Alexandria University
Faculty of Engineering
Computer and System Engineering
Department

CS 322: Database Systems

Book Store Project

Students' Name:
- Arsany Atef Abdo (10)
- Kirelos Malak Habib (35)
- Michael Said Beshara (38)

Professor's Name:
DR. Yousry Taha

TA's Name:
Eng. Reham Osama

Links of SQL Scripts for the Project

- **Creation of schema and tables:**

  https://drive.google.com/file/d/1zLamEgX6NuQAIKfeeksTPk4iwL038-xx/view?usp=sharing

- **Creation Triggers:**

  https://drive.google.com/file/d/1KyWXpBBcbY1WiUg53u-yj-ZL7DX2FLGQ/view?usp=sharing

- **ERD Diagram:**

  https://drive.google.com/file/d/1aD5W_v6yhnA2Yrn2JSrubOvKlR__Q3_O/view?usp=sharing

---

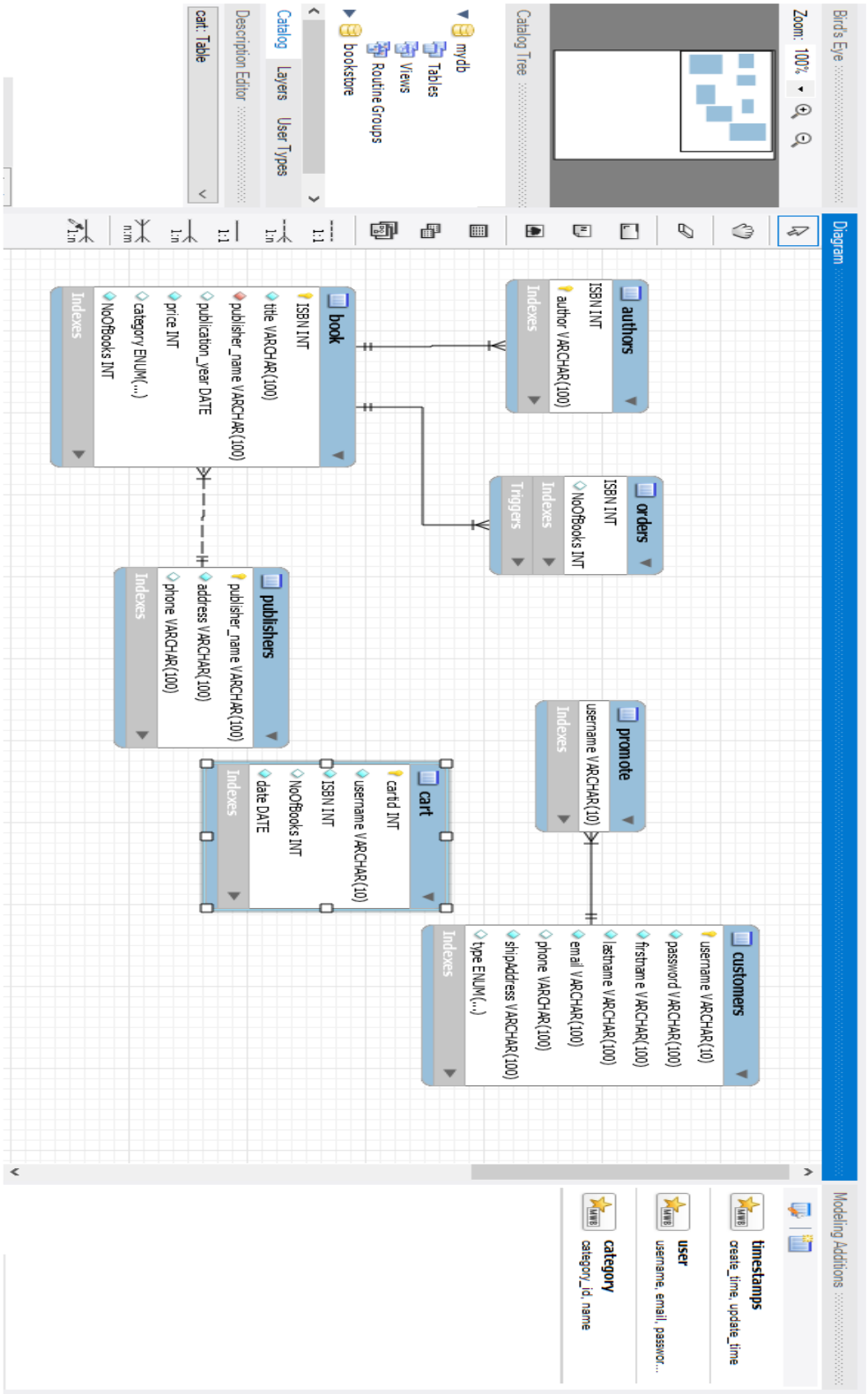
Links of Jasper files

I open them with atom editor

- **Link of the folder contains the three files:**

  https://drive.google.com/drive/folders/1KbhlU_T6WdUKzCTMPHZEf4sxIVKvDjrQ?usp=sharing

- **Database ERD diagrams:**



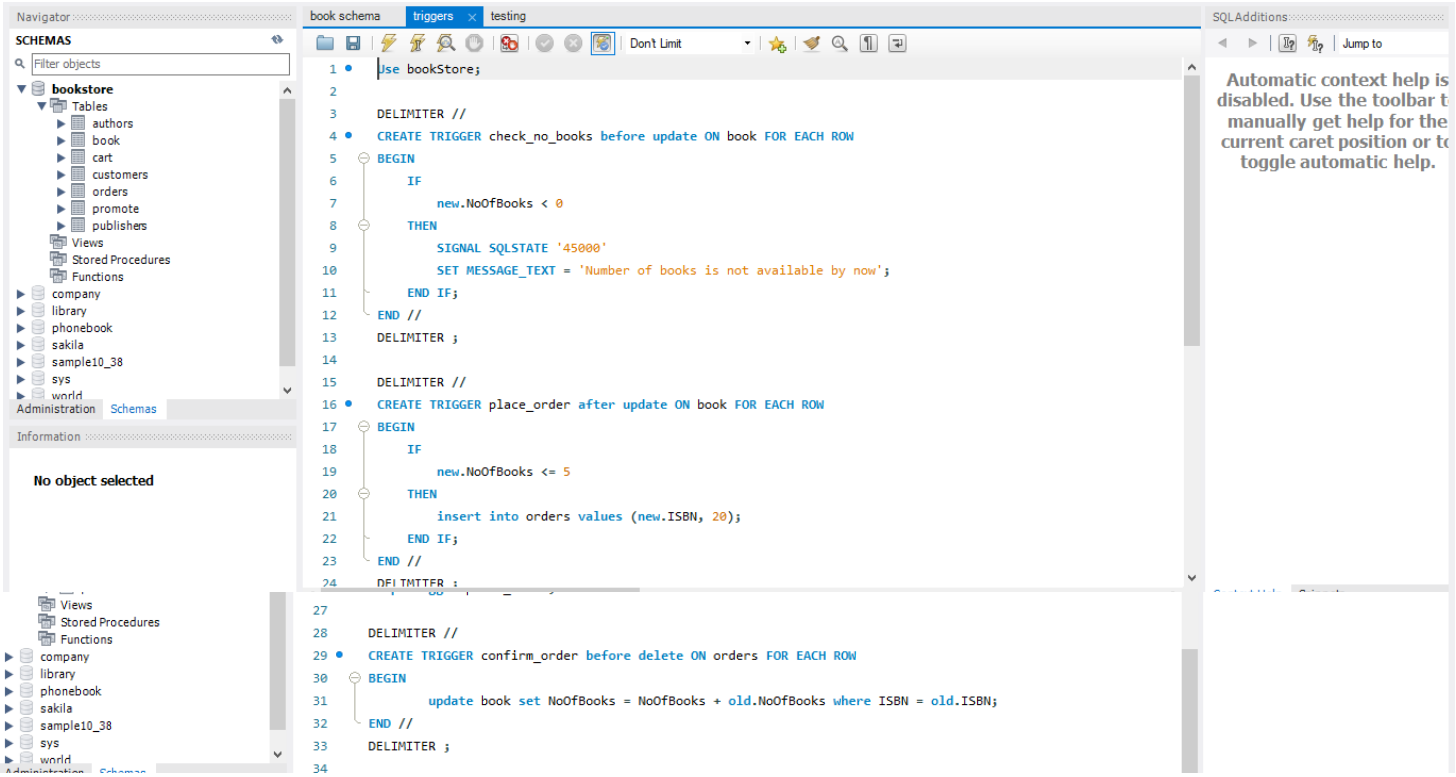
- **Description, Analysis:**
  - **Schema Assumptions:**
    - There are 7 tables (publishers, book, authors, orders, customers, cart and promote).
    - Orders table has the orders have been done buy the managers, has two attributes (ISBN and number of books).
    - Cart has the sells of the book store, has four attributes (cart id (auto generated and incremented), username, ISBN, and number of books).
    - Promote has the user who want to be a manager and has attribute which is the users' names.
    - Book table will have ISBN auto generated and increment one by one with each insertion in it.
    - The Authors have another table extracted from the book table to accomplish the BCNF to be atomic as authors are multivalued.

  - **Triggers used and its assumptions:**
    - Check number of books to not to be negative.
    - Order books of about 20 new books if number of books reach 5.
    - Confirm the order by the manager.

