



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

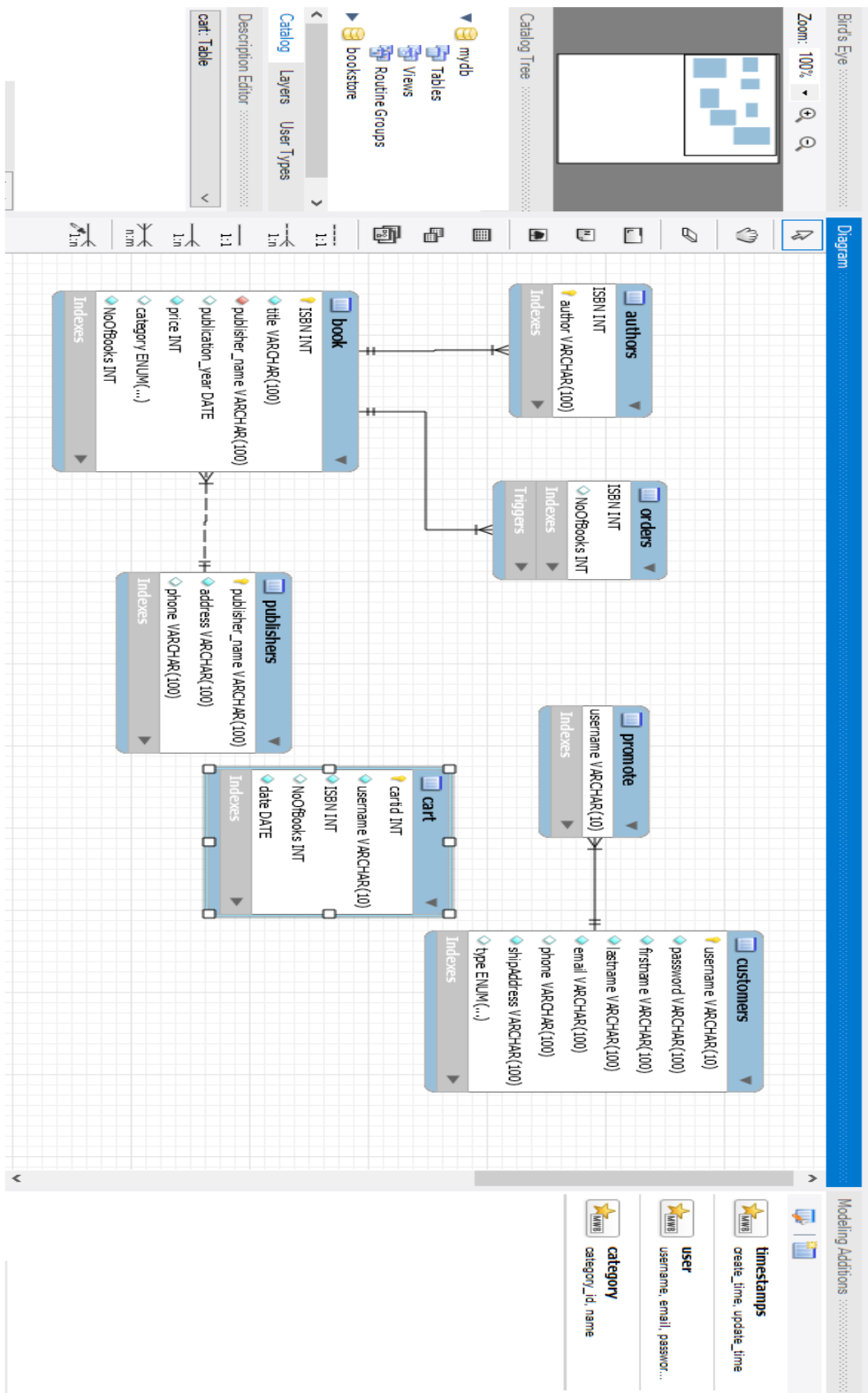
Links of Jasper files

I open them with atom editor

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

https://drive.google.com/drive/folders/1KbhIU_T6WdUKzCTMPHZ_Ef4sxIVKvDjrQ?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:

The screenshots show a SQL IDE interface with a Navigator pane on the left, a central SQL editor, and a right-hand pane with context help.

