WolfPubDb

for the *WolfCity* publishing house

CSC540 Database Management Systems
Project Report 3

Vamsi Varada, Aswin Itha, Mohan Kumar, Ishwari Zare
April 14 , 2022

Assumptions about the Project:

- To create a publication, we need an Editor associated with it.
- Each article is published only in one issue.
- All articles belong to an issue.
- Each chapter belongs to exactly one book.
- Shipping cost is borne by WolfPublications.
- We are not keeping track of payment for individual orders but the total outstanding balance. But this can be calculated using the order transaction details.
- Periodicity is defined only for an issue.
- Every author/editor is always working on some or the other book/issue at any given time.
- Authors are only for Books and Articles and not for Individual chapters.
- Table of contents and text of articles are a single attribute.
- Every staff author/editor gets a fixed salary and invited author/editors are paid differently.
- We will never be allowed to delete a Publication.
- Users would be allowed to delete a book/chapter/issue/article.
- Distributor can be deleted if and only if he/she has no transactions are recorded earlier.

Cass Name : com.ncsu.wolfpub.dao.TransactionsDAO class
1. Method Name : createPaymentForShippingCost()
2. Method Name : createPaymentByDistributor()
3 Method Name: createPaymentToStaff()

Class Name: com.ncsu.wolfpub.dao.DistributorDAO class
4. Method Name: placeOrderAndUpdateDistributor

Transaction 1 : Enter Payment Details of Shipping

```
public void createPaymentForShippingCost(int tid, Date transactionDate, float amount, String debitCredit,
                String paymentMode, String transactionType, int orderId) {

        Connection connection = client.getConnection();

        try {
                connection.setAutoCommit(false);
                PreparedStatement statement1 = connection.prepareStatement(INSERT_PAYMENT);

                statement1.setInt(1, tid);
                statement1.setDate(2, new java.sql.Date(transactionDate.getTime()));
                statement1.setFloat(3, amount);
                statement1.setString(4, debitCredit);
                statement1.setString(5, paymentMode);
                statement1.setString(6, transactionType);

                PreparedStatement statement2 = connection.prepareStatement(INSERT_PAYMENT_FOR_ORDER);
                statement2.setInt(1, tid);
                statement2.setInt(2, orderId);

                statement1.executeUpdate();
                statement2.executeUpdate();
                connection.commit();

        } catch (SQLException e) {
                System.out.println("Operation Failed. Please enter valid details");
                if (connection != null) {
                        try {
                                connection.rollback();
                        } catch (Exception nestedException) {
                                //Nothing to do
                        }

                }
        } finally {
                try {
                        connection.setAutoCommit(true);
                } catch (SQLException e) {
                        //Nothing to do
                }
        }
}
```

Transaction 2 : Enter Payment made by Distributor

```
public void createPaymentByDistributor(int tid, Date transactionDate, float amount, String debitCredit,
                String paymentMode, String transactionType, int dID) {
        DistributorDAO dDao = new WolfDAOFactory().getDistributorDAO();
        Connection connection = client.getConnection();

        try {
                connection.setAutoCommit(false);
                PreparedStatement statement1 = connection.prepareStatement(INSERT_PAYMENT);

                statement1.setInt(1, tid);
                statement1.setDate(2, new java.sql.Date(transactionDate.getTime()));
                statement1.setFloat(3, amount);
                statement1.setString(4, debitCredit);
                statement1.setString(5, paymentMode);
                statement1.setString(6, transactionType);

                PreparedStatement statement2 = connection.prepareStatement(INSERT_CREDIT_BY_DISTRIBUTOR);
                statement2.setInt(1, tid);
                statement2.setInt(2, dID);

                float balance = dDao.getDistributorOutStandingBalance(dID);
                float outstanding = balance - amount;

                PreparedStatement statement3 = connection.prepareStatement(UPDATE_OUTSTANDING_BALANCE);
                statement3.setFloat(1, outstanding);
                statement3.setInt(2, dID);

                statement1.executeUpdate();
                statement2.executeUpdate();
                statement3.executeUpdate();
                connection.commit();

        } catch (SQLException e) {
                System.out.println("Operation Failed. Please enter valid details");
                if (connection != null) {
                        try {
                                connection.rollback();
                        } catch (Exception nestedException) {
                                //Nothing to do
                        }

                }
        } finally {
                try {
                        connection.setAutoCommit(true);
                } catch (SQLException e) {
                        //Nothing to do
                }
        }

}
```