○ Schema screenshots:

```
Create schema bookStore;
use bookStore;

# ---------------------------------------------
# ------------------PART 1--------------------
# ---------------------------------------------
Create Table publishers (
publisher_name Varchar (100) primary key Not Null,
address Varchar(100) Not Null,
phone Varchar(100));

#----  may be change ISBN to INT -------

Create Table book (
ISBN int AUTO_INCREMENT Primary Key Not Null,
title Varchar(100) Not Null,
publisher_name Varchar (100) Not Null,
publication_year Date,
price INT default 0  Not Null,
category ENUM ('Science', 'Art', 'Religion', 'History', 'Geography'),
NoOfBooks INT default 20 Not Null,
constraint fk1 foreign key (publisher_name) references publishers (publisher_name) on delete cascade on upda

Create Table authors (
ISBN int Not Null,
author Varchar(100) Not Null,
constraint pk1 primary key (ISBN, author),
constraint fk2 foreign key (ISBN) references book (ISBN) on delete cascade on update cascade);

Create Table orders (
ISBN int Primary Key Not Null,
NoOfBooks INT default 20,
constraint fk3 foreign key (ISBN) references book (ISBN) on delete cascade on update cascade);

# ---------------------------------------------
Create Table customers (
username Varchar (10) Primary Key Not Null,
password Varchar(100) Not Null,
firstname Varchar (100) Not Null,
lastname Varchar (100) Not Null,
email Varchar (100)  Not Null,
phone Varchar(100),
shipAddress Varchar(100) Not Null,
type ENUM ('user', 'manager'));

Create Table cart (
cartid int AUTO_INCREMENT PRIMARY KEY,
username Varchar (10) Not Null,
ISBN int Not Null ,
NoOfBooks INT,
date Date Not Null);

Create Table promote (
username Varchar (10) Primary Key Not Null,
constraint fk6 foreign key (username) references customers (username) on delete cascade on update cascade);
```

○ Triggers screenshots:



○ Java description and assumptions:

- **Components:**
  - *Book*

    Contains the book attributes identified in the SQL.
  - *Cart*

    Contains the cart attributes identified in the SQL.
  - *Person*

    Abstract class contain the name, address and phone of person.
  - *User*

    Class extends from the person in addition to the rest of attributes of the customer in the SQL file.
  - *Publisher*

    Class extends from the person.

- **Back end:**

  - *Book Store*

    - Singleton class identified in it the main functions of the online bookstore.

    - ***signUp function:***

      - Takes the new user information and return true if operation is done successfully and false otherwise.

```java
@Override
public boolean signUp(IUser user) {
    // TODO Auto-generated method stub
    HashMap<String, Pair<String, String>> conditions = new HashMap<>();
    conditions.put("username", new Pair<String, String>("=", user.getName()));
    ResultSet result = this.mySqlConnection.search_item(USERS_TABLE, null, conditions);

    try {
        if(!result.next()) {

            if(this.mySqlConnection.insert_item(USERS_TABLE, user.getAttributes())) {
                this.user = user;
                return true;
            }
        }
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        System.out.println(e.toString());
    }
    return false;
}
```

    - ***logIn function:***

      - Takes the username and password of the user and return the all information of this username if exists in the system.

```java
@Override
public IUser logIn(String username, String password) {
    // TODO Auto-generated method stub
    this.user = null;
    HashMap<String, Pair<String, String>> conditions = new HashMap<>();
    conditions.put("username", new Pair<String, String>("=", username));
    ResultSet result = this.mySqlConnection.search_item(USERS_TABLE, null, conditions);

    try {
        if(result.next() && result.getString("password").equals(password)) {
            this.user = new User(username, result.getString("shipAddress"), result.getString("phone"));
            this.user.setEmail(result.getString("email"));
            this.user.setFirst_name(result.getString("firstname"));
            this.user.setLast_name(result.getString("lastname"));
            this.user.setPassword(result.getString("password"));
            this.user.setIsManager("manager".equals(result.getString("type")));
            return this.user;
        }
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        System.out.println(e.toString());
    }
    return this.user;
}
```

- ***updateSettings function:***

- Takes the new user information and return true if operation is done successfully and false otherwise.

```
@Override
public boolean updateSettings(IUser user) {
    // TODO Auto-generated method stub
    HashMap<String, Pair<String, String>> conditions = new HashMap<>();
    conditions.put("username", new Pair<String, String>("=", this.user.getName()));
    //--------------------------
    if(mySqlConnection.update_item(USERS_TABLE, user.getAttributes(), conditions)) {
        this.user = user;
        return true;
    }
    return false;
}
```

- ***demandUser function:***

- This function adds the current user to promote table to have managers credentials.

```
@Override
public void demandUser() {
    // TODO Auto-generated method stub
    if(user.getIsManager()) {
        return;
    }
    HashMap<String, String> attributes = new HashMap<>();
    attributes.put("username", user.getName());
    this.mySqlConnection.insert_item(PROMOTE_TABLE, attributes);
}
```

-

- ***addBookToCart function:***
- This function adds the selected book to user cart to.
- ***chechOut function:***
- This function takes the card number and its expiry date.
- Return true if the information is correct false otherwise.

```
@Override
public Cart addBookToCart(IBook newBook) {
    // TODO Auto-generated method stub
    this.user.getCart().add_items(newBook);
    return this.user.getCart();
}

@Override
public boolean chechOut(String card, String date) {
    // TODO Auto-generated method stub
    try {
        if(ft.parse(date).before(new Date())) {
            return false;
        }
    } catch (ParseException e) {
        // TODO Auto-generated catch block
        System.out.println(e);
        return false;
    }

    return this.checkOut();
}
```

- ***removeBookFromCart function:***
- This function removes the selected book from user cart to.
- ***clearCart function:***
- This function removes the selected book from user cart to.

```
@Override
public void clearCart() {
    // TODO Auto-generated method stub
    this.user.getCart().clear();

}

@Override
public Cart removeBookFromCart(IBook newBook) {
    // TODO Auto-generated method stub
    return this.user.getCart();
}

@Override
public Cart removeBookFromCart(int index) {
    // TODO Auto-generated method stub
    this.user.getCart().remove_item(index);
    return this.user.getCart();
}
```

- ***search function:***
  - This function to Search for books by any of the book's attributes.

```
@Override
public ArrayList<IBook> search(HashMap<String, Pair<String, String>> filters) {
    // TODO Auto-generated method stub
    ResultSet result = this.mySqlConnection.search_item(BOOKS_TABLE, null, filters);
    ArrayList<IBook>books = new ArrayList<IBook>();
    HashMap<String, IBook> map = new HashMap<>();

    try {
        while(result.next()) {
            int ISBN = result.getInt("ISBN");
            if(!map.containsKey(""+ISBN)) {
                IBook book = new Book();
                book.setISBN(ISBN);
                book.setTitle(result.getString("title"));
                book.setPublisher_name(result.getString("publisher_name"));
                book.setPublication_year(result.getString("publication_year"));
                book.setPrice(result.getInt("price"));
                book.setNo_Of_Books(result.getInt("NoOfBooks"));
                book.setCategory(cat.valueOf(result.getString("category")));
                book.addAuthor(result.getString("author"));
                map.put(ISBN+"", book);
                books.add(book);
            }else {
                map.get(Integer.toString(ISBN)).addAuthor(result.getString("author"));
            }
        }
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        System.out.println(e.toString());
        return new ArrayList<IBook>();
    }
    return books;
}
```

- ***These functions:***
  - Adding new books.
  - Update exited books.

```
@Override
public boolean addNewBook(IBook book) {
    // TODO Auto-generated method stub
    return mySqlConnection.insert_item(BOOKS_TABLE, book.getAttributes());
}

@Override
public boolean addNewBook(HashMap<String, String> attributes) {
    // TODO Auto-generated method stub
    return mySqlConnection.insert_item(BOOKS_TABLE, attributes);
}

@Override
public boolean updateBook(String ISBN, HashMap<String, String> attributes) {
    // TODO Auto-generated method stub
    HashMap<String, Pair<String, String>> conditions = new HashMap<>();
    conditions.put("ISBN", new Pair<String, String>("=", ISBN));
    return mySqlConnection.update_item(BOOKS_TABLE, attributes, conditions);
}
```

- ***These functions:***
  - Place orders.
  - Confirm orders.
  - Get orders.

```java
@Override
public boolean placeOrder(String ISBN, int amount) {
    // TODO Auto-generated method stub
    HashMap<String, String> attributes = new HashMap<>();
    attributes.put("ISBN", ISBN);
    attributes.put("NoOfBooks", ""+amount);
    return mySqlConnection.insert_item(ORDERS_TABLE, attributes);
}

@Override
public HashMap<String, String> getOrders() {
    // TODO Auto-generated method stub
    ResultSet result = this.mySqlConnection.search_item(ORDERS_TABLE, null, null);
    HashMap<String, String> orders = new HashMap<>();

    try {
        while(result.next()) {
            orders.put(result.getString("ISBN"), result.getString("NoOfBooks"));
        }
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        System.out.println(e);
    }
    return orders;
}
```

```java
@Override
public boolean confirmOrders(ArrayList<String> ISBNs) {
    // TODO Auto-generated method stub
    HashMap<String, Pair<String, String>> conditions = new HashMap<>();

    for(String ISBN : ISBNs) {
        conditions.put("ISBN", new Pair<String, String>("=", ISBN));
        mySqlConnection.delete_item(ORDERS_TABLE, conditions);
    }
    return true;
}
```

- ***These functions:***
  - Accept request of user to have managers credentials.
  - Reject request of user to have managers credentials.
  - Get user requests.

```java
    @Override
    public ArrayList<String> getdemandUsers() {
        // TODO Auto-generated method stub
        ResultSet result = this.mySqlConnection.search_item(PROMOTE_TABLE, null, null);
        ArrayList<String> users = new ArrayList<String>();
        try {
            while(result.next()) {
                users.add(result.getString("username"));
            }
        } catch (SQLException e) {
            // TODO Auto-generated catch block
//          e.printStackTrace();
            System.out.println(e);
        }
        return users;
    }

    @Override
    public boolean acceptUser(ArrayList<String> usernames) {
        // TODO Auto-generated method stub
        HashMap<String, Pair<String, String>> conditions = new HashMap<>();
        HashMap<String, String> attributes = new HashMap<>();
        attributes.put("type", "manager");
        for(String username : usernames) {
            conditions.put("username", new Pair<String, String>("=", username));
            mySqlConnection.delete_item(PROMOTE_TABLE, conditions);
            mySqlConnection.update_item(USERS_TABLE, attributes, conditions);
        }
        return true;
    }
```

```java
    @Override
    public boolean rejectUser(ArrayList<String> usernames) {
        // TODO Auto-generated method stub
        HashMap<String, Pair<String, String>> conditions = new HashMap<>();
        for(String username : usernames) {
            conditions.put("username", new Pair<String, String>("=", username));
            mySqlConnection.delete_item(PROMOTE_TABLE, conditions);
        }
        return true;
    }
```

- *My sql connection*
  - Singleton class identified in it the connection between the java and MySQL Server.
  - Contains seven functions which implements the search, insertion, delete, update from tables, commit, roll back, and printing SQL exception.