Top Screenshot:

- Navigator:** Shows the 'bookstore' schema with tables: authors, book, cart, customers, orders, promote, publishers. Views: Stored Procedures, Functions. Other schemas: company, library, phonebook, sakila, sample10_38, sys, world.
- SQL Editor:** Contains the following SQL code:

```
1 • Create schema bookstore;
2 • use bookstore;
3
4 # -----
5 # -----PART 1-----
6 # -----
7 • Create Table publishers (
8   publisher_name Varchar (100) primary key Not Null,
9   address Varchar(100) Not Null,
10  phone Varchar(100));
11
12 #---- may be change ISBN to INT -----
13
14 • Create Table book (
15   ISBN int AUTO_INCREMENT Primary Key Not Null,
16   title Varchar(100) Not Null,
17   publisher_name Varchar (100) Not Null,
18   publication_year Date,
19   price INT default 0 Not Null,
20   category ENUM ('Science', 'Art', 'Religion', 'History', 'Geography'),
21   NoOfBooks INT default 20 Not Null,
22   constraint fk1 foreign key (publisher_name) references publishers (publisher_name) on delete cascade on update cascade);
23
24 • Create Table authors (
25   ISBN int Not Null,
26   author Varchar(100) Not Null,
27   constraint pk1 primary key (ISBN, author),
28   constraint fk2 foreign key (ISBN) references book (ISBN) on delete cascade on update cascade);
29
30 • Create Table orders (
31   ISBN int Primary Key Not Null,
32   NoOfBooks INT default 20,
33   constraint fk3 foreign key (ISBN) references book (ISBN) on delete cascade on update cascade);
34
```
- Right Pane:** Displays context help for the current caret position, stating: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

Bottom Screenshot:

- Navigator:** Same as the top screenshot.
- SQL Editor:** Contains the following SQL code:

```
37 # -----
38 • Create Table customers (
39   username Varchar (10) Primary Key Not Null,
40   password Varchar(100) Not Null,
41   firstname Varchar (100) Not Null,
42   lastname Varchar (100) Not Null,
43   email Varchar (100) Not Null,
44   phone Varchar(100),
45   shipAddress Varchar(100) Not Null,
46   type ENUM ('user', 'manager'));
47
48 • Create Table cart (
49   cartid int AUTO_INCREMENT PRIMARY KEY,
50   username Varchar (10) Not Null,
51   ISBN int Not Null ,
52   NoOfBooks INT,
53   date Date Not Null);
54
55 • Create Table promote (
56   username Varchar (10) Primary Key Not Null,
57   constraint fk6 foreign key (username) references customers (username) on delete cascade on update cascade);
58
```
- Right Pane:** Same as the top screenshot, displaying context help.

○ Triggers screenshots:

The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'SCHEMAS' pane shows a tree view with 'bookstore' expanded, containing tables like 'authors', 'book', 'cart', 'customers', 'orders', 'promote', and 'publishers'. The 'Information' pane below it shows 'No object selected'. The main editor window, titled 'book schema triggers', shows the following SQL code:

```
1 Use bookstore;
2
3 DELIMITER //
4 CREATE TRIGGER check_no_books before update ON book FOR EACH ROW
5 BEGIN
6     IF
7         new.NoOfBooks < 0
8     THEN
9         SIGNAL SQLSTATE '45000'
10        SET MESSAGE_TEXT = 'Number of books is not available by now';
11    END IF;
12 END //
13 DELIMITER ;
14
15 DELIMITER //
16 CREATE TRIGGER place_order after update ON book FOR EACH ROW
17 BEGIN
18     IF
19         new.NoOfBooks <= 5
20     THEN
21         insert into orders values (new.ISBN, 20);
22     END IF;
23 END //
24 DELIMITER ;
25
26 DELIMITER //
27
28
29 CREATE TRIGGER confirm_order before delete ON orders FOR EACH ROW
30 BEGIN
31     update book set NoOfBooks = NoOfBooks + old.NoOfBooks where ISBN = old.ISBN;
32 END //
33 DELIMITER ;
34
```

On the right, the 'SQLAdditions' pane displays a message: 'Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.'

○ 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.

```
@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.

```
@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.

- **checkOut 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 checkOut(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.

```
@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;
}
```

```
@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.

```
@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;
}
```

```
@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.

```
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.