Transaction 3 : Record the Payment Made to Staff Member

```
public void createPaymentToStaff(int tid, Date transactionDate, float amount, String debitCredit,
                String paymentMode, String transactionType, int ID, String workType) {

        IDBConnection client = DBConnectionClient.getDBClient();
        Connection connection = client.getConnection();

        try {
                connection.setAutoCommit(false);
                PreparedStatement statement1 = connection.prepareStatement(INSERT_PAYMENT);

                statement1.setInt(1, tid);
                statement1.setDate(2, new java.sql.Date(transactionDate.getTime()));
                statement1.setFloat(3, amount);
                statement1.setString(4, debitCredit);
                statement1.setString(5, paymentMode);
                statement1.setString(6, transactionType);

                PreparedStatement statement2 = connection.prepareStatement(INSERT_STAFF_PAYMENT);
                statement2.setInt(1, tid);
                statement2.setInt(2, ID);
                statement2.setString(3, workType);

                statement1.executeUpdate();
                statement2.executeUpdate();
                connection.commit();

        } catch (SQLException e) {
                System.out.println("Operation Failed. Please enter valid details");
                if (connection != null) {
                        try {
                                connection.rollback();
                        } catch (Exception nestedException) {
                                //Nothing to do
                        }

                }
        } finally {
                try {
                        connection.setAutoCommit(true);
                } catch (SQLException e) {
                        //Nothing to do
                }
        }
}
```

Transaction 4 : Place an Order

```
public void placeOrderAndUpdateDistributor(int orderId, int did, Date orderDate, Date deliveryDate,
                float shippingCost, List<int[]> items) {
        Connection connection = client.getConnection();
        try {
                connection.setAutoCommit(false);

                PreparedStatement statement1 = connection.prepareStatement(CREATE_ORDER);
                statement1.setInt(1, orderId);
                statement1.setInt(2, did);
                statement1.setDate(3, new java.sql.Date(orderDate.getTime()));
                statement1.setDate(5, new java.sql.Date(deliveryDate.getTime()));
                statement1.setFloat(6, shippingCost);
                float orderCost = 0;

                PreparedStatement statement2 = connection.prepareStatement(ADD_ITEMS_TO_ORDER);
                for(int[] item : items) {
                        statement2.setInt(1, item[0]);
                        statement2.setInt(2, orderId);
                        statement2.setInt(3, item[1]);
                        float cost = costOfPID(item[0]);
                        statement2.setFloat(4, cost);
                        orderCost += (item[1] * cost);
                        statement2.addBatch();
                }

                float distributorBalance = getDistributorOutStandingBalance(did);

                PreparedStatement statement3 = connection.prepareStatement(UPDATE_OUTSTANDING_BALANCE);
                float outStandingBalance = distributorBalance+orderCost;
                statement3.setFloat(1, outStandingBalance);
                statement3.setInt(2, did);

                statement1.setFloat(4, orderCost);

                statement1.executeUpdate();
                statement2.executeBatch();
                statement3.executeUpdate();
                connection.commit();

        } catch (SQLException e) {
                System.out.println("Operation Failed. Please enter valid details " + e.getMessage());
                if(connection != null) {
                        try{
                                connection.rollback();
                        } catch(Exception nestedException) {
                                //Nothing to do
                        }

                }
        } finally {
                try {
                        connection.setAutoCommit(true);
                } catch (SQLException e) {
                                //Nothing to do
                }
        }
}
```

Design Decisions:

The system has a main menu that gives user the following options:

1. Editing and publishing
2. Production of a book edition or of an issue of a publication
3. Distribution
4. Reports
0. Exit

When prompted, the user enters a number from 0-4 corresponding to the task to be executed. To end the process, the user can select 0. For each menu displayed, subtasks are defined that have specific operations listed.If a user enters an invalid option , the control returns to the main menu.
WolfPubDb has two main classes.

1. **Create Contents:**
    - It reads schema from wolf_schema.txt and data from wolf_data.txt
    - The read data is populated into the WolfPub database.
2. **App**
    - Runs the application and provides the options for the user in the main menu.
    - Once the user selects his choice, the control flows to the respective processor class.
    - All Database calls are handled by the DAO layer.

    - We have a DAO Factory to avoid creating multiple instances of DAO
    - Whenever a database call is required, DAO factory will give its respective DAO class

    - An interface called IDBConnection is created. When an application has to support any other database, we need to implement this interface to connect to the new kind of database.
    - Currently MariaDBConnection is the only class implementing IDBConnection, as MariaDB is the only DB supported now.
    - The ConnectionClient will be provided by DBConnectionClient.

    - A Single connection to the database will be initialized and we are using the single connection throughout the entire application.
    - This is done to improve the application performance by avoiding the overhead of establishing connections every time.