Kaylee Moore and Jack Lindner

Game Rental Database

CS 461, Database Systems Final Project
Professor Sutton

This project was designed to help students learn how to create a database management system from scratch. Given a project PDF, we set out to create an in-depth database setup to demonstrate our skills in database management systems (DBMS).

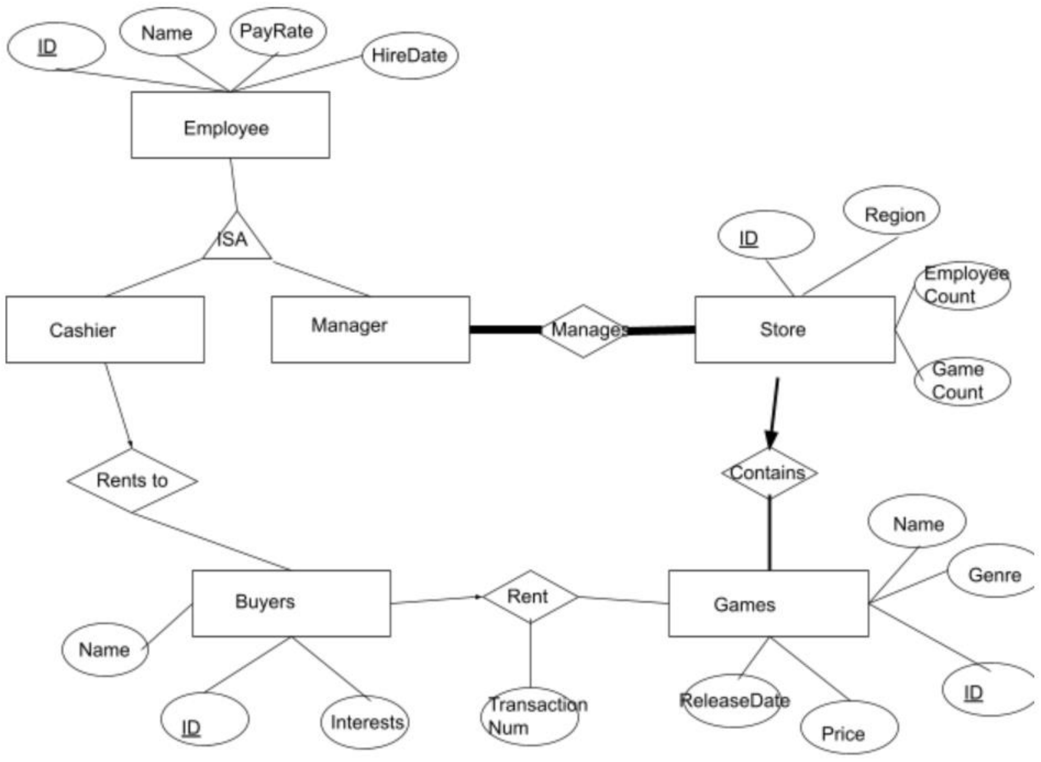
Our goal with this project was to provide a system that manages the inventory and rental of video games in a rental store network of franchises. In this franchise system, there are many stores, each in a different region carrying different stock and having different employees and customers.

One of the main technical problems we faced when designing this database system was regarding the editing of the database itself. Adding objects to the tables in the database required a bit of creative work to form the custom SQL statements embedded in the code.

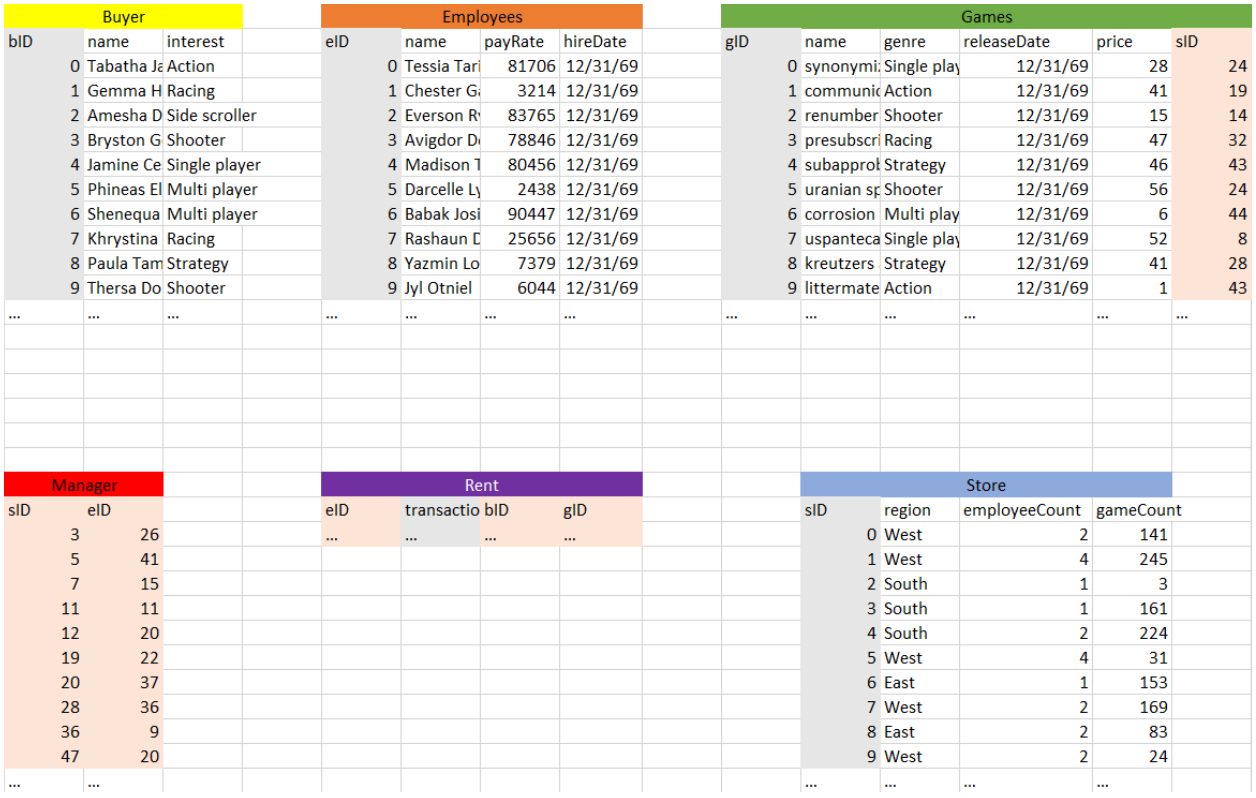
Some of the main requirements of this database were: buyers register with their name and interests, buyers can view game information, buyers can search for games at either their region's store or different store, cashiers and managers can check out buyers, buyer can see cashiers name but no other information, everyone can see if game is rented out currently, cashiers and managers can view all store information, buyer can only view store region and game count, managers can edit inventory manually, and cashiers can but all game information but not edit the game information.

In order to mitigate redundancies, we decided to break up all the items as we built the schema. We did this by choosing not to make a single entry for each game and have a bunch of attributes, making everything self-contained instead. This eliminates overlap and redundancies between the tables which leaves less room for errors and unexpected outputs.

When designing the GUI, we went for a very simple and easy-to-follow design. With five buttons along the top and a dropdown selection, you can specify which table you would like to view and then act upon that data. Some actions available currently are add, delete, and modify entries, and move between pages using “previous” or “next” buttons.