```
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.

```
// Update function
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:**

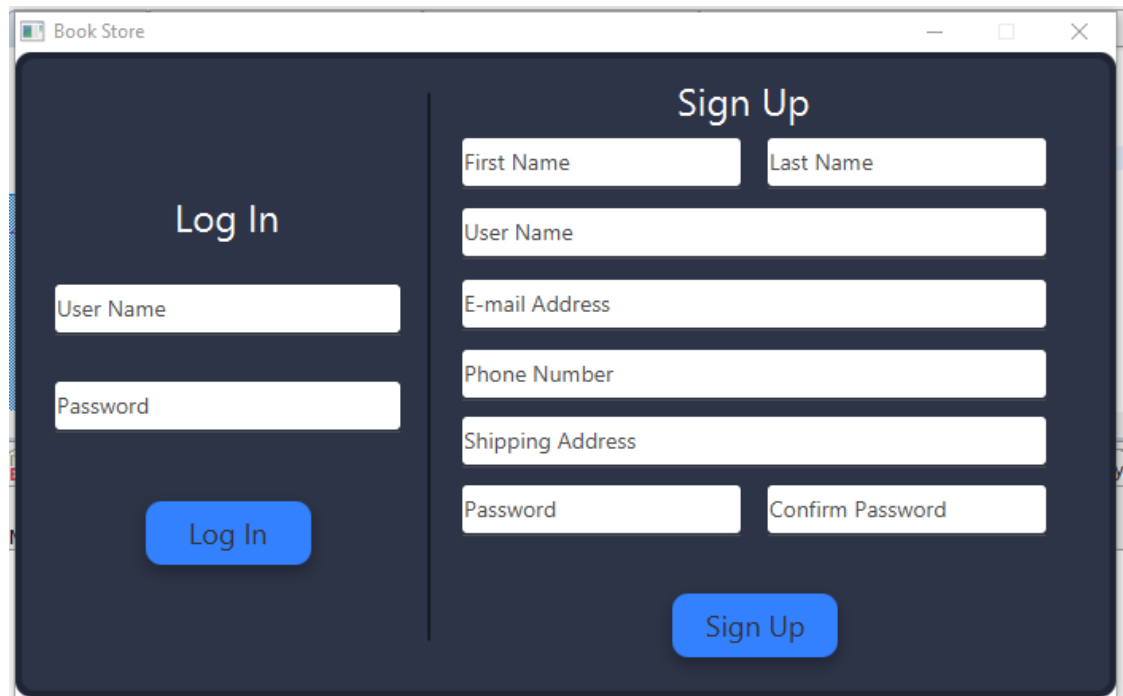
- To commit any action or any transaction did in database.
- To help if there is an error to rollback again to the last consistent point.