  - ***Search function:***
    Takes the table name, attributes of the table and conditions of the search. Return the result set of the search operation.

```java
public ResultSet search_item(String table, ArrayList<String> attributes,
        HashMap<String, Pair<String, String>> conditions) {
    // TODO Auto-generated method stub
    ResultSet rs = null;
    StringBuilder sqlBuilder = new StringBuilder();
    sqlBuilder.append("select ");
    if (attributes == null) {
        sqlBuilder.append("* ");
    } else {
        for (int i = 0; i < attributes.size(); i++) {
            if (i == attributes.size() - 1) {
                sqlBuilder.append(attributes.get(i));
            }
            sqlBuilder.append(attributes.get(i) + ", ");
        }
    }
    sqlBuilder.append("from " + table);
    if ("book".equals(table)) {
        sqlBuilder.append(" natural join authors ");
    }
    if (conditions != null && !conditions.isEmpty()) {
        sqlBuilder.append(" where ");
        for (Entry<String, Pair<String, String>> e:conditions.entrySet()) {
            sqlBuilder.append(e.getKey());
            sqlBuilder.append(e.getValue().getKey());
            sqlBuilder.append("\'");
            sqlBuilder.append(e.getValue().getValue());
            sqlBuilder.append("\'");
            sqlBuilder.append(" AND ");
        }
        sqlBuilder.delete(sqlBuilder.length() - 4, sqlBuilder.length());
    }
    try {

        System.out.println(sqlBuilder.toString());
        rs = stmt.executeQuery(sqlBuilder.toString());
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        printSQLException(e);
    }
    return rs;
}
```

- ***Insert function:***

  Takes the table name and attributes of the table.
  Return the Boolean that the insertion is been done in right way or not.

```java
public boolean insert_item(String table, HashMap<String, String> attributes) {
    // TODO Auto-generated method stub
    String temp = "";
    StringBuilder sqlBuilder = new StringBuilder();
    sqlBuilder.append("insert into " + table + " (");

    if (!attributes.isEmpty()) {
        if (attributes.containsKey("author")) {
            temp = attributes.get("author");
            attributes.remove("author");
        }
        for (Entry<String, String> a:attributes.entrySet()) {
            sqlBuilder.append(a.getKey() + ", ");
        }

        sqlBuilder.replace(sqlBuilder.length() - 2, sqlBuilder.length() - 1, ")");
        sqlBuilder.append(" values (");

        for (Entry<String, String> a:attributes.entrySet()) {
            sqlBuilder.append("\'");
            sqlBuilder.append(a.getValue() + "\', ");
        }

        sqlBuilder.replace(sqlBuilder.length() - 2, sqlBuilder.length() - 1, ")");

        try {
            System.out.println(sqlBuilder.toString());

            if("book".equals(table)) {
                stmt.executeUpdate(sqlBuilder.toString(), Statement.RETURN_GENERATED_KEYS);
            }else {
                stmt.executeUpdate(sqlBuilder.toString());
            }

        } catch (SQLException e) {
            // TODO Auto-generated catch block
            rollback();
            printSQLException(e);
            return false;
        }
```

```
            if (!temp.isEmpty()) {
//              "select last_insert_id() as last_id;"
                int ISBN = 0;
                try {
                    ResultSet rs = stmt.getGeneratedKeys();
                    if(rs.next()) {
                        ISBN = rs.getInt(1);
                    }
                } catch (SQLException e1) {
                    // TODO Auto-generated catch block
                    rollback();
                    printSQLException(e1);
                    return false;
                }
                String [] authors = temp.split(",");
                for (int i = 0; i < authors.length; i++) {
                    sqlBuilder = new StringBuilder();
                    sqlBuilder.append("insert into authors values (\'");
                    sqlBuilder.append(ISBN + "\', \'");
                    sqlBuilder.append(authors[i] + "\')");
                    try {

                        System.out.println(sqlBuilder.toString());

                        stmt.executeUpdate(sqlBuilder.toString());
                    } catch (SQLException e) {
                        // TODO Auto-generated catch block
                        rollback();
                        printSQLException(e);
                        return false;
                    }
                }
            }
        }
        commit();
        return true;
    }
```

- ***Update function:***

  Takes the table name, attributes of the table and conditions of the update. Return the Boolean that the update is been done in right way or not.

```java
public boolean update_item(String table, HashMap<String, String> attributes,
        HashMap<String, Pair<String, String>> conditions) {
    // TODO Auto-generated method stub
    StringBuilder sqlBuilder = new StringBuilder();
    sqlBuilder.append("update ");
    sqlBuilder.append(table);

    if (!attributes.isEmpty()) {

        if ("book".equals(table)) {
            sqlBuilder.append(" natural join authors ");
        }

        sqlBuilder.append(" set ");
        for (Entry<String, String> a:attributes.entrySet()) {
            sqlBuilder.append(a.getKey() + "='" + a.getValue() + "', ");
        }
        sqlBuilder.delete(sqlBuilder.length() - 2, sqlBuilder.length() - 1);
        sqlBuilder.append(" where ");
        for (Entry<String, Pair<String, String>> e:conditions.entrySet()) {
            sqlBuilder.append(e.getKey());
            sqlBuilder.append(e.getValue().getKey());
            sqlBuilder.append("'");
            sqlBuilder.append(e.getValue().getValue());
            sqlBuilder.append("'");
            sqlBuilder.append(" AND ");
        }
        sqlBuilder.delete(sqlBuilder.length() - 4, sqlBuilder.length());
        try {

            System.out.println(sqlBuilder.toString());
            stmt.executeUpdate(sqlBuilder.toString());
        } catch (SQLException e1) {
            // TODO Auto-generated catch block
            rollback();
            printSQLException(e1);
            return false;
        }
    }
    commit();
    return true;
}
```

- ***Delete function:***

  Takes the table name and conditions of the search.
  Return the Boolean that the delete is been done in right way or not.

```
public boolean delete_item(String table, HashMap<String, Pair<String, String>> conditions) {
    // TODO Auto-generated method stub
    StringBuilder sqlBuilder = new StringBuilder();
    sqlBuilder.append("delete ");

    if ("book".equals(table)) {
        sqlBuilder.append("book, authors from (book natural join authors) ");
    } else {
        sqlBuilder.append(" from ");
        sqlBuilder.append(table);
    }

    if (!conditions.isEmpty()) {
        sqlBuilder.append(" where ");
        for (Entry<String, Pair<String, String>> e:conditions.entrySet()) {
            sqlBuilder.append(e.getKey());
            sqlBuilder.append(e.getValue().getKey());
            sqlBuilder.append("\'");
            sqlBuilder.append(e.getValue().getValue());
            sqlBuilder.append("\'");
            sqlBuilder.append(" AND ");
        }
        sqlBuilder.delete(sqlBuilder.length() - 4, sqlBuilder.length());
        try {

            System.out.println(sqlBuilder.toString());

            stmt.executeUpdate(sqlBuilder.toString());
        } catch (SQLException e1) {
            // TODO Auto-generated catch block
            rollback();
            printSQLException(e1);
            return false;
        }
    }
    commit();
    return true;
}
```

- ***Print SQL Exception function:***

  Prints the exception expected from the SQL Server.