We decided to break up the Entity-Relationship diagram in the fashion above so that we would reduce functional dependencies and possible weak points in our database.



Buyer		
bID	name	interest
0	Tabatha Ja	Action
1	Gemma H	Racing
2	Amesha D	Side scroller
3	Bryston G	Shooter
4	Jamine Ce	Single player
5	Phineas El	Multi player
6	Shenequa	Multi player
7	Khrystina	Racing
8	Paula Tam	Strategy
9	Thersa Do	Shooter
...	...	...

Employees			
eID	name	payRate	hireDate
0	Tessia Tari	81706	12/31/69
1	Chester G	3214	12/31/69
2	Everson R	83765	12/31/69
3	Avigdor D	78846	12/31/69
4	Madison T	80456	12/31/69
5	Darcelle L	2438	12/31/69
6	Babak Josi	90447	12/31/69
7	Rashaun D	25656	12/31/69
8	Yazmin Lo	7379	12/31/69
9	Jyl Otniel	6044	12/31/69
...	...	...	...

Games					
gID	name	genre	releaseDate	price	sID
0	synonymi	Single play	12/31/69	28	24
1	communic	Action	12/31/69	41	19
2	renumber	Shooter	12/31/69	15	14
3	presubscri	Racing	12/31/69	47	32
4	subapprol	Strategy	12/31/69	46	43
5	uranian sp	Shooter	12/31/69	56	24
6	corrosion	Multi play	12/31/69	6	44
7	uspanteca	Single play	12/31/69	52	8
8	kreutzers	Strategy	12/31/69	41	28
9	littermate	Action	12/31/69	1	43
...	...	...	...	...	...

Manager	
sID	eID
3	26
5	41
7	15
11	11
12	20
19	22
20	37
28	36
36	9
47	20
...	...

Rent			
eID	transactio	bID	gID
...	...	...	...

Store			
sID	region	employeeCount	gameCount
0	West	2	141
1	West	4	245
2	South	1	3
3	South	1	161
4	South	2	224
5	West	4	31
6	East	1	153
7	West	2	169
8	East	2	83
9	West	2	24
...	...	...	...

As discussed earlier in the report, we designed our database with redundancies and dependencies in mind. In the image above, gray columns are primary keys and the beige columns are foreign keys. bID from Buyer, eID from Employees, gID from Games, transactionNum from Rent, and sID from Store are all unique keys that are never duplicated. This improves our database setup because we will not need to create a different unique key for each table. Since we built the database with redundancies and dependencies in mind, BCNF did not have an effect on our database.

To generate the random data in the database, we used two sets of Internet-provided lists and one list generated by Jack Lindner. The first list, genres.txt, was created by Jack and has a bunch of random game genres that he could think of off the top of his head. The two others, names.txt and words_alpha.txt, are both pulled of the Internet. We then used those three lists and the Java random number generator to randomly pick lines from each of these files to generate different things like renter names, game names, and game genres. To generate the pay rate of employees, we used the Math.random function to generate a number from 0 to 100,000. To generate the date for employee hiring and game release dates, we generated a random integer for each part of the date: 1 through 12 for month, 1 through 28 for day, and 1 through 2019 for the year and then assembled this into a string. You can generate new random data **ONLY IF THERE ENTIRE DATABASE IS EMPTY** by running the program with '1' as an argument (No quotes though).

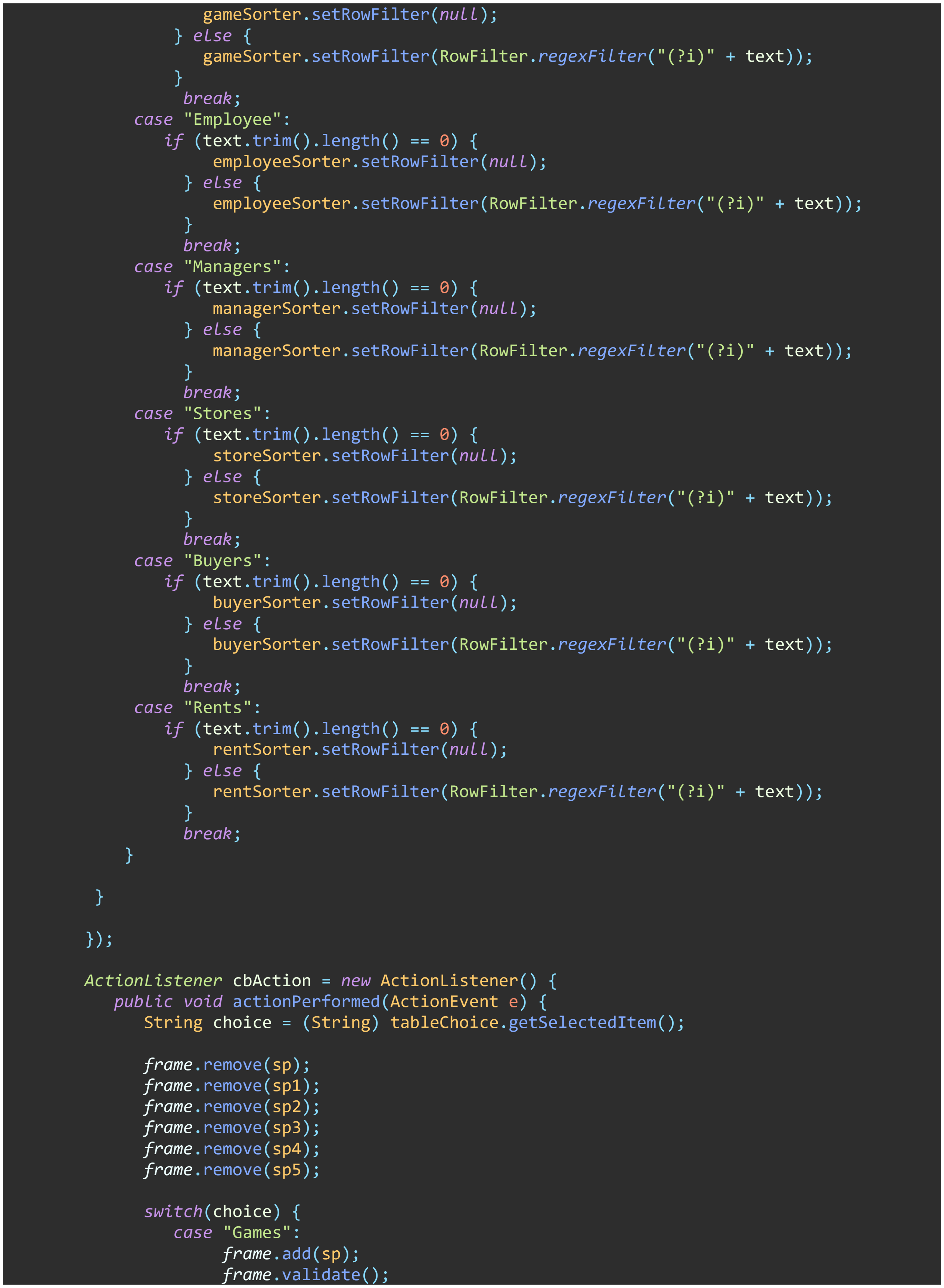
The user-interface for this database is laid out in a straightforward way. The 'New Transaction' button will allow you to input the information for a new transaction. The given dropdown prompts will show the available options for keys, as all information needs to be compatible with other tables. The 'Delete' button will display a popup box depending on the currently displayed table asking for the primary key of the item you are looking to delete. Entering the correct information will delete that entry in the table. The 'Update' button will display a series of popups, which will gather the information for a new entry to the currently displayed table. The 'Next' and 'Previous' buttons will cycle between the available tables. The dropdown box will allow you to pick a table directly, without needing to cycle through. The search textbox will filter the current displayed table, and only show the entries that match the entered information. You will need to press 'Enter' after your search query. Below the buttons, the currently displayed table will show all entries for the database table in question.

