

# CS 166 Project Report

## Group Information

Group 46

Girum yaye—gyaye001

Zaniah Lewis—zlewi004

## Implementation Description

Our goal was to add functionality and an overall better user interface by adding functionality to the GameRental class to manage user profiles, view game catalogs, place orders, and view orders in a PostgreSQL database using SQL queries executed through JDBC. Thus, handling input and output operations while interacting with the database to fetch and update data as required without any malfunctions. All together we have thirteen queries. This updateprofile query updates the user's phone number, password or favorite games based on their login. The viewrecentorders query allows the user to retrieve a list of a certain number of recent orders from the RentalOrder table. The viewcatalog query retrieves and displays all games in the catalog. The placeorder query inserts a new order into the RentalOrder table. The viewAllOrders query retrieves and displays all orders from the RentalOrder table in the database. The createuser query inserts a new user into the Users table. The login query checks if the login credentials entered exist in the Users table. The viewprofile query retrieves and displays the users profile. The vieworderinfo query retrieves and displays the details of a specific rental order. The viewtrackinginfo query retrieves and displays tracking information for a specific rental order from the rental table. The updateTrackingInfo query updates tracking information for a specific rental order from the rental table. The updateCatalog query updates the details of a game in the catalog. The updateUser query updates the role and number of overdue games for a user.

```
public static void showCatalog(GameRental eqsl) {
    try {
        System.out.println("Search catalog by:");
        System.out.println("1. Genre");
        System.out.println("2. Price");
        System.out.println("3. Sort by Price (High to Low)");
        System.out.println("4. Sort by Price (Low to High)");
        int choice = readChoice();
        String query = "";

        switch(choice) {
            case 1:
                System.out.println("Enter genre:");
                String genre = in.readLine();
                query = String.format("SELECT * FROM Catalog WHERE genre = '%s'", genre);
                break;
            case 2:
                System.out.println("Enter max/min price:");
                String price = in.readLine();
                query = String.format("SELECT * FROM Catalog WHERE price = '%s'", price);
                break;
            case 3:
                query = "SELECT * FROM Catalog ORDER BY price DESC";
                break;
            case 4:
                query = "SELECT * FROM Catalog ORDER BY price ASC";
                break;
            default:
                System.out.println("Invalid choice!");
                return;
        }
        eqsl.executeQuery(query);
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}

public static void placeOrder(GameRental eqsl, String username) {
    try {
        System.out.println("Enter game ID:");
        String gameId = in.readLine();
        System.out.println("Enter number of units:");
        int unitsOrdered = Integer.parseInt(in.readLine());

        // Generate unique rentalOrderID
        String rentalOrderID = "" + System.currentTimeMillis();

        // Calculate total price
        String query = String.format("SELECT price FROM Catalog WHERE gameId = '%s'", gameId);
        ResultSet rs = eqsl.executeQuery(query);
        double totalPrice = unitsOrdered * rs.getDouble(1);
        double totalPrice = unitsOrdered * rs.getDouble(1);
        // Set current timestamp and due date (7 days later)
        String orderTimestamp = "now()";
        String dueDate = "now() + interval '7 days'";

        // Insert into RentalOrder
        String insertRentalOrder = String.format("INSERT INTO RentalOrder (rentalOrderID, login, userID, totalPrice, orderTimestamp, dueDate) VALUES ('%s', '%s', '%s', '%s', '%s', '%s')", rentalOrderID, username, unitsOrdered, totalPrice, orderTimestamp, dueDate);
        eqsl.executeUpdate(insertRentalOrder);

        // Insert into GameRental
        String insertGameRental = String.format("INSERT INTO GameRental (rentalOrderID, gameId, unitsOrdered) VALUES ('%s', '%s', '%s', '%s', '%s', '%s')", rentalOrderID, gameId, unitsOrdered);
        eqsl.executeUpdate(insertGameRental);

        // Generate unique trackingID
        System.out.println("Enter unique tracking ID:");
        String trackingID = in.readLine();

        // Insert into TrackingInfo
        String insertTrackingInfo = String.format("INSERT INTO TrackingInfo (trackingID, rentalOrderID, status, currentLocation, destination, lastUpdate) VALUES ('%s', '%s', 'Pending', '%s', '%s', '%s')", trackingID, rentalOrderID, orderTimestamp);
        eqsl.executeUpdate(insertTrackingInfo);

        System.out.println("Order successfully placed!");
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}

public static void createuser(GameRental eqsl) {
    try {
        System.out.println("Enter username:");
        String username = in.readLine();
        System.out.println("Enter Password:");
        String password = in.readLine();
        System.out.println("Enter Phone Number:");
        String phoneNumber = in.readLine();

        String query = String.format("INSERT INTO Users(login, password, role, phoneNumber, numberOfOverdue) VALUES ('%s', '%s', 'customer', '%s', 0)", username, password, phoneNumber);
        System.out.println(query);
        eqsl.executeUpdate(query);
        System.out.println("User successfully created!");
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}

// End of createuser

// Check login in credentials for an existing user
// Return user login or null if the user does not exist
// Null

public static String Login(GameRental eqsl) {
    try {
        System.out.println("Enter username:");
        String username = in.readLine();
        System.out.println("Enter password:");
        String password = in.readLine();

        String query = String.format("SELECT * FROM Users WHERE login = '%s' AND password = '%s'", username, password);
        int userNum = eqsl.executeQuery(query);

        if (userNum > 0) {
            System.out.println("Successful Log in!");
            return username;
        } else {
            System.out.println("Invalid username or password.");
            return null;
        }
    } catch (Exception e) {
        System.out.println(e.getMessage());
        return null;
    }
}

// End of Login

// Next of the functions definition go in here

public static void viewProfile(GameRental eqsl, String username) {
    try {
        String query = String.format("SELECT * FROM Users WHERE login = '%s'", username);
        ResultSet rs = eqsl.executeQuery(query);
        catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }

    public static void updateProfile(GameRental eqsl, String username) {
        try {
            System.out.println("What would you like to update?");
            System.out.println("1. Password");
            System.out.println("2. Phone Number");
            System.out.println("3. Favorite Game");
            int choice = readChoice();
            String query = "";

            switch(choice) {
                case 1:
                    System.out.println("Enter new Password:");
                    String newPassword = in.readLine();
                    query = String.format("UPDATE Users SET password = '%s' WHERE login = '%s'", newPassword, username);
                    break;
                case 2:
                    System.out.println("Enter new Phone Number:");
                    String newPhoneNumber = in.readLine();
                    query = String.format("UPDATE Users SET phoneNumber = '%s' WHERE login = '%s'", newPhoneNumber, username);
                    break;
                case 3:
                    System.out.println("Enter new Favorite Game:");
                    String newFavoriteGame = in.readLine();
                    query = String.format("UPDATE Users SET favoriteGame = '%s' WHERE login = '%s'", newFavoriteGame, username);
                    break;
            }
            eqsl.executeUpdate(query);
            System.out.println("Profile successfully updated!");
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