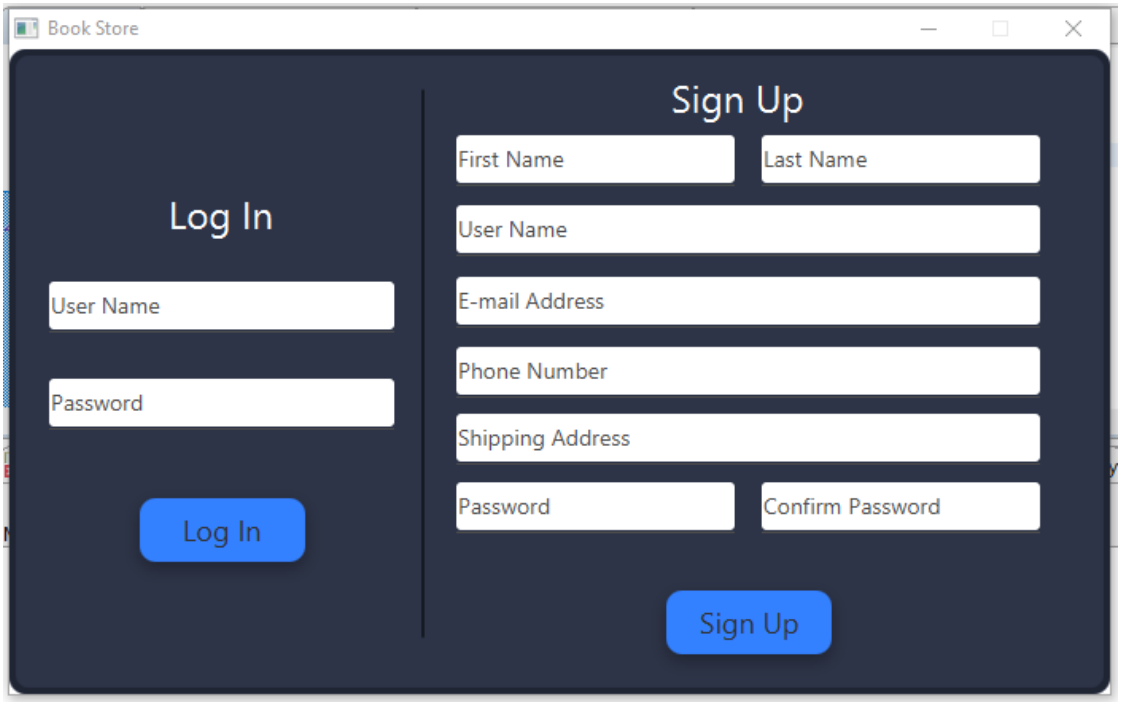
```
    public static void printSQLException(SQLException ex) {
        for (Throwable e: ex) {
            if (e instanceof SQLException) {
                e.printStackTrace(System.err);
                System.err.println("SQLState: " + ((SQLException) e).getSQLState());
                System.err.println("Error Code: " + ((SQLException) e).getErrorCode());
                System.err.println("Message: " + e.getMessage());
                Throwable t = ex.getCause();
                while (t != null) {
                    System.out.println("Cause: " + t);
                    t = t.getCause();
                }
            }
        }
    }

}
```

- ***Commit and Roll back functions:***

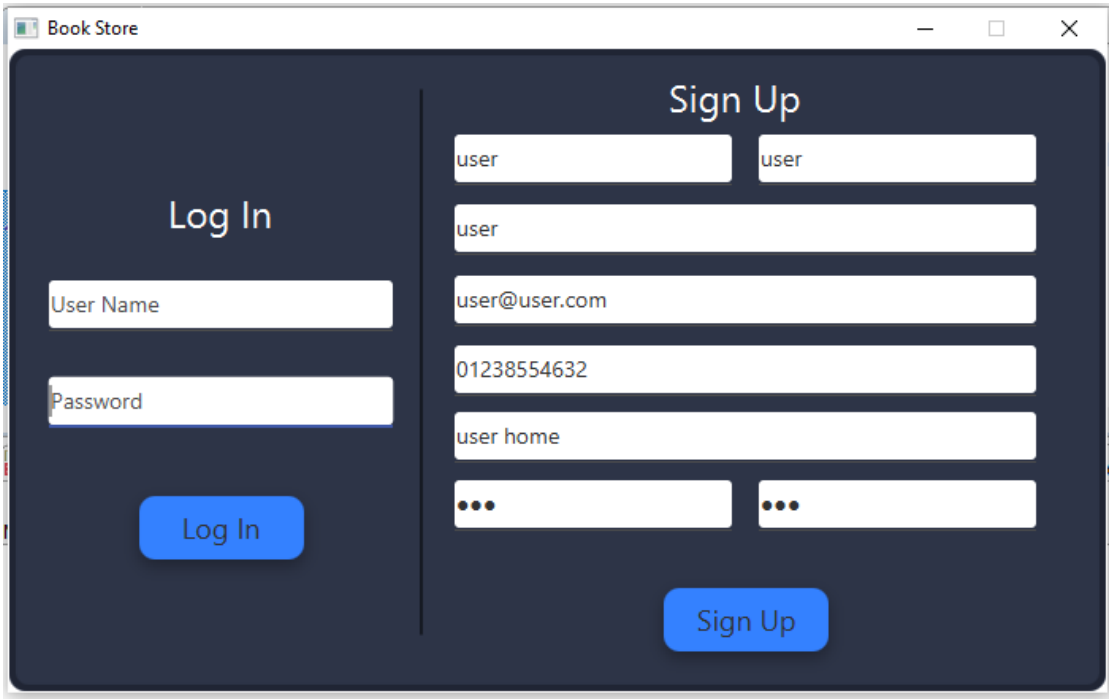
- To commit any action or any transaction did in database.
- To help if there is an error to rollback again to the last consistent point.

```
    private void commit() {
        if(conn != null) {
            try {
                conn.commit();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                printSQLException(e);
            }
        }
    }

    private void rollback() {

        if(conn != null) {
            try {
                conn.rollback();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                printSQLException(e);
            }
        }
    }

}
```