DatabaseUI.java -

```java
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableRowSorter;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.*;
import java.text.SimpleDateFormat;
import java.util.ArrayList;


public class DatabaseUI
{
   public static Connection conn;
   public static Statement stmt = null;
   public static JFrame frame = new JFrame("Game Rental Database");
    private static String sql;
    private static GenerateRandom generateRandom = new GenerateRandom();
    public static SimpleDateFormat sdf = new SimpleDateFormat("MM-DD-yyyy");

    public static void main(String[] args) throws SQLException
    {
        // Creating the Frame
       //JFrame frame = new JFrame("Game Rental Database");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(625, 550);

        connectDB();

        Statement stmt = conn.createStatement();

        // Add default values if program is ran with '1' as argument
        if(args.length != 0 && args[0].equals("1"))
        {
            addBuyers(100);
            addStore(50);
            addEmployees(99);
            addGames(250);
            addManager(20);
        }
        else
        {
            System.out.println("Database may be empty. Run program with '1' as argument to
generate values.");
        }

        // Adding buttons
        JButton newTransButton = new JButton("New Transaction");
        newTransButton.setBounds(0,50,50,50);
        JButton delButton = new JButton("Delete");
        delButton.setBounds(0,50,100,50);
        JButton updateButton = new JButton("Update Database");
        updateButton.setBounds(0,50,100,50);
        JButton nextButton = new JButton("Next");
        nextButton.setBounds(0,50,100,50);
        JButton prevButton = new JButton("Previous");
        prevButton.setBounds(0,50,100,50);
        frame.add(newTransButton);
        frame.add(delButton);
        frame.add(updateButton);
        frame.add(nextButton);
        frame.add(prevButton);

        // Setting flow style
        frame.setLayout(new FlowLayout(FlowLayout.LEFT));


        //Init Game Table --------------------------------------------------------------------
---
        sql = "SELECT * FROM 'Games';";

        ResultSet rs = stmt.executeQuery(sql);

        int gColAm = rs.getMetaData().getColumnCount();

        String[] gameColumn = {"ID Number", "Name", "Genre", "Release Date", "Price", "Store
ID"};

        DefaultTableModel gameTableModel = new DefaultTableModel(gameColumn, 0);

        JTable gameTable= new JTable(gameTableModel);

        while(rs.next()) {
           Object[] gameRow = new Object[gColAm];

           gameRow[0] = rs.getInt("gID");
           gameRow[1] = rs.getString("name");
           gameRow[2] = rs.getString("genre");
           gameRow[3] = rs.getString("releaseDate");
           gameRow[4] = rs.getDouble("price");
           gameRow[5] = rs.getInt("sID");

           gameTableModel.addRow(gameRow);
        }

        gameTable.setBounds(0,40,600,300);
        gameTable.setRowSelectionInterval(0,0);
        JScrollPane sp = new JScrollPane(gameTable);
        //Init Buyers Table ------------------------------------------------------------------
-----
        sql = "SELECT * FROM 'Buyers'";

        rs = stmt.executeQuery(sql);

        int bColAm = rs.getMetaData().getColumnCount();

        String[] buyerColumn = {"ID Number", "Name", "Interest"};

        DefaultTableModel buyerTableModel = new DefaultTableModel(buyerColumn, 0);

        JTable buyerTable= new JTable(buyerTableModel);

        while(rs.next()) {
           Object[] buyerRow = new Object[bColAm];

           buyerRow[0] = rs.getInt("bID");
           buyerRow[1] = rs.getString("name");
           buyerRow[2] = rs.getString("interest");

           buyerTableModel.addRow(buyerRow);

        }

        buyerTable.setBounds(0,40,400,300);
        buyerTable.setRowSelectionInterval(0,0);
        JScrollPane sp4 = new JScrollPane(buyerTable);

        //Init Employee Table ----------------------------------------------------------------
-------
        sql = "SELECT * FROM 'Employee';";

        rs = stmt.executeQuery(sql);

        int eColAm = rs.getMetaData().getColumnCount();

        String[] employeeColumn = {"ID Number", "Name", "Pay Rate", "Hire Date"};

        DefaultTableModel employeeTableModel = new DefaultTableModel(employeeColumn, 0);

        JTable employeeTable= new JTable(employeeTableModel);

        while(rs.next()) {
           Object[] employeeRow = new Object[eColAm];

           employeeRow[0] = rs.getInt("eID");
           employeeRow[1] = rs.getString("name");
           employeeRow[2] = rs.getInt("payRate");
           employeeRow[3] = rs.getDate("hireDate");

           employeeTableModel.addRow(employeeRow);

        }

        employeeTable.setBounds(0,40,400,300);
        employeeTable.setRowSelectionInterval(0,0);
        JScrollPane sp1 = new JScrollPane(employeeTable);

        //Init Manager Table -----------------------------------------------------------------
------
```

