name_input.text()
    c_lastname = self.c_lastname_input.text()
    c_contactinfo = self.c_contactinfo_input.text()
    c_gender = self.c_gender_input.currentText()
    c_birthdate_before = self.birthdate_before_input.date().toString("yyyy-MM-dd")
    c_birthdate_after = self.birthdate_after_input.date().toString("yyyy-MM-dd")

    query = select(customers_table).where(
        (customers_table.c.c_name.like(f'"{c_name}"')) &
        (customers_table.c.c_lastname.like(f'"{c_lastname}"')) &
        (customers_table.c.c_contactinfo.like(f'"{c_contactinfo}"')) &
        (customers_table.c.c_birthdate.between(c_birthdate_after, c_birthdate_before))
    )

    if c_id_text:
        try:
            c_id = int(c_id_text)
        except ValueError:
            QMessageBox.warning(self, 'Input Error', 'ID must be a number.')
            return
        query = query.where(customers_table.c.c_id == c_id)

    if c_gender != "All":
        query = query.where(customers_table.c.c_gender == c_gender)

    with db_engine.connect() as connection:
        result = connection.execute(query)
        rows = result.fetchall()
        connection.commit()

    # Check if any customers were found
    if len(rows) == 0:
        QMessageBox.warning(self, 'No Customers Found', 'No customers were found that match your search criteria.')
```


	ID	Name	Lastname	Contact info	Password	gender	Birthdate	Action
1	24	asasasa	sasa	sasas	asasasas	Male	2000-01-01	Books
2	25	ssdf	sdfsdf	sdfs	dfsdfsdf	Male	2000-01-01	Books
3	26	utyjtjytj	tyjty	jtyjyt	jtyjty	Male	2000-01-01	Books
4	27	asdasd	asdsadsa	dasd	asdsad	Male	2000-01-01	Books
5	28	sdasd	asdas	dasdasd	sadsad	Male	2000-01-01	Books
6	29	dvsdff	dsfd	fsdfs	dfsdfsdf	Male	2000-01-01	Books
7	30	asdsad	sdasdas	dasdas	dasdasdas	Male	2000-01-01	Books
8	31	sdasd	asdasd	asdasd	asdasd	Male	2000-01-01	Books
9	32	sdadasd	dasdasd	sdas	asda	Male	2000-01-01	Books
10	33	sdadasd	dasdasd	sdas	asda	Male	2000-01-01	Books
11	34	sdadasd	dasdasd	sdas	asda	Male	2000-01-01	Books
12	35	f	f	f	f	Male	2000-01-01	Books
13	36	sas	as	asas	as	Male	2000-01-01	Books
14	37	yhj	jgh	hjgh	hjg	Male	2000-01-01	Books
15	38	asAS	SASAS	ASA	AS	Male	2000-01-01	Books

Back

In the list shown to the administrator, for each customer there is a **Books** button that shows the books that the customer has borrowed so far.

```
def load_data(self):
    self.table.setSortingEnabled(False)
    self.table.clear()

    query = (
        select(
            books_table.c.b_id,
            books_table.c.b_name,
            books_table.c.author,
            books_table.c.genre,
            books_table.c.p_year,
            customers_books_history_table.c.date_borrowed,
            customers_books_history_table.c.date_returned
        )
        .join_from(customers_books_history_table, books_table, customers_books_history_table.c.b_id == books_table.c.b_id)
        .where(customers_books_history_table.c.c_id == self.customer_id)
    )

    with db_engine.connect() as connection:
        result = connection.execute(query)
        rows = result.fetchall()

    if len(rows) == 0:
        QMessageBox.warning(self, 'No Books Found', 'No books have been borrowed yet')
```



```

def addbook(self):
    # Get the input values
    b_name = self.b_name_input.text()
    genre = self.genre_input.text()
    author = self.author_input.text()
    try:
        year = int(self.year_input.text())
        availability = int(self.availability_input.text())
    except ValueError:
        QMessageBox.warning(self, 'Input Error', 'Year and Availability must be numbers.')
        return

    if not b_name or not genre or not year or not author or not availability :
        QMessageBox.warning(self, 'Input Error', 'All fields must be filled.')
        return

    query = books_table.insert().values(
        b_name = b_name,
        author = author,
        genre = genre,
        p_year = year,
        num_avb = availability
    )

    with db_engine.connect() as connection:
        result = connection.execute(query)
        connection.commit()

```

By pressing the button **Add Manager**, the following page will be displayed and by entering the details and confirming it, a new administrator instance will be created.

The image shows a web application window titled "python". It contains a form with the following elements:

- Name:** A text input field.
- Last Name:** A text input field.
- Contact Info:** A text input field.
- Password:** A text input field.
- Buttons:** At the bottom, there are two buttons: a green "Submit" button and an orange "Back" button.

```
def submit_registration(self):  
    # Get the input values  
    name = self.name_input.text()  
    lastname = self.lastname_input.text()  
    contactinfo = self.contactinfo_input.text()  
    password = self.password_input.text()  
  
    if not name or not lastname or not contactinfo or not password:  
        QMessageBox.warning(self, 'Input Error', 'All fields must be filled.')  
        return  
  
    query = managers_table.insert().values(  
        m_name=name,  
        m_lastname=lastname,  
        m_contactinfo=contactinfo,  
        m_password=password,  
    )  
  
    with db_engine.connect() as connection:  
        result = connection.execute(query)  
        connection.commit()
```