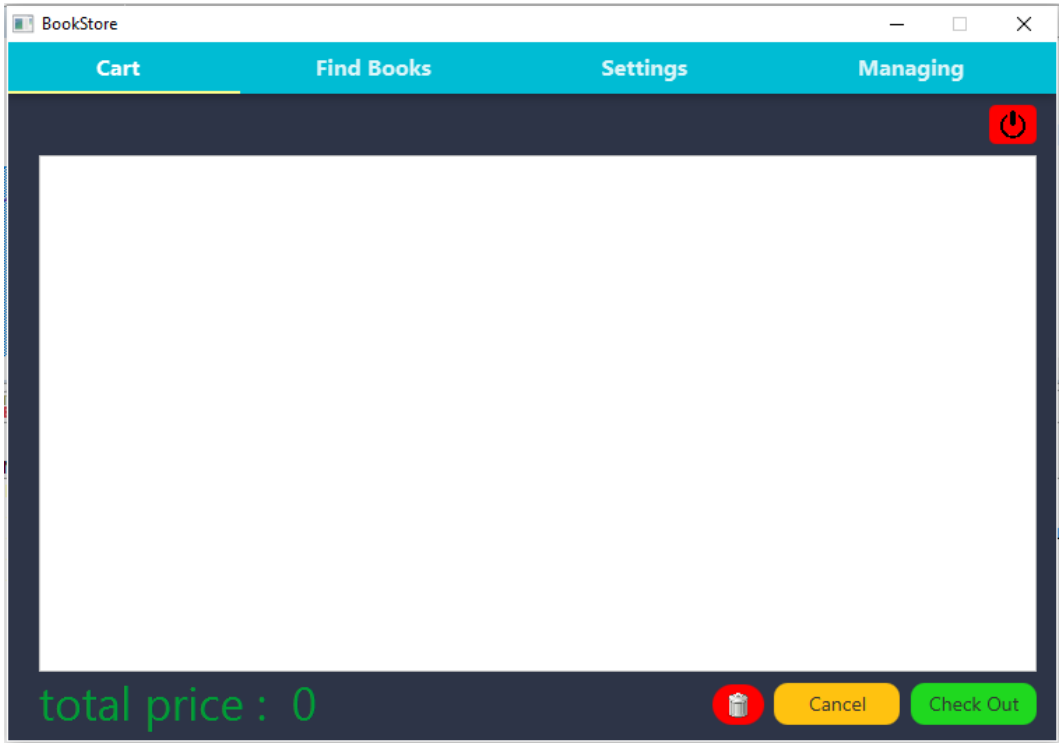
- **GUI:**
  - **Login view**



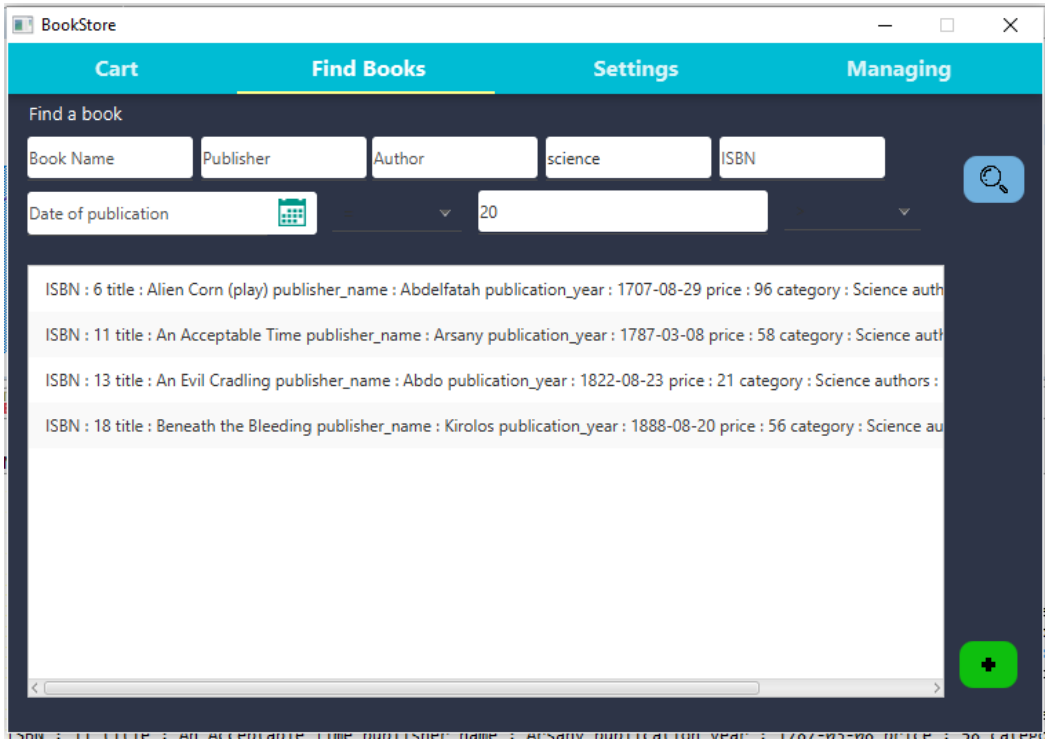
  - **SignUp**



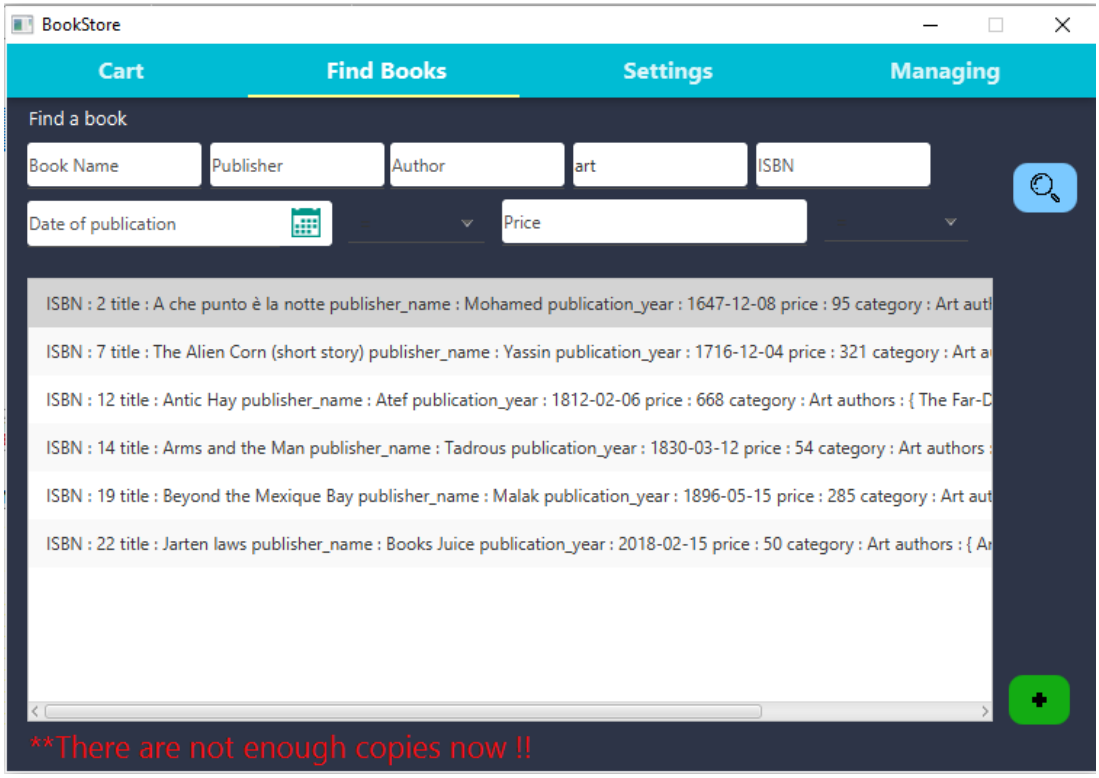
- **Free cart**



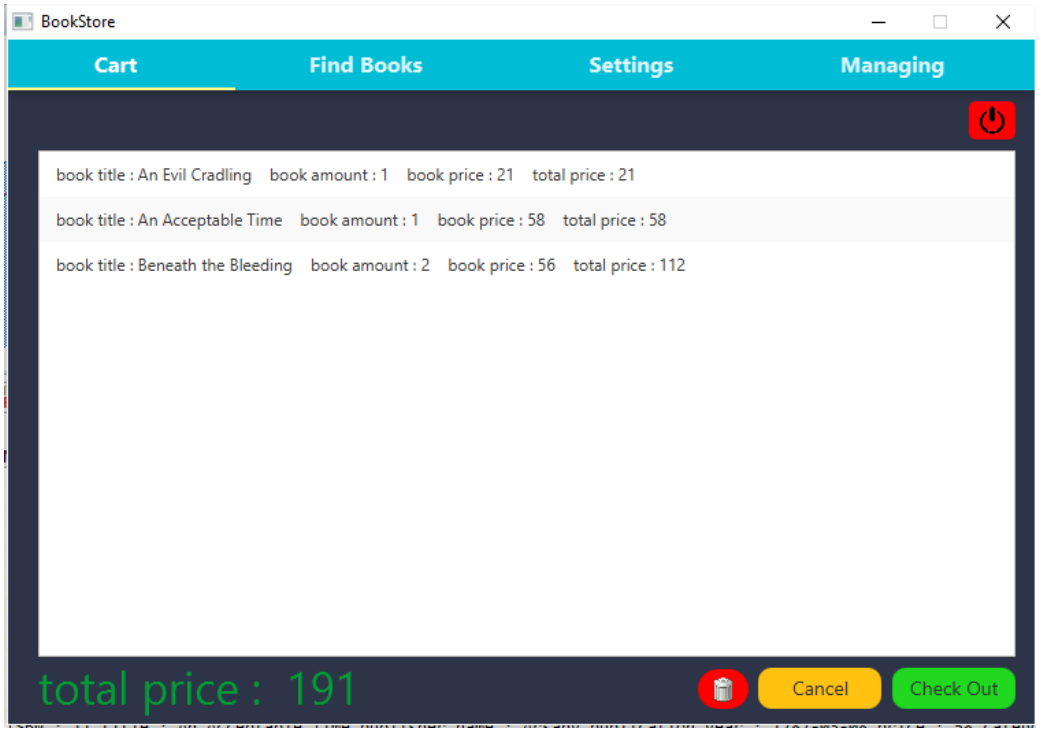
- **Search for a book by category**



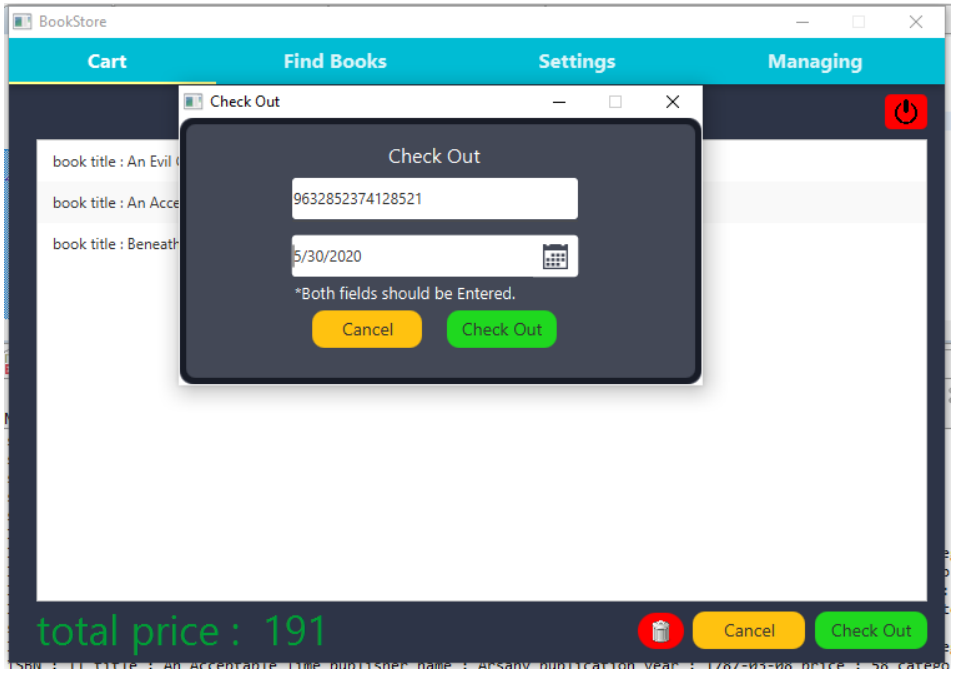
- **Book out of stock**



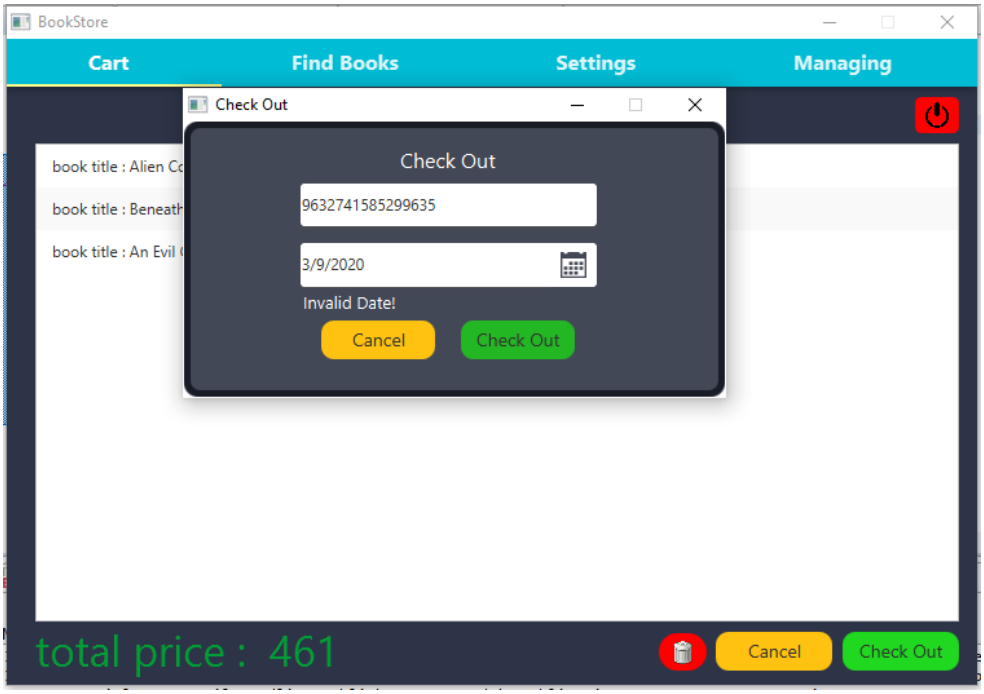
- **Cart with books**



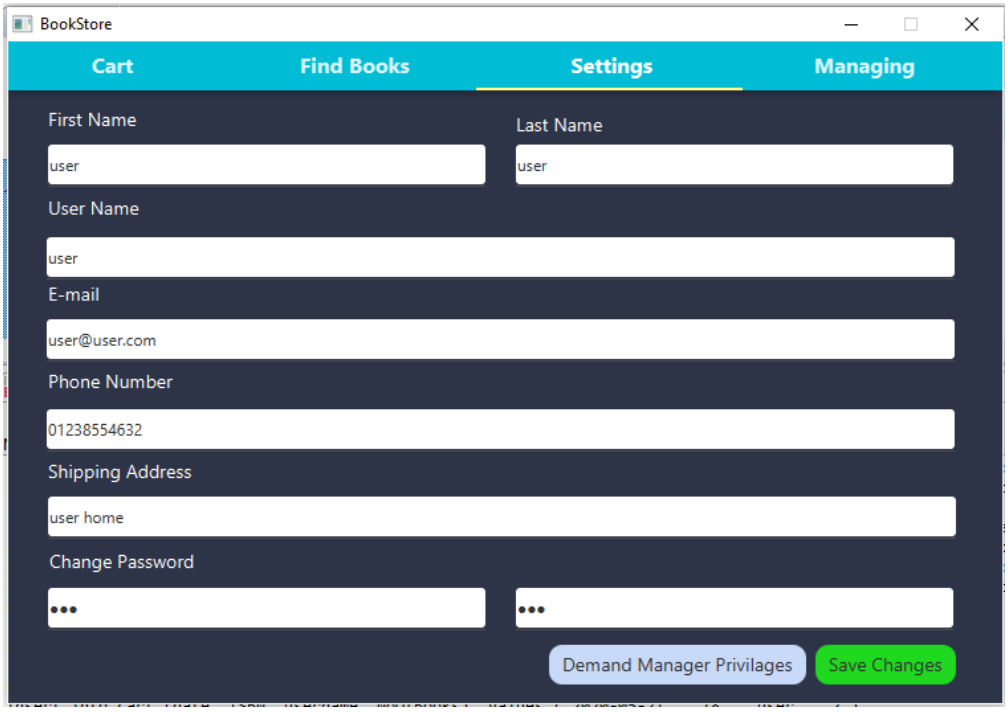
- **Checkout**



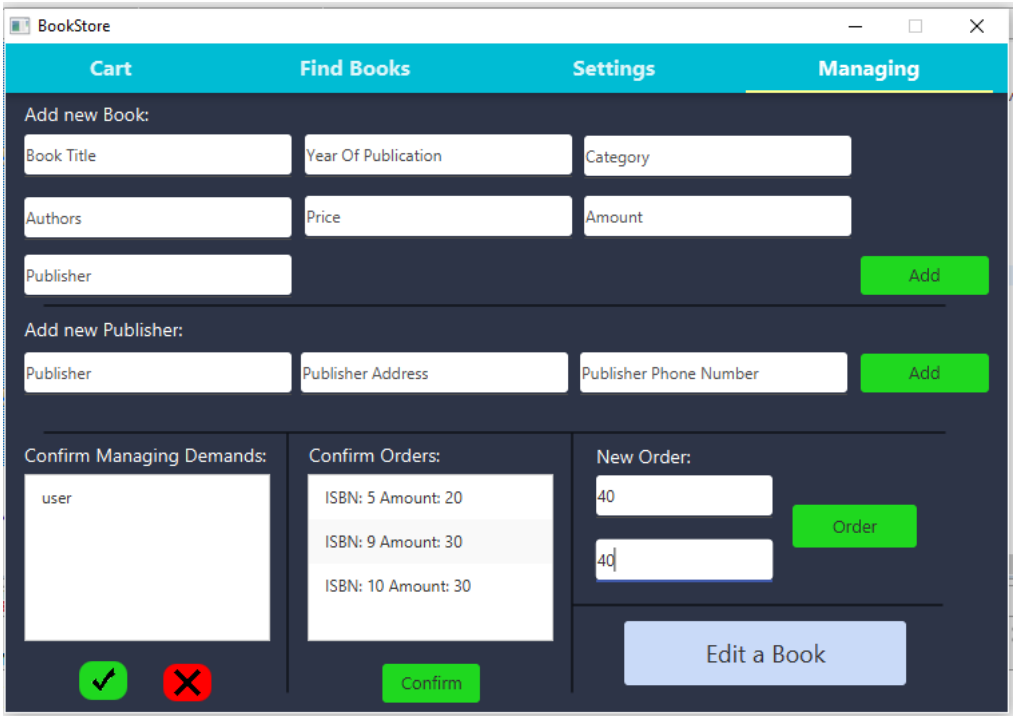