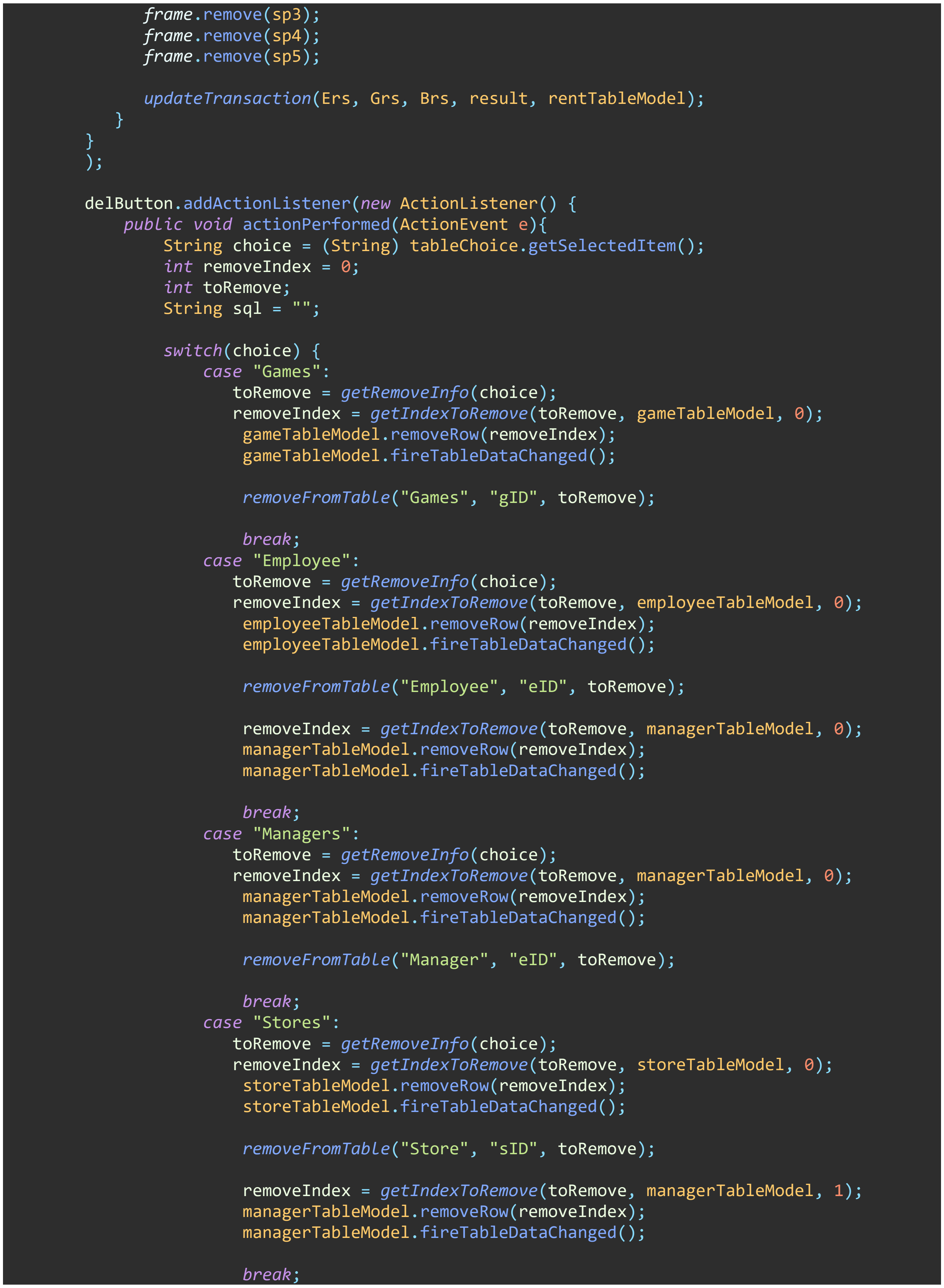
```
        sql = "SELECT * FROM 'Manager';";

        rs = stmt.executeQuery(sql);

        int mColAm = rs.getMetaData().getColumnCount();

        String[] managerColumn = {"Employee ID Number", "Store ID Number"};

        DefaultTableModel managerTableModel = new DefaultTableModel(managerColumn, 0);

        JTable managerTable = new JTable(managerTableModel);

        while(rs.next()) {
           Object[] managerRow = new Object[mColAm];

           managerRow[0] = rs.getInt("eID");
           managerRow[1] = rs.getInt("sID");

           managerTableModel.addRow(managerRow);
        }

        managerTable.setBounds(0,40,400,300);
        managerTable.setRowSelectionInterval(0,0);
        JScrollPane sp2 = new JScrollPane(managerTable);

        //Init Rent Table ------------------------------------------------------------
---
        sql = "SELECT * FROM 'Rent';";

        rs = stmt.executeQuery(sql);

        int rColAm = rs.getMetaData().getColumnCount();

        String[] rentColumn = {"Employee ID Number", "Transaction Number", "Buyer ID Number",
"Game ID Number"};

        DefaultTableModel rentTableModel = new DefaultTableModel(rentColumn, 0);

        JTable rentTable = new JTable(rentTableModel);

        while(rs.next()) {
           Object[] rentRow = new Object[rColAm];

           rentRow[0] = rs.getInt("eID");
           rentRow[1] = rs.getInt("transactionNum");
           rentRow[2] = rs.getInt("bID");
           rentRow[3] = rs.getInt("gID");

           rentTableModel.addRow(rentRow);
        }

        rentTable.setBounds(0,40,400,300);
        rentTable.setRowSelectionInterval(0,0);
        JScrollPane sp5 = new JScrollPane(rentTable);

        //Init Store Table -----------------------------------------------------------
----
        sql = "SELECT * FROM 'Store';";

        rs = stmt.executeQuery(sql);
```

```java
        int sColAm = rs.getMetaData().getColumnCount();

        String[] storeColumn = {"Store ID Number", "Region", "Employee Count", "Game Count"};

        DefaultTableModel storeTableModel = new DefaultTableModel(storeColumn, 0);

        JTable storeTable = new JTable(storeTableModel);

        while(rs.next()) {
           Object[] storeRow = new Object[sColAm];

           storeRow[0] = rs.getInt("sID");
           storeRow[1] = rs.getString("region");
           storeRow[2] = rs.getInt("employeeCount");
           storeRow[3] = rs.getInt("gameCount");

           storeTableModel.addRow(storeRow);
        }

        storeTable.setBounds(0,40,400,300);
        storeTable.setRowSelectionInterval(0,0);
        JScrollPane sp3 = new JScrollPane(storeTable);

        //adding a dropdown for the table menus

        String[] tableOptions = {"Games", "Employee", "Managers", "Stores", "Buyers",
"Rents"};
        JComboBox tableChoice = new JComboBox(tableOptions);

        frame.add(tableChoice);

        //add search function after the dropdown
        JLabel search = new JLabel("Search: ");
        JTextField searchField = new JTextField();
        searchField.setColumns(15);
        frame.add(search);
        frame.add(searchField);

        frame.add(sp);

        TableRowSorter gameSorter = new TableRowSorter(gameTable.getModel());
        gameTable.setRowSorter(gameSorter);
        TableRowSorter employeeSorter = new TableRowSorter(employeeTable.getModel());
        employeeTable.setRowSorter(employeeSorter);
        TableRowSorter managerSorter = new TableRowSorter(managerTable.getModel());
        managerTable.setRowSorter(managerSorter);
        TableRowSorter storeSorter = new TableRowSorter(storeTable.getModel());
        storeTable.setRowSorter(storeSorter);
        TableRowSorter buyerSorter = new TableRowSorter(buyerTable.getModel());
        buyerTable.setRowSorter(buyerSorter);
        TableRowSorter rentSorter = new TableRowSorter(rentTable.getModel());
        rentTable.setRowSorter(rentSorter);

        searchField.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            String text = searchField.getText();
            String choice = (String) tableChoice.getSelectedItem();

            switch(choice) {
            case "Games":
               if (text.trim().length() == 0) {
```



```
            gameSorter.setRowFilter(null);
         } else {
            gameSorter.setRowFilter(RowFilter.regexFilter("(?i)" + text));
         }
          break;
     case "Employee":
        if (text.trim().length() == 0) {
             employeeSorter.setRowFilter(null);
          } else {
             employeeSorter.setRowFilter(RowFilter.regexFilter("(?i)" + text));
          }
          break;
     case "Managers":
        if (text.trim().length() == 0) {
             managerSorter.setRowFilter(null);
          } else {
             managerSorter.setRowFilter(RowFilter.regexFilter("(?i)" + text));
          }
          break;
     case "Stores":
        if (text.trim().length() == 0) {
             storeSorter.setRowFilter(null);
          } else {
             storeSorter.setRowFilter(RowFilter.regexFilter("(?i)" + text));
          }
          break;
     case "Buyers":
        if (text.trim().length() == 0) {
             buyerSorter.setRowFilter(null);
          } else {
             buyerSorter.setRowFilter(RowFilter.regexFilter("(?i)" + text));
          }
          break;
     case "Rents":
        if (text.trim().length() == 0) {
             rentSorter.setRowFilter(null);
          } else {
             rentSorter.setRowFilter(RowFilter.regexFilter("(?i)" + text));
          }
          break;
    }

 }

});

ActionListener cbAction = new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      String choice = (String) tableChoice.getSelectedItem();

      frame.remove(sp);
      frame.remove(sp1);
      frame.remove(sp2);
      frame.remove(sp3);
      frame.remove(sp4);
      frame.remove(sp5);

      switch(choice) {
         case "Games":
              frame.add(sp);
              frame.validate();
```

```
                frame.setVisible(true);
                break;
           case "Employee":
                frame.add(sp1);

                frame.setVisible(true);
                break;
           case "Managers":
                frame.add(sp2);

                frame.setVisible(true);
                break;
           case "Stores":
                frame.add(sp3);

                frame.setVisible(true);
                break;
           case "Buyers":
                frame.add(sp4);

                frame.setVisible(true);
                break;
           case "Rents":
                frame.add(sp5);

                frame.setVisible(true);
                break;
        }
     }
  };

  tableChoice.addActionListener(cbAction);

  frame.setVisible(true);

  // Button Listeners

      // New Transaction Listener
  String sql1 = "SELECT eID FROM 'Employee';";
  stmt = conn.createStatement();
  final ResultSet Ers = stmt.executeQuery(sql1);

  String sql2 = "SELECT gID FROM 'Games';";
  stmt = conn.createStatement();
final ResultSet Grs = stmt.executeQuery(sql2);

String sql3 = "SELECT bID FROM 'Buyers';";
  stmt = conn.createStatement();
final ResultSet Brs = stmt.executeQuery(sql3);

String sql4 = "SELECT transactionNum FROM 'Rent';";
  stmt = conn.createStatement();
final ResultSet result = stmt.executeQuery(sql4);


  newTransButton.addActionListener(new ActionListener() {
     public void actionPerformed(ActionEvent e){
        frame.remove(sp);
        frame.remove(sp1);
        frame.remove(sp2);
```



```java
              frame.remove(sp3);
              frame.remove(sp4);
              frame.remove(sp5);

              updateTransaction(Ers, Grs, Brs, result, rentTableModel);
           }
        }
        );

        delButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e){
                String choice = (String) tableChoice.getSelectedItem();
                int removeIndex = 0;
                int toRemove;
                String sql = "";

                switch(choice) {
                    case "Games":
                       toRemove = getRemoveInfo(choice);
                       removeIndex = getIndexToRemove(toRemove, gameTableModel, 0);
                        gameTableModel.removeRow(removeIndex);