```

    }

    public static void viewOrderInfo(GameHental esql, String username) {
        try {
            System.out.println("Enter Rental Order ID: ");
            String rentalOrderID = in.readLine();
            String query = String.format("SELECT * FROM GamesInOrder WHERE rentalOrderID = '%s'", rentalOrderID, username);
            esql.executeQueryAndPrintResult(query);
        } catch (Exception e) {
            System.err.println(e.getMessage());
        }
    }

    public static void viewTrackingInfo(GameHental esql, String username) {
        try {
            System.out.println("Enter Tracking ID: ");
            String trackingID = in.readLine();
            String query = String.format("SELECT * FROM TrackingInfo WHERE trackingID = '%s'", trackingID, username);
            esql.executeQueryAndPrintResult(query);
        } catch (Exception e) {
            System.err.println(e.getMessage());
        }
    }

    public static void updateTrackingInfo(GameHental esql) {
        try {
            System.out.println("Enter Tracking ID: ");
            String trackingID = in.readLine();
            System.out.println("Enter new status: ");
            String status = in.readLine();
            System.out.println("Enter new courier name: ");
            String courierName = in.readLine();
            System.out.println("Enter additional comments: ");
            String additionalComments = in.readLine();

            String query = String.format("UPDATE TrackingInfo SET status = '%s', courierName = '%s', additionalComments = '%s' WHERE trackingID = '%s'", status, courierName, additionalComments, trackingID);
            esql.executeUpdate(query);
            System.out.println("Tracking information updated successfully");
        } catch (Exception e) {
            System.err.println(e.getMessage());
        }
    }

    public static void updateCatalog(GameHental esql) {
        try {
            System.out.println("Enter game ID: ");
            String gameID = in.readLine();
            System.out.println("Enter new price: ");
            double price = Double.parseDouble(in.readLine());
            System.out.println("Enter description: ");
            String descrip = in.readLine();

            String query = String.format("UPDATE Catalog SET description = '%s', price = '%s' WHERE gameID = '%s'", descrip, price, gameID);
            esql.executeUpdate(query);
            System.out.println("Catalog updated successfully");
        } catch (Exception e) {
            System.err.println(e.getMessage());
        }
    }

    public static void updateUser(GameHental esql) {
        try {
            System.out.println("Enter username: ");
            String username = in.readLine();
            System.out.println("Enter new role: ");
            String role = in.readLine();
            System.out.println("Enter new number of Overdue Games: ");
            int numOverdueGames = Integer.parseInt(in.readLine());

            String query = String.format("UPDATE Users SET role = '%s', numOverdueGames = %d WHERE login = '%s'", role, numOverdueGames, username);
            esql.executeUpdate(query);
            System.out.println("User information has been updated");
        } catch (Exception e) {
            System.err.println(e.getMessage());
        }
    }
}

//end GameHental

```

## Problems/Findings

One of the difficulties we had was with making sure we were currently writing the sql queries for the JDBC drivers in Java. We also had to take the time to properly understand how all of the imported methods work interactively with our queries and the database, as well as what's input and output for each. One problem that I did have was actually with vscode in itself because it wasn't allowing my text files and sql files which needed to be added to execute our code. In the end, we simply shut down our database and re-copied our files to the server. Another challenge was handling user input and output smoothly while interacting with the database, which was what we used the bufferedreader method and the catch exceptions for.

## Contributions

Girum focused on designing the database schema and creating the necessary tables, while Zaniah wrote the SQL scripts and ensured that relationships and constraints were

properly implemented. Zaniah worked on the implementation of the Java methods to interact with the database, which involved executing the SQL queries. Girum made sure our code and queries were tested for functionality with good results. Our collaborative efforts ensured the overall performance, and user experience for phase 3 of our project was efficient.