- **Checkout with invalid credit card**

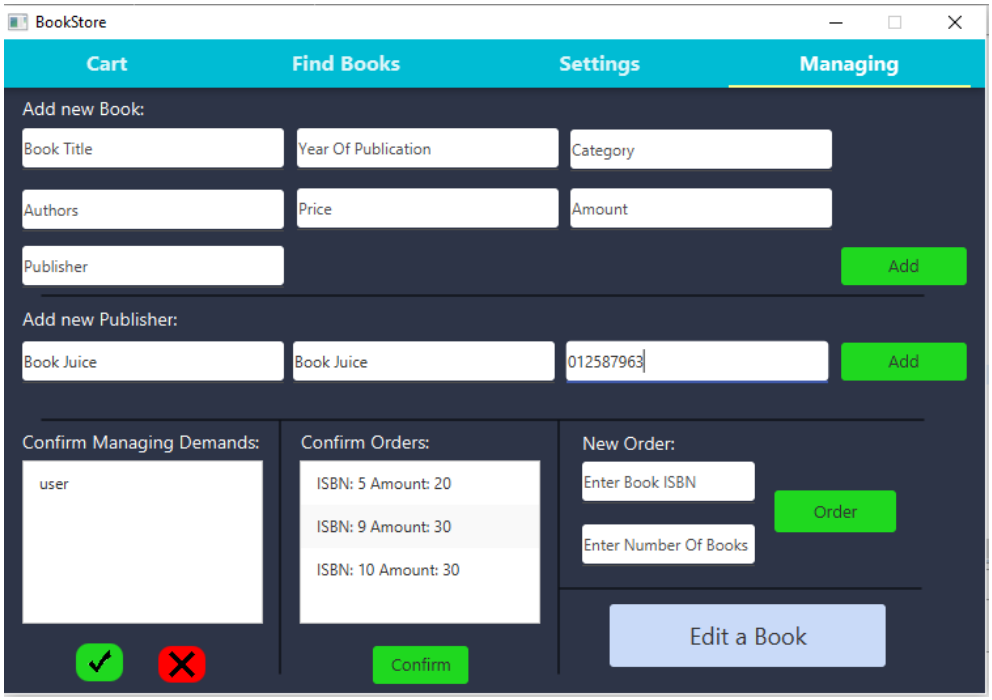


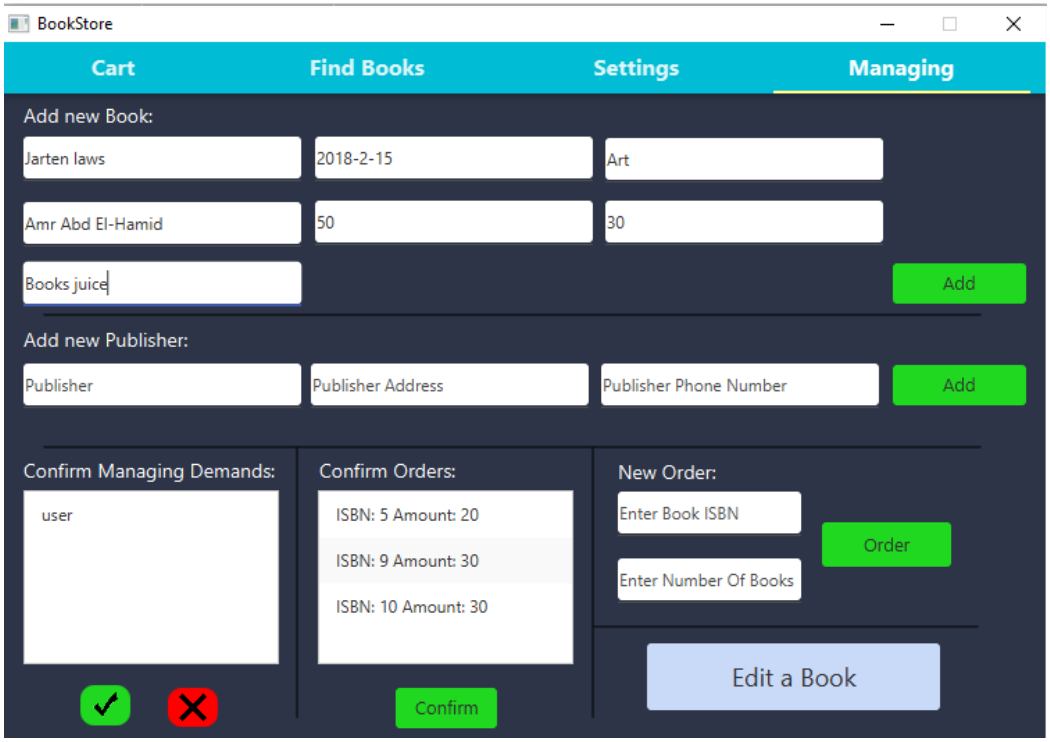
- **User settings**



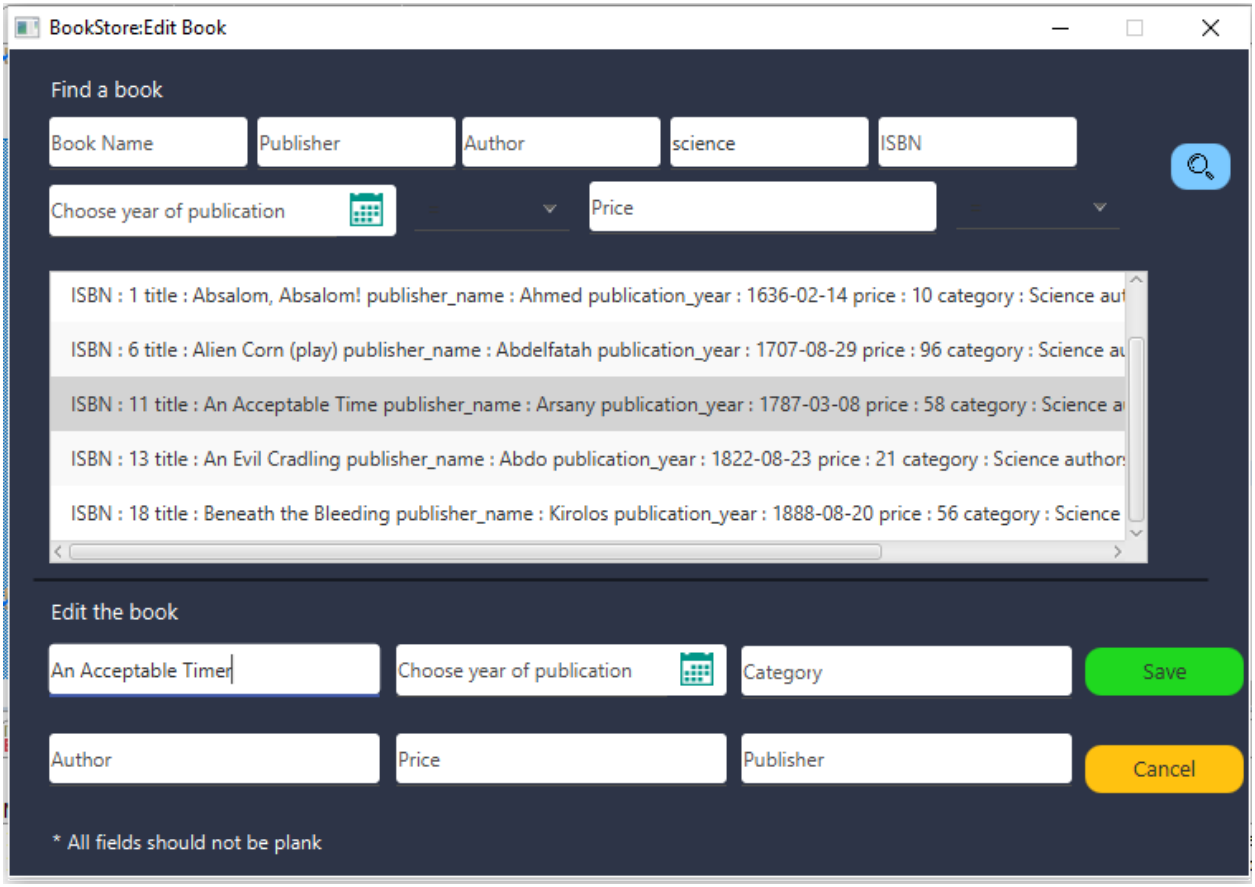
- **Managing the store frame which is available to the manager**



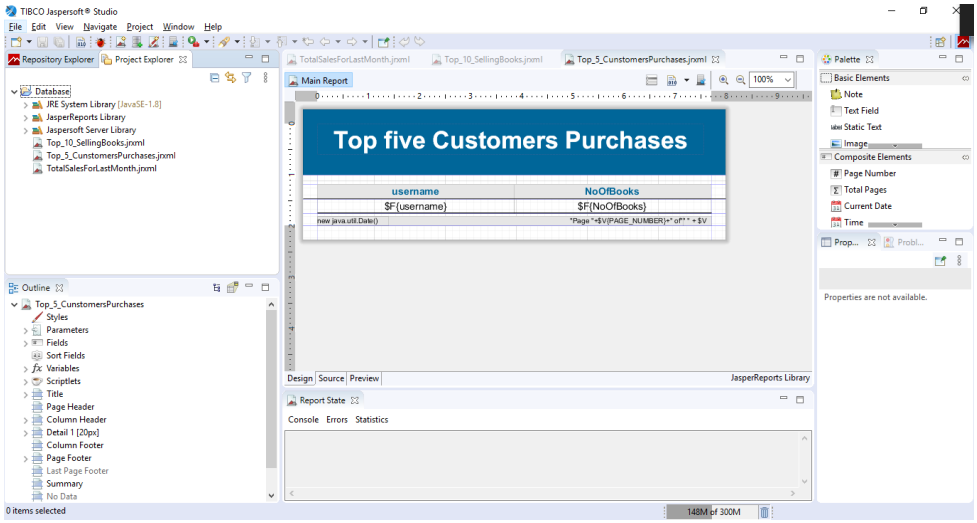

BookStore
Cart
Find Books
Settings
Managing
Add new Book:
Book Title
Year Of Publication
Category
Authors
Price
Amount
Publisher
Add
Add new Publisher:
Book Juice
Book Juice
012587963
Add
Confirm Managing Demands:
user
Confirm Orders:
ISBN: 5 Amount: 20
ISBN: 9 Amount: 30
ISBN: 10 Amount: 30
Confirm
New Order:
Enter Book ISBN
Enter Number Of Books
Order
Edit a Book


BookStore
Cart
Find Books
Settings
Managing
Add new Book:
Jarten laws
2018-2-15
Art
Amr Abd El-Hamid
50
30
Books juice
Add
Add new Publisher:
Publisher
Publisher Address
Publisher Phone Number
Add
Confirm Managing Demands:
user
Confirm Orders:
ISBN: 5 Amount: 20
ISBN: 9 Amount: 30
ISBN: 10 Amount: 30
Confirm
New Order:
Enter Book ISBN
Enter Number Of Books
Order
Edit a Book