```
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);
        }
    }
}
```

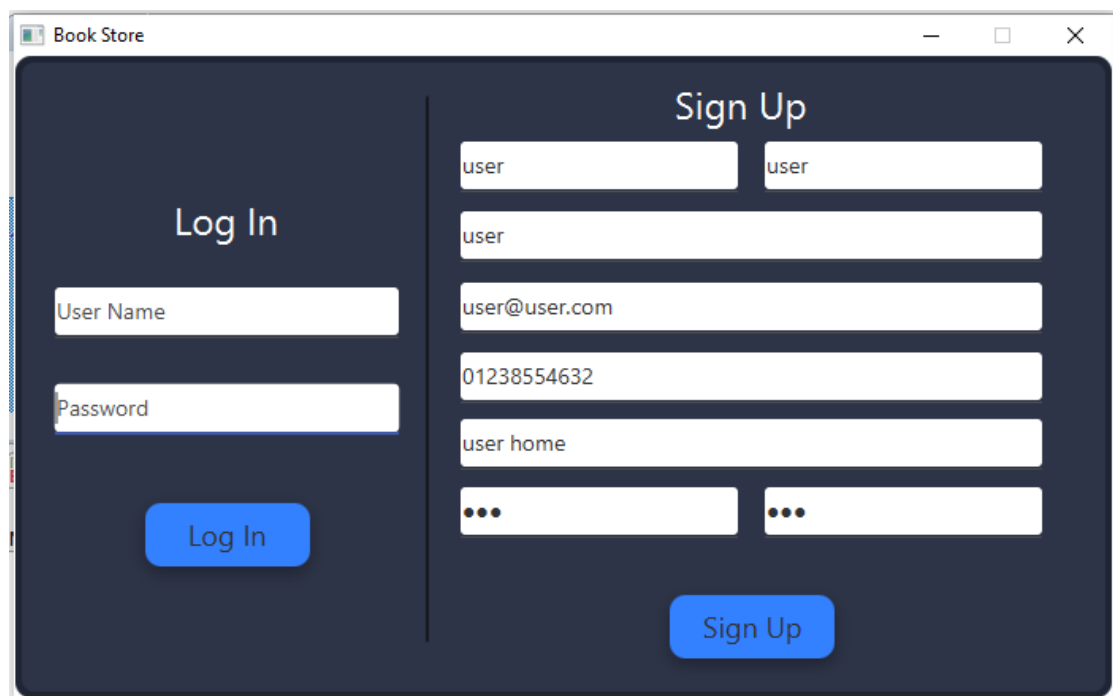
- GUI:

- Login view



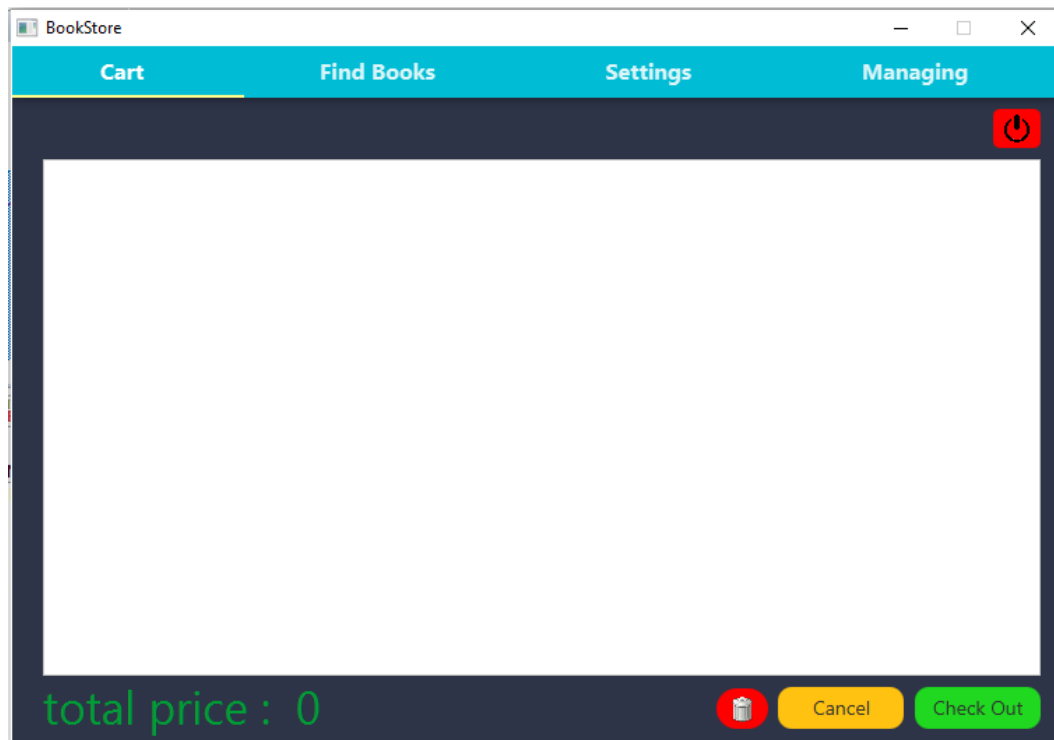
The screenshot shows a window titled "Book Store" with a dark blue background. On the left, the "Log In" section has two input fields: "User Name" and "Password", followed by a blue "Log In" button. On the right, the "Sign Up" section has several input fields: "First Name" and "Last Name" (side-by-side), "User Name", "E-mail Address", "Phone Number", "Shipping Address", "Password", and "Confirm Password", followed by a blue "Sign Up" button.

- SignUp

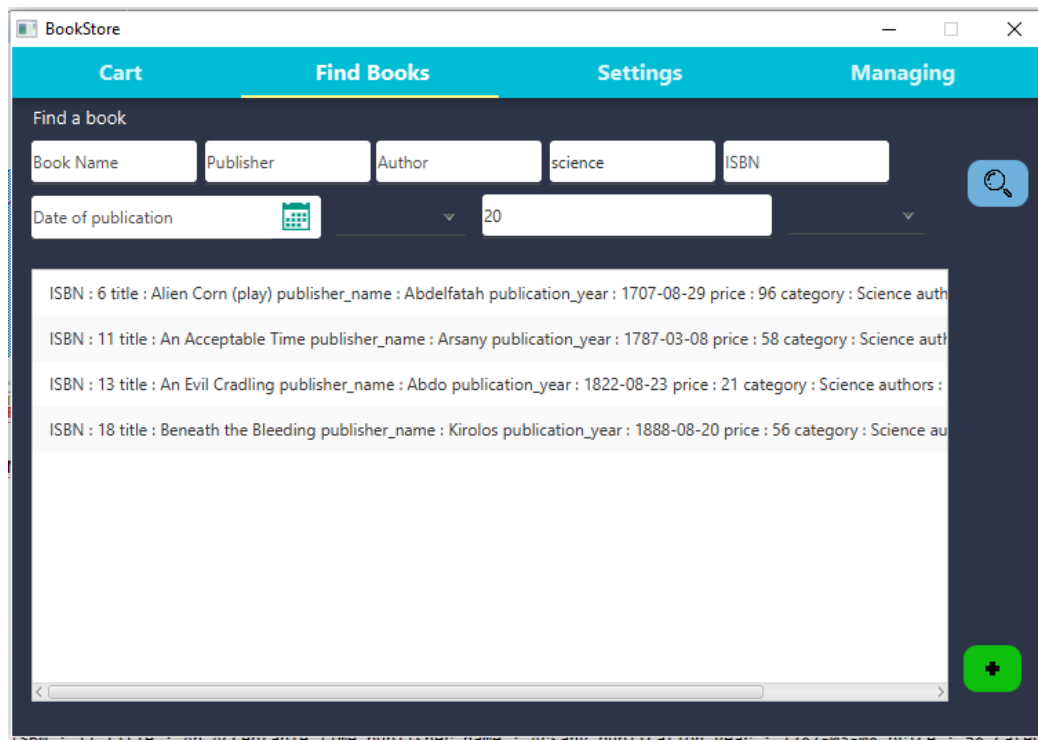


The screenshot shows the same "Book Store" window, but the "Sign Up" form is now filled with sample data. The "First Name" field contains "user", "Last Name" contains "user", "User Name" contains "user", "E-mail Address" contains "user@user.com", "Phone Number" contains "01238554632", and "Shipping Address" contains "user home". The "Password" and "Confirm Password" fields are masked with three dots. The "Log In" section remains empty.