                        gameTableModel.fireTableDataChanged();

                        removeFromTable("Games", "gID", toRemove);

                        break;
                    case "Employee":
                       toRemove = getRemoveInfo(choice);
                       removeIndex = getIndexToRemove(toRemove, employeeTableModel, 0);
                        employeeTableModel.removeRow(removeIndex);
                        employeeTableModel.fireTableDataChanged();

                        removeFromTable("Employee", "eID", toRemove);

                        removeIndex = getIndexToRemove(toRemove, managerTableModel, 0);
                        managerTableModel.removeRow(removeIndex);
                        managerTableModel.fireTableDataChanged();

                        break;
                    case "Managers":
                       toRemove = getRemoveInfo(choice);
                       removeIndex = getIndexToRemove(toRemove, managerTableModel, 0);
                        managerTableModel.removeRow(removeIndex);
                        managerTableModel.fireTableDataChanged();

                        removeFromTable("Manager", "eID", toRemove);

                        break;
                    case "Stores":
                       toRemove = getRemoveInfo(choice);
                       removeIndex = getIndexToRemove(toRemove, storeTableModel, 0);
                        storeTableModel.removeRow(removeIndex);
                        storeTableModel.fireTableDataChanged();

                        removeFromTable("Store", "sID", toRemove);

                        removeIndex = getIndexToRemove(toRemove, managerTableModel, 1);
                        managerTableModel.removeRow(removeIndex);
                        managerTableModel.fireTableDataChanged();

                        break;
                    case "Buyers":
                       toRemove = getRemoveInfo(choice);
                       removeIndex = getIndexToRemove(toRemove, buyerTableModel, 0);
                        buyerTableModel.removeRow(removeIndex);
                        buyerTableModel.fireTableDataChanged();

                        removeFromTable("Buyers", "bID", toRemove);

                        break;
                    case "Rents":
                       toRemove = getRemoveInfo(choice);
                       removeIndex = getIndexToRemove(toRemove, rentTableModel, 1);
                        rentTableModel.removeRow(removeIndex);
                        rentTableModel.fireTableDataChanged();

                        removeFromTable("Rent", "transactionNum", toRemove);

                        break;
                }
            }
        });

        updateButton.addActionListener(new ActionListener() {
           public void actionPerformed(ActionEvent e) {
              String choice = (String)tableChoice.getSelectedItem();
              JFrame frame2 = new JFrame("Get Information");

              switch(choice) {
              case "Games":
                 Object[] gameRow = new Object[gColAm];

                 try
                    {
                        gameRow[0] = Integer.parseInt((String)
JOptionPane.showInputDialog(frame2, "Enter the Game ID Number: ", "Enter Info",
JOptionPane.PLAIN_MESSAGE, null, null, 0));
                        gameRow[1] = (String) JOptionPane.showInputDialog(frame2, "Enter the
Game Name: ", "Enter Info", JOptionPane.PLAIN_MESSAGE, null, null, 0);
                        gameRow[2] = (String) JOptionPane.showInputDialog(frame2, "Enter the
Game Genre: ", "Enter Info", JOptionPane.PLAIN_MESSAGE, null, null, 0);
                    }
                 catch(Exception ex)
                    {
                        System.out.println("ERROR WITH INPUT. TRY AGAIN.");
                    }
                   try {
                  gameRow[3] = (String) JOptionPane.showInputDialog(frame2, "Enter the Game
Release Date (MM-DD-yyyy): ", "Enter Info", JOptionPane.PLAIN_MESSAGE, null, null, 0);
               } catch (HeadlessException e2) {
                  e2.printStackTrace();
               }

                   try
                    {
                        gameRow[4] = Double.parseDouble((String)
JOptionPane.showInputDialog(frame2, "Enter the Game Price: ", "Enter Info",
JOptionPane.PLAIN_MESSAGE, null, null, 0));
                        gameRow[5] = Integer.parseInt((String)
JOptionPane.showInputDialog(frame2, "Enter the Store: ", "Enter Info",
JOptionPane.PLAIN_MESSAGE, null, null, 0));
                    }
                   catch(Exception ex2)
                    {
                        System.out.println("ERROR WITH INPUT. TRY AGAIN.");
                    }

                   gameTableModel.addRow(gameRow);
                   gameTableModel.fireTableDataChanged();

                   sql = "INSERT INTO 'Games' (gID, name, genre, releaseDate, price, sID)" +
                            "VALUES (" + gameRow[0] + ", '" + gameRow[1] + "', '" + gameRow[2]
+ "', '" + gameRow[3] + "', " + gameRow[4] + ", " + gameRow[5] + ");";
                   try {
                  Statement stmt1 = conn.createStatement();
                  stmt1.executeUpdate(sql);
               } catch (SQLException e1) {
                  e1.printStackTrace();
               }

                 break;
              case "Employee":
                 Object[] employeeRow = new Object[eColAm];

                 try
                    {
                        employeeRow[0] = Integer.parseInt((String)
JOptionPane.showInputDialog(frame2, "Enter the Employee ID Number: ", "Enter Info",
JOptionPane.PLAIN_MESSAGE, null, null, 0));
                        employeeRow[1] = (String) JOptionPane.showInputDialog(frame2, "Enter
the Employee Name: ", "Enter Info", JOptionPane.PLAIN_MESSAGE, null, null, 0);
                        employeeRow[2] = Integer.parseInt((String)
JOptionPane.showInputDialog(frame2, "Enter the Employee Pay Rate: ", "Enter Info",
JOptionPane.PLAIN_MESSAGE, null, null, 0));
                    }
                 catch(Exception ex3)
                    {
                        System.out.println("ERROR WITH INPUT. TRY AGAIN.");
                    }

                   try {
                  employeeRow[3] = (String) JOptionPane.showInputDialog(frame2, "Enter the
Employee Hire Date (mm.dd.yyyy): ", "Enter Info", JOptionPane.PLAIN_MESSAGE, null, null, 0);
               } catch (HeadlessException e2) {
                  e2.printStackTrace();
               }

                   employeeTableModel.addRow(employeeRow);
                   employeeTableModel.fireTableDataChanged();

                   sql = "INSERT INTO 'Employee' (eID, name, payRate, hireDate) " +
                            "VALUES (" + employeeRow[0] + ", '" + employeeRow[1] + "', " +
employeeRow[2] + ", '" + employeeRow[3] + "');";
                   try {
                      Statement stmt1 = conn.createStatement();
                      stmt1.executeUpdate(sql);
                   }catch(SQLException e1) {
                      e1.printStackTrace();
                   }

                 break;
              case "Managers":
                 Object[] managerRow = new Object[mColAm];
```

```
                  try
                     {
                         managerRow[0] = Integer.parseInt((String)
JOptionPane.showInputDialog(frame2, "Enter the Employee ID Number: ", "Enter Info",
JOptionPane.PLAIN_MESSAGE, null, null, 0));
                         managerRow[1] = Integer.parseInt((String)
JOptionPane.showInputDialog(frame2, "Enter the Store ID Number: ", "Enter Info",
JOptionPane.PLAIN_MESSAGE, null, null, 0));
                     }
                  catch(Exception ex4)
                     {
                         System.out.println("ERROR WITH INPUT. TRY AGAIN.");
                     }

                    managerTableModel.addRow(managerRow);
                    managerTableModel.fireTableDataChanged();

                    sql = "INSERT INTO 'Manager' (sID, eID)" +
                             "VALUES (" + managerRow[0] + ", " + managerRow[1] + ");";
                    try {
                       Statement stmt1 = conn.createStatement();
                       stmt1.executeUpdate(sql);
                    }catch(SQLException e1) {