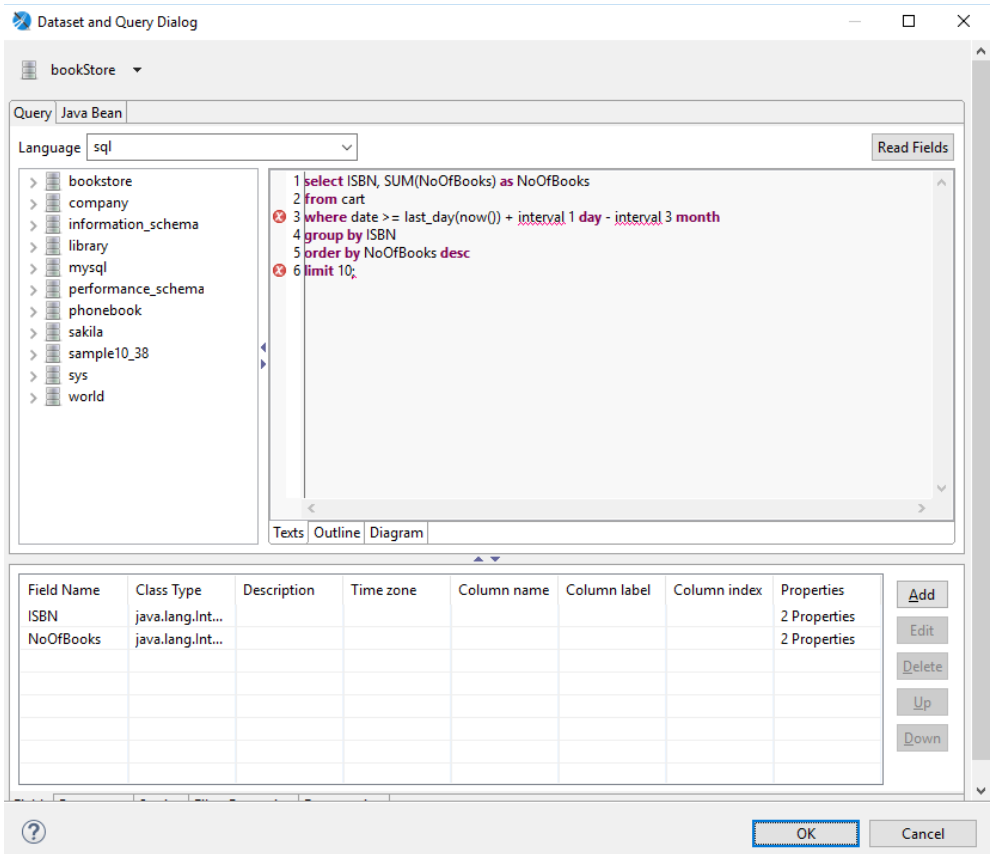
- **Search on a book to edit**

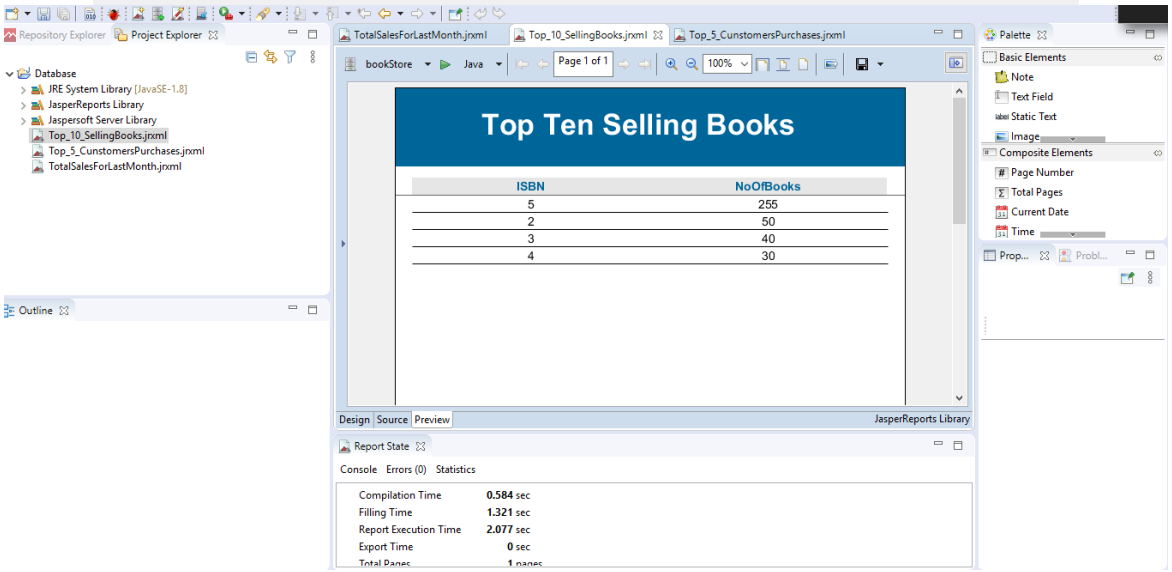


○ Jasper using Jaspersoft studio:

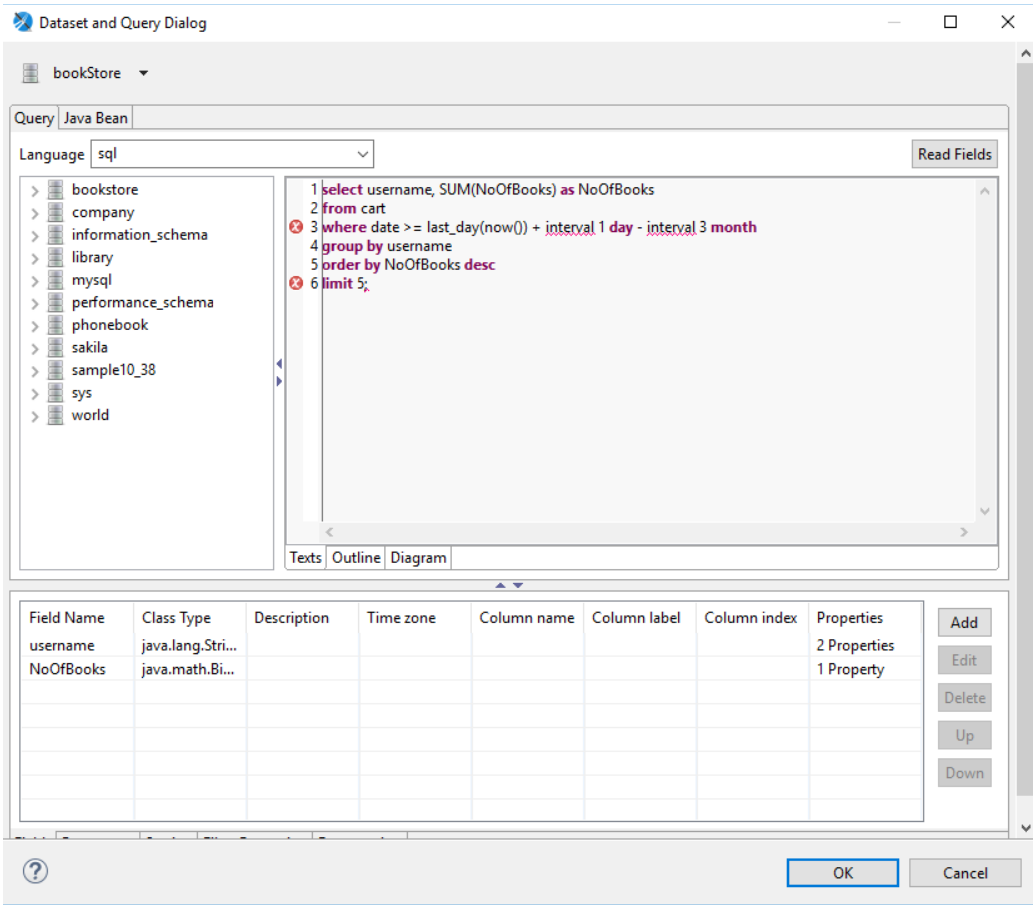


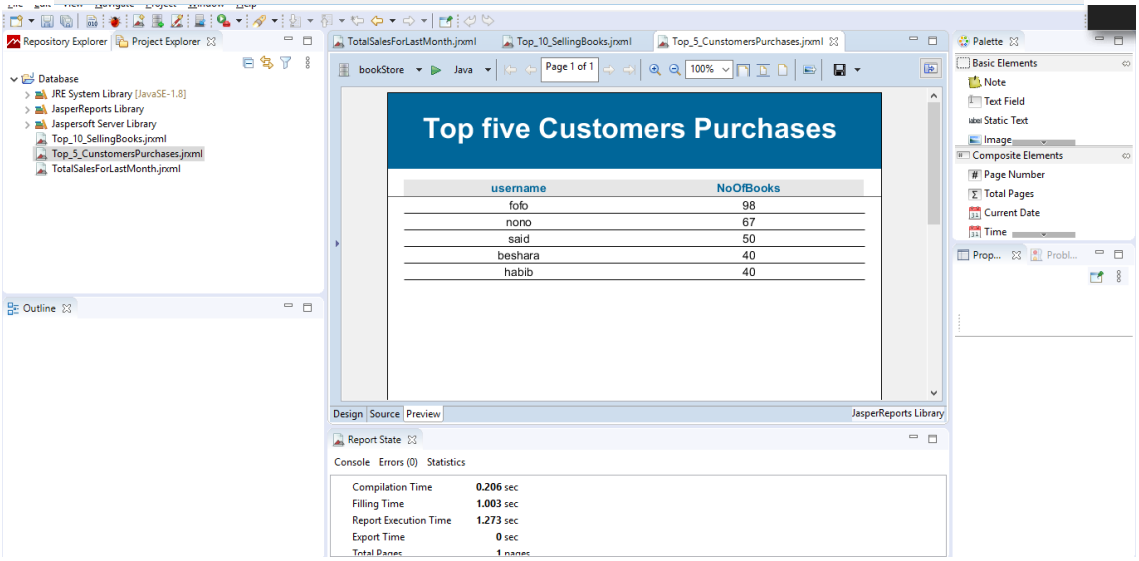
- **Top 10 selling books:**





- **Top 5 customers purchase:**





- **Total sales for last month**

Dataset and Query Dialog

bookStore

Query | Java Bean

Language: sql

Read Fields

```
select *
from cart c
where
/* Greater or equal to the start of last month */
c.date >= last_day(now()) + interval 1 day - interval 2 month;
```

Field Name	Class Type	Description	Time zone	Column name	Column label	Column index	Properties
cartid	java.lang.Int...						2 Properties
username	java.lang.Stri...						2 Properties
ISBN	java.lang.Int...						2 Properties
NoOfBooks	java.lang.Int...						2 Properties
date	java.sql.Date						2 Properties

Add | Edit | Delete | Up | Down

OK | Cancel

Database
- JRE System Library [JavaSE-1.8]
- JasperReports Library
- Jaspersoft Server Library
- Top_10_SellingBooks.jrxml
- Top_5_CunstomersPurchases.jrxml
- TotalSalesForLastMonth.jrxml

Outline

bookStore | Java | Page 1 of 1 | 100%

Last sales in previous month

cartid	username	ISBN	NoOfBooks	date
5	said	2	50	4/5/20 12:00 AM
6	beshara	3	40	4/5/20 12:00 AM
7	malak	4	30	4/5/20 12:00 AM
8	habib	5	20	4/5/20 12:00 AM
9	habib	5	20	4/15/20 12:00 AM
10	andrew	5	20	5/15/20 12:00 AM
11	ahmed	5	20	4/10/20 12:00 AM
14	nono	5	67	5/20/20 12:00 AM

Design | Source | Preview | JasperReports Library

Report State

Console | Errors (0) | Statistics

Compilation Time	0.184 sec
Filling Time	0.976 sec
Report Execution Time	1.253 sec
Export Time	0 sec
Total Pages	1 pages

Basic Elements: Note, Text Field, Static Text, Image

Composite Elements: Page Number, Total Pages, Current Date, Time