- Free cart



- Search for a book by category



- Book out of stock

BookStore

Cart Find Books Settings Managing

Find a book

Book Name Publisher Author art ISBN

Date of publication Price

ISBN : 2 title : A che punto è la notte publisher_name : Mohamed publication_year : 1647-12-08 price : 95 category : Art autl

ISBN : 7 title : The Alien Corn (short story) publisher_name : Yassin publication_year : 1716-12-04 price : 321 category : Art a

ISBN : 12 title : Antic Hay publisher_name : Atef publication_year : 1812-02-06 price : 668 category : Art authors : { The Far-D

ISBN : 14 title : Arms and the Man publisher_name : Tadrous publication_year : 1830-03-12 price : 54 category : Art authors :

ISBN : 19 title : Beyond the Mexique Bay publisher_name : Malak publication_year : 1896-05-15 price : 285 category : Art aut

ISBN : 22 title : Jarten laws publisher_name : Books Juice publication_year : 2018-02-15 price : 50 category : Art authors : { A

***There are not enough copies now !!

- Cart with books

BookStore

Cart Find Books Settings Managing

book title : An Evil Cradling book amount : 1 book price : 21 total price : 21

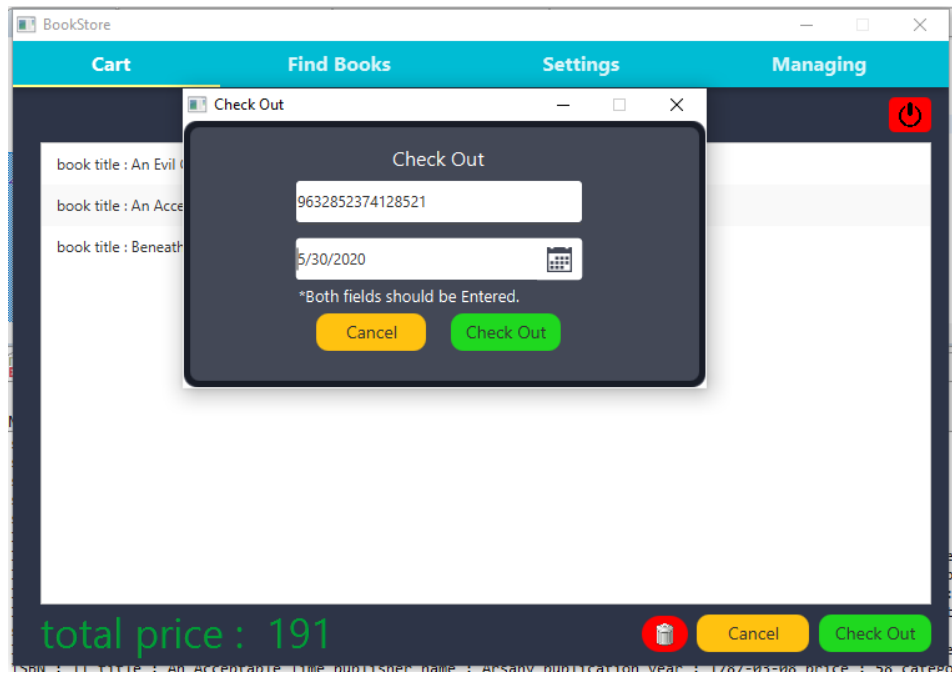
book title : An Acceptable Time book amount : 1 book price : 58 total price : 58

book title : Beneath the Bleeding book amount : 2 book price : 56 total price : 112

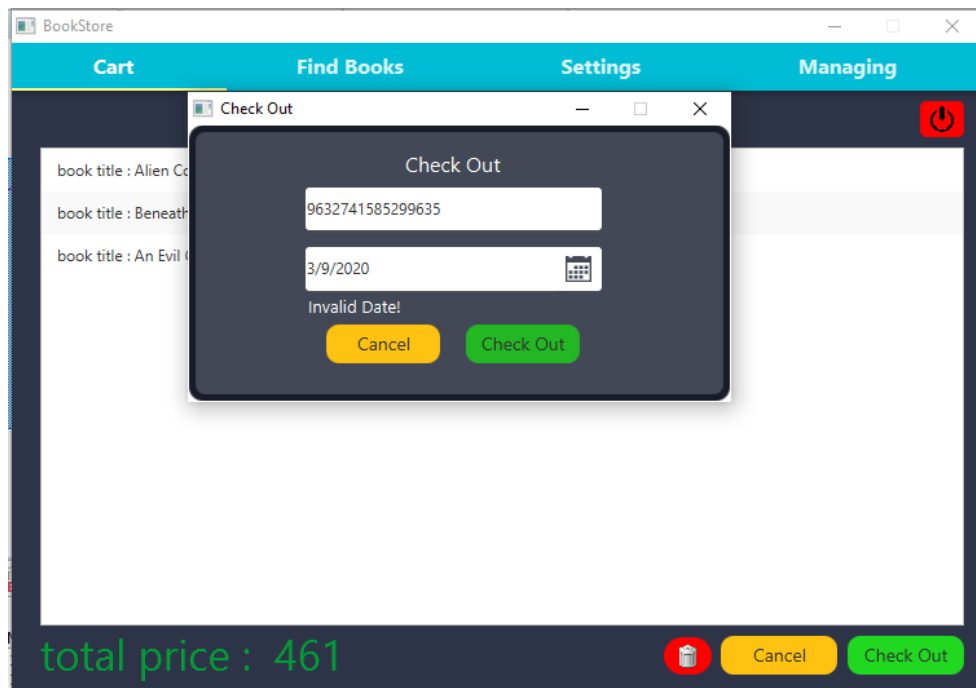
total price : 191

Cancel Check Out

- Checkout



- Checkout with invalid credit card



- User settings

BookStore

Cart Find Books **Settings** Managing

First Name user Last Name user

User Name user

E-mail user@user.com

Phone Number 01238554632

Shipping Address user home

Change Password

Demand Manager Privileges Save Changes

- 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:

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

New Order:

40 Order

40

Confirm Edit a Book

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

Order

Enter Number Of Books

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

Order

Enter Number Of Books

Edit a Book

- Search on a book to edit

BookStore:Edit Book

Find a book

Book Name

Publisher