                       e1.printStackTrace();
                    }

                  break;
               case "Stores":
                  Object[] storeRow = new Object[sColAm];

                  try
                     {
                         storeRow[0] = Integer.parseInt((String)
JOptionPane.showInputDialog(frame2, "Enter the Store ID Number: ", "Enter Info",
JOptionPane.PLAIN_MESSAGE, null, null, 0));
                         storeRow[1] = (String) JOptionPane.showInputDialog(frame2, "Enter the
Store Region: ", "Enter Info", JOptionPane.PLAIN_MESSAGE, null, null, 0);
                         storeRow[2] = Integer.parseInt((String)
JOptionPane.showInputDialog(frame2, "Enter the Employee Count: ", "Enter Info",
JOptionPane.PLAIN_MESSAGE, null, null, 0));
                         storeRow[3] = Integer.parseInt((String)
JOptionPane.showInputDialog(frame2, "Enter the Game Count: ", "Enter Info",
JOptionPane.PLAIN_MESSAGE, null, null, 0));
                     }
                  catch(Exception ex5)
                     {
                         System.out.println("ERROR WITH INPUT. TRY AGAIN.");
                     }

                    storeTableModel.addRow(storeRow);
                    storeTableModel.fireTableDataChanged();

                    sql = "INSERT INTO 'Store' (sID, region, employeeCount, gameCount) " +
                             "VALUES (" + storeRow[0] + ", '" + storeRow[1] + "', " +
storeRow[2] + ", " + storeRow[3] + ");";
                    try {
                       Statement stmt1 = conn.createStatement();
                       stmt1.executeUpdate(sql);
                    }catch(SQLException e1) {
                       e1.printStackTrace();
```

```
                   }

                 break;
              case "Buyers":
                 Object[] buyerRow = new Object[bColAm];

                 try
                    {
                        buyerRow[0] = Integer.parseInt((String)
JOptionPane.showInputDialog(frame2, "Enter the Buyer ID: ", "Enter Info",
JOptionPane.PLAIN_MESSAGE, null, null, 0));
                        buyerRow[1] = (String) JOptionPane.showInputDialog(frame2, "Enter the
Buyer Name: ", "Enter Info", JOptionPane.PLAIN_MESSAGE, null, null, 0);
                        buyerRow[2] = (String) JOptionPane.showInputDialog(frame2, "Enter the
Buyer Interest: ", "Enter Info", JOptionPane.PLAIN_MESSAGE, null, null, 0);
                    }
                 catch(Exception ex6)
                    {
                        System.out.println("ERROR WITH INPUT. TRY AGAIN.");
                    }

                   buyerTableModel.addRow(buyerRow);
                   buyerTableModel.fireTableDataChanged();

                   sql = "INSERT INTO 'Buyers' (bID, name, interest) " +
                            "VALUES (" + buyerRow[0] + ", '" + buyerRow[1] + "', '" +
buyerRow[2] + "');";
                   try {
                      Statement stmt1 = conn.createStatement();
                      stmt1.executeUpdate(sql);
                   }catch(SQLException e1) {
                      e1.printStackTrace();
                   }

                 break;
              case "Rents":
                 JOptionPane.showConfirmDialog(frame2, "Please use the 'New' Option to add a
new transaction.");
                 break;

              }

           }
        }
        );

        nextButton.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
               if(sp.isShowing()) {
                  frame.remove(sp);
                  frame.add(sp1);
                  frame.setVisible(true);
                  tableChoice.setSelectedIndex(1);
               }else if(sp1.isShowing()) {
                  frame.remove(sp1);
                  frame.add(sp2);
                  frame.setVisible(true);
                  tableChoice.setSelectedIndex(2);
               }else if(sp2.isShowing()) {
```

```java
                  frame.remove(sp2);
                  frame.add(sp3);
                  frame.setVisible(true);
                  tableChoice.setSelectedIndex(3);
               }else if(sp3.isShowing()) {
                  frame.remove(sp3);
                  frame.add(sp4);
                  frame.setVisible(true);
                  tableChoice.setSelectedIndex(4);
               }else if(sp4.isShowing()) {
                  frame.remove(sp4);
                  frame.add(sp5);
                  frame.setVisible(true);
                  tableChoice.setSelectedIndex(5);
               }else if(sp5.isShowing()) {
                  frame.remove(sp5);
                  frame.add(sp);
                  frame.setVisible(true);
                  tableChoice.setSelectedIndex(0);
               }

            }
        });

        prevButton.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {

               if(sp.isShowing()) {
                  frame.remove(sp);
                  frame.add(sp5);
                  frame.setVisible(true);
                  tableChoice.setSelectedIndex(5);
               }else if(sp1.isShowing()) {
                  frame.remove(sp1);
                  frame.add(sp);
                  frame.setVisible(true);
                  tableChoice.setSelectedIndex(0);
               }else if(sp2.isShowing()) {
                  frame.remove(sp2);
                  frame.add(sp1);
                  frame.setVisible(true);
                  tableChoice.setSelectedIndex(1);
               }else if(sp3.isShowing()) {
                  frame.remove(sp3);
                  frame.add(sp2);
                  frame.setVisible(true);
                  tableChoice.setSelectedIndex(2);
               }else if(sp4.isShowing()) {
                  frame.remove(sp4);
                  frame.add(sp3);
                  frame.setVisible(true);
                  tableChoice.setSelectedIndex(3);
               }else if(sp5.isShowing()) {
                  frame.remove(sp5);
                  frame.add(sp4);
                  frame.setVisible(true);
                  tableChoice.setSelectedIndex(4);
               }
            }
        });
    }

    public static int getRemoveInfo(String choice) {

       String option = "";
       JFrame frame1 = new JFrame("Enter Information");

       switch(choice) {
        case "Games":
           option = (String) JOptionPane.showInputDialog(frame1, "Enter the game ID number to 
remove: ", "Enter Information", JOptionPane.PLAIN_MESSAGE, null, null, "Input");

            break;
        case "Employee":
           option = (String) JOptionPane.showInputDialog(frame1, "Enter the employee ID number 
to remove: ", "Enter Information", JOptionPane.PLAIN_MESSAGE, null, null, "Input");

            break;
        case "Managers":
           option = (String) JOptionPane.showInputDialog(frame1, "Enter the manager ID number 
to remove: ", "Enter Information", JOptionPane.PLAIN_MESSAGE, null, null, "Input");

            break;
        case "Stores":
           option = (String) JOptionPane.showInputDialog(frame1, "Enter the store ID number to 
remove: ", "Enter Information", JOptionPane.PLAIN_MESSAGE, null, null, "Input");

            break;
        case "Buyers":
           option = (String) JOptionPane.showInputDialog(frame1, "Enter the buyer ID number to 
remove: ", "Enter Information", JOptionPane.PLAIN_MESSAGE, null, null, "Input");

            break;
        case "Rents":
           option = (String) JOptionPane.showInputDialog(frame1, "Enter the transaction number 
to remove: ", "Enter Information", JOptionPane.PLAIN_MESSAGE, null, null, "Input");

            break;
    }

       return Integer.parseInt(option);
    }

    public static int getIndexToRemove(int toRemove, DefaultTableModel table, int column) {

       int foundIndex = 0;

       for(int i = 0; i < table.getRowCount(); i++){//For each row
          if((Integer)table.getValueAt(i, column) == toRemove){//Search the model
             foundIndex = i;
           }

       }//For loop outer

       return foundIndex;
    }

    public static void removeFromTable(String table, String column, int data) {
       String sql = "DELETE FROM " +table +" WHERE " +column +" = " +data +";";
       try {
         stmt = conn.createStatement();
         stmt.executeUpdate(sql);
      } catch (SQLException e) {
         e.printStackTrace();
      }


    }

    public static void connectDB()
    {
        try
        {
            Class.forName("org.sqlite.JDBC");
            String url = "jdbc:sqlite:./appSrc/src/gameStoreData.db";

            conn = DriverManager.getConnection(url);

            System.out.println("Connection success");
        }catch(Exception e) {
            System.out.println(e.getMessage());
            System.out.println("Connection failed");
        }
    }

    public static void updateTransaction(ResultSet Ers, ResultSet Grs, ResultSet Brs,
ResultSet result, DefaultTableModel table) {
       //adds to the transaction list and pushes the change to the database, then re-loads the
table

       ArrayList<Integer> emplID = new ArrayList<Integer>();
       ArrayList<Integer> gamID = new ArrayList<Integer>();
       ArrayList<Integer> buyID = new ArrayList<Integer>();
       int tN = 0;
       try {
          while(result.next()) {
             tN = (Integer) result.getInt("transactionNum");
          }

         while(Ers.next()) {
            emplID.add(Ers.getInt("eID"));
         }

         while(Grs.next()) {
            gamID.add(Grs.getInt("gID"));
         }

         while(Brs.next()) {
            buyID.add(Brs.getInt("bID"));
         }

      } catch (Exception e) {
         e.printStackTrace();
      }
       Integer[] EidNums = emplID.toArray(new Integer[0]);
       Integer[] gIdNums = gamID.toArray(new Integer[0]);
       Integer[] bIdNums = buyID.toArray(new Integer[0]);
       final int transNum = tN;

       JPanel panel1 = new JPanel(new FlowLayout());
       JPanel p1 = new JPanel(new FlowLayout());
       JPanel p2 = new JPanel(new FlowLayout());
       JPanel p3 = new JPanel(new FlowLayout());
       JComboBox em = new JComboBox(EidNums);
       em.setBounds(0,50,50,50);
       JLabel emLab = new JLabel("Select Employee ID");
       JComboBox ga = new JComboBox(gIdNums);
       ga.setBounds(0,50,50,50);
       JLabel gaLab = new JLabel("Select Game ID");
       JComboBox bu = new JComboBox(bIdNums);
       bu.setBounds(0,50,50,50);
       JLabel buLab = new JLabel("Select Buyer ID");

       JButton finishButton = new JButton("Save");
       finishButton.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
             //send update and remove the panel1 from frame. then return
             sql = "INSERT INTO Rent(eID, transactionNum, bID, gID) VALUES ("
+em.getSelectedItem() +", "
                             +(transNum+1) +", " +bu.getSelectedItem() +", "
+ga.getSelectedItem() +");";
             try {
                 stmt = conn.createStatement();
               stmt.executeUpdate(sql);
            } catch (SQLException e1) {
               e1.printStackTrace();
            }
             Object[] newItem = new Object[4];
             newItem[0] = em.getSelectedItem();
             newItem[1] = (transNum+1);
             newItem[2] = bu.getSelectedItem();
             newItem[3] = ga.getSelectedItem();
             table.addRow(newItem);
             table.fireTableDataChanged();
             frame.remove(panel1);
             frame.repaint();
            }
       });

       p1.add(emLab);
       p1.add(em);
       p2.add(gaLab);
       p2.add(ga);
       p3.add(buLab);
       p3.add(bu);

       panel1.add(p1);
       panel1.add(p2);
       panel1.add(p3);
       panel1.add(finishButton);
       frame.add(panel1);
       frame.setVisible(true);
    }

    public static void addBuyers(int amount)
    {
        try
        {
            conn.setAutoCommit(false);

            for (int i = 0; i < amount; i++)
            {
                stmt = conn.createStatement();

                int bID = i;
                String name = generateRandom.generateRandomName();
                String interest = generateRandom.generateRandomGenre();

                sql = "INSERT INTO 'Buyers' (bID, name, interest) " +
                        "VALUES (" + bID + ", '" + name + "', '" + interest + "');";
                stmt.executeUpdate(sql);
                stmt.close();
                conn.commit();
            }
        }
        catch(SQLException e)
        {
            System.out.println(e.getMessage());
        }
    }

    public static void addEmployees(int amount)
    {
        try
        {
            conn.setAutoCommit(false);

            for (int i = 0; i < amount; i++)
            {
                stmt = conn.createStatement();

                int eID = i;
                String name = generateRandom.generateRandomName();
                int payRate = generateRandom.generateRandomPay();
                String hireDate = generateRandom.generateRandomDate();

                sql = "INSERT INTO 'Employee' (eID, name, payRate, hireDate) " +
                        "VALUES (" + eID + ", '" + name + "', " + payRate + ", '" + hireDate +
"');";
                stmt.executeUpdate(sql);
                stmt.close();
                conn.commit();
            }
        }
        catch(SQLException e)
        {
            System.out.println(e.getMessage());
        }
    }

    public static void addGames(int amount)
    {
        try
        {
            conn.setAutoCommit(false);

            for (int i = 0; i < amount; i++)
            {
                stmt = conn.createStatement();

                int gID = i;
                String name = generateRandom.generateRandomGame();
                String genre = generateRandom.generateRandomGenre();
                String releaseDate = generateRandom.generateRandomDate();
                int price = generateRandom.generateRandomPrice();
                int sID = (int) (Math.random() * 49);

                sql = "INSERT INTO 'Games' (gID, name, genre, releaseDate, price, sID)" +
                        "VALUES (" + gID + ", '" + name + "', '" + genre + "', '" +
releaseDate + "', " + price + ", " + sID + ");";
                stmt.executeUpdate(sql);
                stmt.close();
                conn.commit();
            }
        }
        catch(SQLException e)
        {
            System.out.println(e.getMessage());
        }
    }

    public static void addManager(int amount)
    {
        int id = 0;

        try
        {
            conn.setAutoCommit(false);

            for (int i = 0; i < amount; i++)
            {
                stmt = conn.createStatement();

                int sID = (int) (Math.random() * 50);
                int eID = (int) (Math.random() * 100);

                sql = "INSERT INTO 'Manager' (sID, eID)" +
                        "VALUES (" + sID + ", " + eID + ");";
                stmt.executeUpdate(sql);
                stmt.close();
                conn.commit();
            }
        }
        catch(SQLException e)
        {
            System.out.println(e.getMessage());
        }
    }

    public void addRent(int amount)
    {
        int id = 0;

        try
        {
            conn.setAutoCommit(false);

            for (int i = 0; i < amount; i++)
            {
                stmt = conn.createStatement();

                String name = generateRandom.generateRandomName();
                sql = "INSERT INTO 'Rent' (eID, transactionNum, bID, gID)";
                stmt.executeUpdate(sql);
            }
        }
        catch(SQLException e)
        {
            System.out.println(e.getMessage());
        }
    }

    public static void addStore(int amount)
    {
        try
        {
            conn.setAutoCommit(false);

            for (int i = 0; i < amount; i++)
            {
                stmt = conn.createStatement();

                int sID = i;
                String region = generateRandom.generateRandomRegion();
                int employeeCount = (int) (Math.random() * 5) + 1;
                int gameCount = (int) (Math.random() * 250) + 1;

                sql = "INSERT INTO 'Store' (sID, region, employeeCount, gameCount) " +
                        "VALUES (" + sID + ", '" + region + "', " + employeeCount + ", " +
gameCount + ");";
                stmt.executeUpdate(sql);
                stmt.close();
                conn.commit();
            }
        }
        catch(SQLException e)
        {
            System.out.println(e.getMessage());
        }
    }
}
```