Author

science

ISBN

Choose year of publication

Price

ISBN : 1 title : Absalom, Absalom! publisher_name : Ahmed publication_year : 1636-02-14 price : 10 category : Science au

ISBN : 6 title : Alien Corn (play) publisher_name : Abdelfatah publication_year : 1707-08-29 price : 96 category : Science a

ISBN : 11 title : An Acceptable Time publisher_name : Arsany publication_year : 1787-03-08 price : 58 category : Science a

ISBN : 13 title : An Evil Cradling publisher_name : Abdo publication_year : 1822-08-23 price : 21 category : Science author

ISBN : 18 title : Beneath the Bleeding publisher_name : Kirolos publication_year : 1888-08-20 price : 56 category : Science

Edit the book

An Acceptable Timer

Choose year of publication

Category

Save

Author

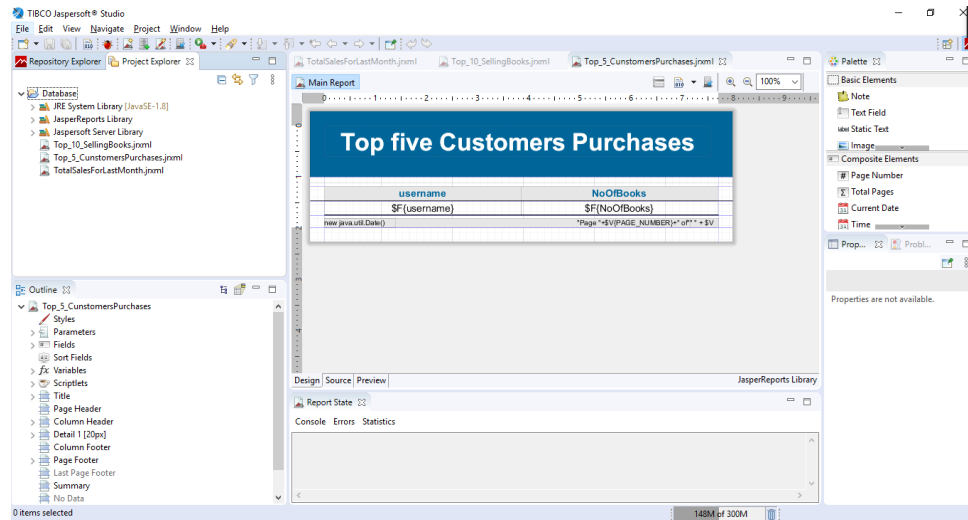
Price

Publisher

Cancel

* All fields should not be plank

○ Jasper using Jaspersoft studio:



- Top 10 selling books:

The screenshot displays a Java IDE environment. The top window, 'Dataset and Query Dialog', shows a SQL query for the 'bookStore' dataset. The query is as follows:

```

1 select ISBN, SUM(NoOfBooks) as NoOfBooks
2 from cart
3 where date >= last_day(now()) + interval 1 day - interval 3 month
4 group by ISBN
5 order by NoOfBooks desc
6 limit 10;

```

Below the query editor, a table lists the fields and their properties:

Field Name	Class Type	Description	Time zone	Column name	Column label	Column index	Properties
ISBN	java.lang.Int...						2 Properties
NoOfBooks	java.lang.Int...						2 Properties

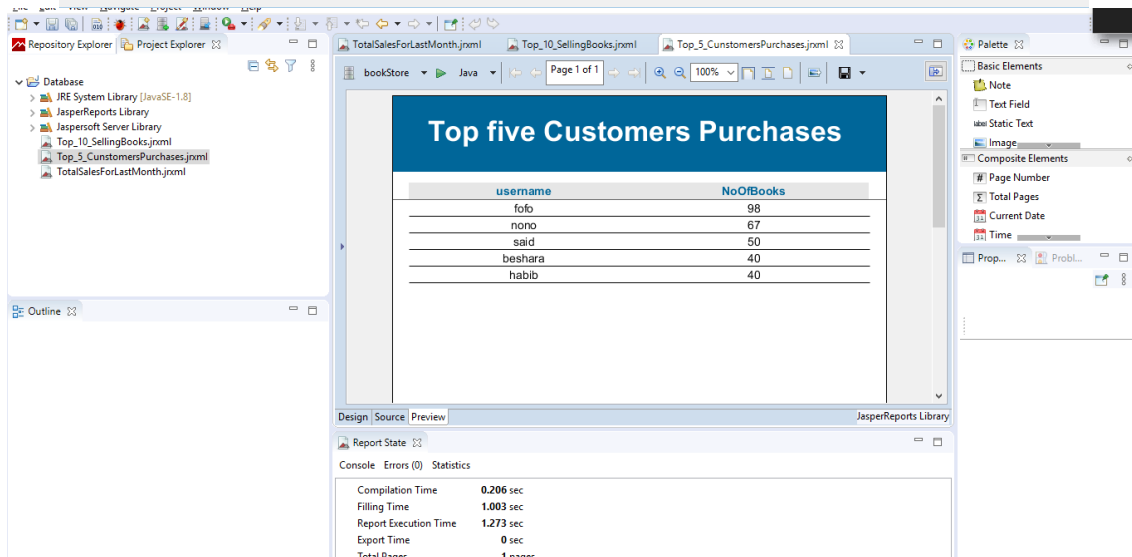
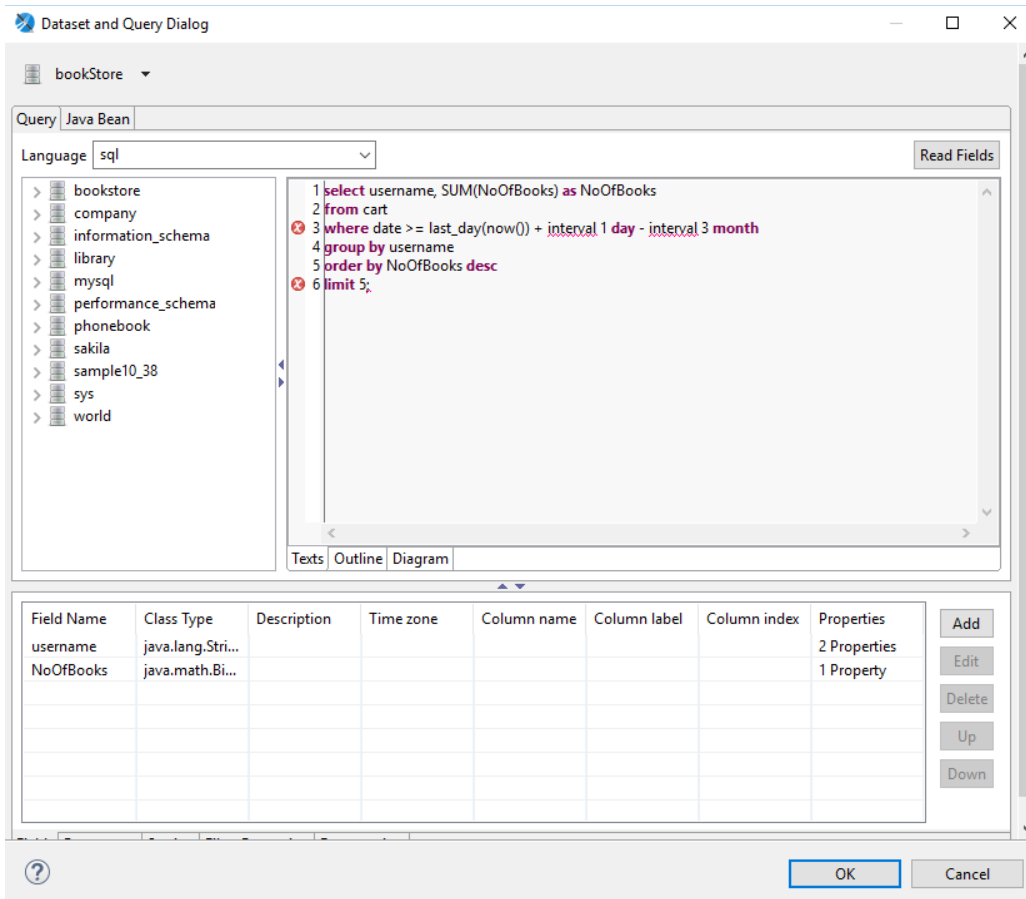
The bottom window shows the 'Top Ten Selling Books' report in the 'Preview' tab. The report has a blue header with the title 'Top Ten Selling Books'. The data is presented in a table with two columns: 'ISBN' and 'NoOfBooks'.

ISBN	NoOfBooks
5	255
2	50
3	40
4	30

At the bottom of the IDE, the 'Report State' and 'Statistics' panel shows the following performance metrics:

Metric	Value
Compilation Time	0.584 sec
Filling Time	1.321 sec
Report Execution Time	2.077 sec
Export Time	0 sec
Total Pages	1 pages

- Top 5 customers purchase:



- Total sales for last month