GenerateRandom.java –

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.HashMap;
import java.util.Scanner;

public class GenerateRandom
{
    public String generateRandomName()
    {
        HashMap<Integer, String> names = new HashMap<Integer, String>();
        int count = 0;

        try
        {
            File dict = new File("./appSrc/src/names.txt");
            Scanner readNames = new Scanner(dict);

            while(readNames.hasNextLine())
```

```java
            {
                names.put(count++, readNames.nextLine());
            }
        }
        catch(FileNotFoundException e)
        {
            System.out.println("File not found. Make sure file 'names.txt' is in src 
folder.");
        }

        int random1 = (int) (Math.random() * count);
        int random2 = (int) (Math.random() * count);

        return names.get(random1) + " " + names.get(random2);
    }

    public String generateRandomGenre()
    {
        HashMap<Integer, String> genres = new HashMap<Integer, String>();
        int count = 0;

        try
        {
            File dict = new File("./appSrc/src/genres.txt");
            Scanner readGenres = new Scanner(dict);

            while(readGenres.hasNextLine())
            {
                genres.put(count++, readGenres.nextLine());
            }
        }
        catch(FileNotFoundException e)
        {
            System.out.println("File not found. Make sure file 'genres.txt' is in src 
folder.");
        }

        int random = (int) (Math.random() * count);

        return genres.get(random);
    }

    public int generateRandomPay()
    {

        return (int) (Math.random() * 100000);
    }

    public String generateRandomDate()
    {
        int month = (int) (Math.random() * 11) + 1;
        int day = (int) (Math.random() * 28) + 1;
        int year = (int) (Math.random() * 2019) + 1;

        return month + "-" + day + "-" + year;
    }

    public String generateRandomRegion()
    {
        HashMap<Integer, String> regions = new HashMap<Integer, String>();
        regions.put(0, "North");
        regions.put(1, "East");
        regions.put(2, "South");
        regions.put(3, "West");

        int random = (int) (Math.random() * 3) + 1;

        return regions.get(random);
    }

    public int generateRandomPrice()
    {

        return (int) (Math.random() * 60);
    }

    public String generateRandomGame()
    {
        HashMap<Integer, String> dictionary = new HashMap<Integer, String>();
        int count = 0;

        try
        {
            File dict = new File("./appSrc/src/words_alpha.txt");
            Scanner readDict = new Scanner(dict);

            while(readDict.hasNextLine())
            {
                dictionary.put(count++, readDict.nextLine());
            }
        }
        catch(FileNotFoundException e)
        {
            System.out.println("File not found. Make sure file 'words-alpha.txt' is in src
folder.");
        }

        int random1 = (int) (Math.random() * count);
        int random2 = (int) (Math.random() * count);

        return dictionary.get(random1) + " " + dictionary.get(random2);
    }
}
```

README.md –

```
# databaseproject
#### Kaylee Moore and Jack Lindner

This CS 461 Final Project is our Game Rental Database system. The database has five tables
linked together using keys,
as well as a GUI to display and modify, delete, or add entries.

###Running the program with '1' as an argument will generate all sorts of new data but ONLY IF
THERE IS NO DATA IN THE DATABASE AT ALL.

To run this database program, simply run the DatabaseUI.java file. This will output whether
the connection to the
database was successful, then displaying a GUI with entries and buttons. The file can be found
in the folder:
```

```
./appSrc/src

The GUI will display the Games table by default.

The 'New Transaction' button will allow you to input the information
for a new transaction. The given dropdown prompts will show the available options for keys, as
all information
needs to be compatible with other tables.

The 'Delete' button will display a popup box depending on the currently displayed table asking
for the primary key
of the item you are looking to delete. Entering the correct information will delete that entry
in the table.

The 'Update' button will display a series of popups, which will gather the information for a
new entry to the currently
displayed table.

The 'Next' and 'Previous' buttons will cycle between the available tables.

The dropdown box will allow you to pick a table directly, without needing to cycle through.

The search textbox will filter the current displayed table, and only show the entries that
match the entered
information. You will need to press 'Enter' after your search query.

Below the buttons, the currently displayed table will show all entries for the database table
in question.

-- Jack Lindner and Kaylee Moore
```

When creating our database and GUI, we ran into numerous issues. Unlike other students in CS 461, we built our GUI from scratch using Java’s AWT framework. This made our application less pretty but taught me a great deal about designing GUIs for applications and connecting to databases. This also means we had to build a lot of the database interactions from scratch. We got a lot of practice in embedded SQL and table management throughout this project. The most interesting problem we had was the slight separation of the viewed tables and the database itself – the front-end tables had to be edited separately from the back-end database, which caused a lot of problems for a while. However, now all functions are synced and we got a lot more familiar with the ins and outs of database management.